Power Driven Microarchitecture Workshop

in Conjunction with ISCA 98
Barcelona, Spain

Sunday, June 28, 1998

Organizers:
Dirk Grunwald, University of Colorado
Srilatha (Bobbie) Manne, University of Colorado
Trevor Mudge, University of Michigan

Table of Contents

Power-Driven Microarchitecture Workshop

Introduction

In recent years, reducing power dissipation has become a critical design goal for many microprocessors due to portability and reliability requirements. Most of the power reduction was achieved through supply voltage reduction and process shrinks. However, there is a limit to how far supply voltages may be reduced, and the power dissipated on-chip is increasing even as process technology improves. Further advances will require not only circuit and technology improvements but new ideas in microarchitecture. This will be true not only for the obvious situation of portable computers but also for high-performance systems. It was the goal of the Power-Driven Microarchitecture Workshop to provide a forum for examining innovative architectural solutions to the power problem for processors at all levels of performance.

The Power-Driven Microarchitecture Workshop was held in conjunction with the ISCA98 conference. The response to the call for papers was outstanding, enabling us to assemble an strong program of over two dozen papers. Three invited speakers, Mark Horowitz of Stanford, and Vivek Tiwari and Doug Carmean from Intel provided a brief tutorial introduction to power issues in VLSI design, and covered existing problems and solutions in the microprocessor market. The purpose of these tutorials was to establish common ground for discussing power issues in processor design.

It is our hope that, as a result of this workshop, the level of awareness will have been raised in the architecture community about issues related to power dissipation and energy consumption. We further hope that this heightened awareness will lead to exciting new research in the area.

Dirk Grunwald
Bobbie Manne
Trevor Mudge

Bus Transition Activity

Reducing the Energy of Address and Data Buses with the Working-Zone Encoding Technique and its Effect on Multimedia Applications *

Enric Musoll
Cyrix
(National Semiconductor Corp.)
Santa Clara, CA 95052
enric@cyrix.com

Tomás Lang
Dept. of Electrical and Computer Eng.
University of California at Irvine
Irvine, CA 92697
tomas@ece.uci.edu

Jordi Cortadella
Dept. of Software
Universitat Politècnica de Catalunya
08071 Barcelona, Spain
jordic@lsi.upc.es

Abstract

The energy consumption due to I/O pins is a substantial part of the overall chip consumption. This paper gives an overview of the Working Zone Encoding (WZE) method for encoding for low power the external address and data buses, based on the conjecture that programs favor a few working zones of their address space at each instant. In such cases, the method identifies these zones and sends through the address (data) bus only the offset of this reference (data value) with respect to the previous reference (data value) to that zone, along with an identifier of the current working zone. This is combined with a one-hot encoding for the offset.

The paper then focuses on preliminary work on the following two topics:

- reduction of the effect of the WZE delay on the bus access time by overlapping this delay with the virtual to physical address translation. Although the modification to allow this overlapping might increase the bus energy, simulations of the SPEC benchmarks indicate that for a page size of 1 KB or larger the effect is negligible.
- extension of the technique for the data bus to some multimedia applications which are characterized by having packed bytes in a word. For two typical applications, the data-only data bus and data-only address bus I/O activity is reduced by 74% and 51% with respect to the unencoded case, and by 68% and 33% with respect the best of the rest of the encoding techniques.

1 Introduction

The I/O energy is a substantial fraction of the total energy consumption of a microprocessor [4], because the capacitance associated with an external pin is between one hundred and one thousand times larger than that corresponding to an internal node. Consequently, the total energy consumption decreases by reducing the number of transitions on the high-capacitance, off-chip side, although this may come at the expense of some additional transitions on the low-capacitance, on-chip side.

*This work has been partially funded by CICYT TIC 95-0419.

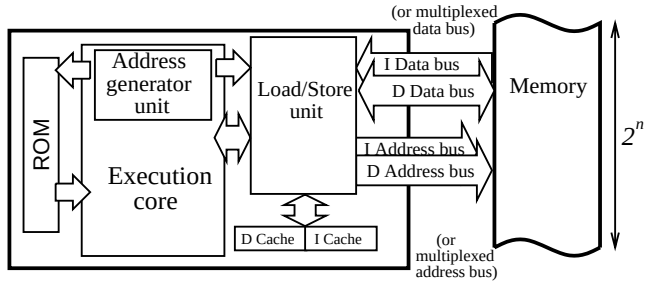


Figure 1: Bus types in a general-purpose microprocessor.

For a microprocessor chip, the main I/O pins correspond to the address and data buses. In this work, we consider an encoding to reduce the activity in both of these buses. If the value carried by n bits has to be transmitted over a bus, a reduction in the switching activity of this bus may be obtained at the cost of extra hardware in the form of an encoder on the sender device, a decoder on the receiver device, and potentially a larger number of wires m.

In [10] and [11] we have presented the Working-Zone Encoding (WZE) method, which is based on the conjecture that applications favor a few working zones of their address space. Moreover, consecutive addresses to each of these working zones frequently differ by a small amount. In such cases, an offset with respect to the previous address for that zone is sent, together with a zone identifier. The offset is encoded so as to reduce the activity of the address bus. This scheme is extended to the data bus [8] by noticing that the data for successive accesses to a working zone frequently differ also by a small amount, so that it is effective also to send the offset.

To evaluate the effectiveness of the technique in general applications, several SPEC95 streams of references to memory along with the corresponding data values were used. Among the possible bus organizations (see Figure 1), we have considered a multiplexed address bus (for instruction and data addresses) and a multiplexed instruction/data bus, with and without a unified 8K-byte direct-mapped cache. We also have compared with previously proposed encodings: Gray, bus-invert, T0, combined T0/bus-invert, inc-xor and dbm-vbm. Table 1 summarizes the results obtained. The table shows the energy reduction ratios of the WZE tech-

	Address Bus Ratio		Ins/Data Bus Ratio		Both Buses Ratio	
	vs. non encoded	vs. best of rest	vs. non encoded	vs. best of rest	vs. non encoded	vs. best of rest
Avg. (no cache)	(0.39) 0.47	(0.54) 0.66	(0.67) 0.73	(0.77) 0.84	(0.56) 0.63	(0.69) 0.78
Avg. (with cache)	(0.71) 0.87	(0.87) 1.06	(0.53) 0.63	(0.61) 0.72	(0.60) 0.72	(0.70) 0.85

Table 1: Results summary for some SPEC95 streams. Ratios are calculated as *Energy WZE/Energy other*, being *other* the unencoded case and the best of the rest of the techniques evaluated. Energy overhead of the encoder/decoder logic is only included for the WZE technique. In parenthesis, without overhead.

nique with respect to the unencoded case and to the best of the rest of the techniques, for each of the buses and for both together. We conclude that the WZE encoding significantly reduces the activity in both buses. Moreover, for the case without cache, the technique presented here outperforms the other previous bus encoding proposals for low power. On the other hand, for the case with cache the best scheme for the address bus is either the WZE presented here or bus-invert with four groups, depending on the overhead of these two techniques. In any case, the WZE method outperforms the rest of the techniques when both buses are encoded, and requires fewer additional wires than the bus-invert with four groups.

In this paper we give an overview of the WZE technique and summarize previous work on the topic; this material is similar to that of [8] and should give the reader a reasonable understanding of the method. For more details consult [10, 11]. We then focus on preliminary work on the following two topics:

- Reduction of the effect of the WZE delay on the bus access time by overlapping this delay with the virtual to physical address translation.
- Use of the WZE technique in multimedia applications, which are characterized by having packed bytes in a word. Because of the particular features of these applications, we explore the possibility of special modifications to the encoding technique for the data bus.

1.1 Previous work

Several encoding techniques for reduced bus activity have been reported, such as one-hot [6], Gray [7], bus-invert [13], T0 and combined bus-invert/T0 [5], and inc-xor and dbm-vbm [12].

One-hot encoding results in a reduced activity because only two bits toggle when the value changes. However, it requires a number of wires equal to the number of values encoded, so that it is not practical for typical buses.

Gray and T0 encoding are targeted to situations in which consecutive accesses differ by one (or by a fixed stride). The Gray encoding is useful because the encoding of these values differs by one bit. In the T0 encoding an additional wire is used to indicate the consecutive access mode, and no activity is required in the bus.

The bus-invert method [13] consists on sending either the value itself or its bit-wise complement, depending on which would result in fewer transitions. An extra wire is used to carry this polarity information. For uniform and independent distributions, this encoding technique works better when the bit-width of the value to be sent is divided into smaller groups and each one encoded independently. The bus-invert technique has been combined with T0 in [5], thus obtaining more activity reduction than each of the techniques by itself.

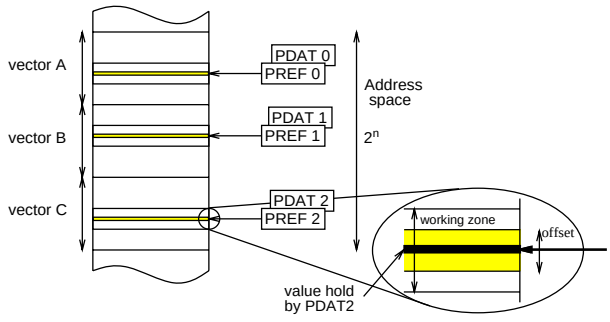


Figure 2: Address space with three vectors.

A source-coding framework is proposed in [12] as well as some specific codes. The scheme is based on obtaining a prediction function and a prediction error. This prediction error is `XOR`ed with the previous value sent to the bus so that the number of transitions is reduced in the likely case when the prediction error has a small number of ones. For addresses, the only new code proposed is the inc-xor code, in which the prediction is the previous address plus one (or any fixed stride) and the prediction error is obtained by the bit-wise `XOR` operation. This code is most beneficial for instruction address buses, where sequential addressing is prevalent. Also presented are codes which relate to the 1-hot encoding used in this paper, such as the dbm-vbm, that are applied to the data bus. In the dbm-vbm technique, the prediction is the previous address and the prediction error is obtained by a function that increases as the absolute difference between the current input and the prediction increases. Afterwards, code-words with fewer 1's are assigned to smaller error values. Finally, the result is `XOR`ed with the previous value sent to the bus.

2 Overview of the WZE technique

In this Section an overview of the WZE technique for the address bus is given along with the implementation decisions made and the rationale behind them. Afterwards, an extension of the WZE technique [8] is reviewed which allows the data bus to be encoded by reusing a large portion of the hardware already used to encode the address bus.

The basis of the WZE technique is as follows:

1. It takes into account the locality of the memory references: applications favor a few working zones of their address space at each instant. In such cases, a reference can be described by an identifier of the working zone and by an offset. This encoding is sent through the bus.

2. The offset can be specified with respect to the base address of the zone or to the previous reference to that zone. Since we want small offsets encoded in a one-hot code, the latter approach is the most convenient.

   As a simple example consider an application that works with three vectors (A, B and C) as shown in Figure 2. Memory references are often interleaved among the three vectors and frequently close to the previous reference to the vector. Thus, if both the sender and the receiver had three registers (henceforth named *Prefs*) holding a pointer to each active working zone, the sender would only need to send:

   - the offset of the current memory reference with respect to the Pref associated to the current working zone
   - an identifier of the current Pref.

3. To reduce the number of transitions, the offset is encoded in a one-hot code. Since the one-hot code produces two transitions if the previous reference was also in the one-hot code and an average of $n/2$ transitions when the previous reference is arbitrary, the number of transitions is reduced by using a transition-signaling code [14]. In this case, before sending the reference through the bus an `XOR` operation is performed with the previous value sent, always resulting in one transition.

4. One value can be sent using a 0-hot code, which with transition signaling produces zero transitions. This code should be used for the most-frequent event, which we have determined to be a repetition of the same offset for the current working zone.

5. When there is a reference that does not correspond to a working zone pointed by any Pref, it is not possible to send an offset; in such a case, the entire current memory reference is sent over the bus. Moreover, it is necessary to signal this situation.

6. In general, the total number of working zones of a program can be larger than the number supported by the hardware. Consequently, these have to be replaced dynamically. The most direct possibility is to replace an active working zone as soon as there is a miss. However, in this case any arbitrary reference would disturb an active working zone. To reduce this effect, we incorporate additional registers (henceforth named *potential working zones*) that store the references that cause a miss. Various heuristics are possible to determine when a potential working zone becomes an active one.

2.1 Implementation decisions

In the general scheme presented above, there are many aspects that have to be decided to obtain a suitable implementation. These decisions affect both the complexity of the implementation and the energy reduction achieved. Since there are many interdependent parameters, it is not practical to explore the whole space. Below we indicate the decisions made and the rationale for them.

- The number of active and potential working zones affects the number of registers and associated logic (and therefore the encoder/decoder energy consumption) and the number of values of the identifier. In the evaluation of the scheme, we have explored a range of values and determined the one that produces the largest reduction. It was determined that a small number of working zones is sufficient.

- When there is a hit to a working zone, an offset and an identifier are sent. There are choices for the set of values of the offset and the code of the identifier. Since the offset is sent in a one-hot code (with transition signaling) the set of values is directly related to the number of bits required. We have decided to use all bits of the original bus to send the offset. Moreover, we have seen that the number of hits is maximized if positive and negative offsets are used. Since all bits of the original bus are used for the offset, it is necessary to have additional wires for the identifier and, to minimize these additional wires, we use a binary code. We have considered using bits of the original bus for the identifier (thus reducing the offset bits) and have observed a significant increase in I/O activity with respect to the use of separate bits.

- When there is a miss, this situation has to be signaled to the receiver. Since in that case, all bits of the original bus are used to send the address, this hit/miss condition has to use some additional wire. As we already have decided to use additional wires for the identifier, one value on these wires might be used to signal the miss. However, this would produce a few transitions when changing from a hit to a miss. To assure only one transition, we have assigned an additional bit to signal a miss.

- The search for a hit in a working zone requires subtracting the previous address with the current one and detecting whether the offset is in the acceptable range. For the selection of which zones to check it is possible to use any of the schemes used for caches. Because of the small number of working zones, we have chosen a fully-associative search.

- There are two replacement procedures required: for the active working zones and for the potential working zones. As indicated before, when there is a miss the address is placed in a potential working zone. Since there are few of these, we use the LRU algorithm for this placement. Moreover, it is necessary to determine when a new active working zone appears and, in this case, which active working zone to replace. Among the possible alternatives, we have chosen to initiate a new active working zone when there is a hit in a potential working zone. Again, here we use the LRU replacement algorithm.

2.2 Extension to the data bus

The technique for the address bus can be extended to include also the data bus. This extension is based on the fact that in many instances the data values of consecutive accesses to a working zone differ by a small amount. If that is the case, the data can also be sent as an offset, coded in the one-hot encoding. In this case, the zero-hot encoding is used when the offset is zero.

To implement this extension, as illustrated in Figure 2, we include an additional register, called *Pdat*, per working zone. On the other hand, if the access is not to an active working zone or if the offset is larger than possible for the

Value sent over the bus (either address or data)	Transition signaling and one-hot retrieval		Receiver action (address bus)	Receiver action (data bus)
	1. XORing	*2. One-hot retrieval*		
(-) 010011	-	-	-	-
(2) 011011	001000	3	offset 3 (Pref #2)	offset 3 (Pdat #2)
(1) 011011	000000	†	same offset (Pref #1)	same data value (Pdat #1)
(1) 011001	000010	1	offset 1 (Pref #1)	offset 1 (Pdat #1)

Table 2: Example of the decoding process. Assuming always hit; () in the first column indicates working zone number.

one-hot encoding, the whole value is sent through the bus. An additional wire is required to distinguish these cases.

In addition, to further reduce the bus transitions, when the value in the data bus is not encoded by the WZE method, we use the bus-invert technique; for the address bus we saw that the benefits of using the bus-invert in this case were very small.

In summary, for the address bus, to send the offset it is necessary to compare it with the previous offset to the same working zone. The following two situations occur:

- the offsets are the same: send again the previous value sent over the bus (zero transitions)
- they are different: send the one-hot encoded value of the offset using transition signaling (one transition).

For the data bus, to send the offset it is necessary to compare the current data value with the Pdat associated to the current working zone, and the following situations occur:

- the values are the same: send again the previous value sent over the bus (zero transitions)
- they are different: send the one-hot encoded value of the offset (one transition).

The decoding of an offset in the receiver is done also in two steps: XORing the value that it receives with the previous one, and retrieving the one-hot of the result. When the XORing produces a 0 vector, the two values were the same and this is interpreted (see Table 2):

- for the address bus, as a repetition of the previous offset to that same working zone,
- for the data bus, as a repetition of the previous data value when that same working zone was last accessed

2.3 Address and data bus fields

As shown in Table 3 (next page) the encoded address and data bus consists of five fields:

- the n_a wires of the original address bus (`word_address`)
- the n_d wires of the original data bus (`word_data`)
- $\lceil \log_2(H+M) \rceil$ wires to specify one of H working zones or M potential zones (`ident`)
- one wire to indicate whether there has been a hit or a miss in any of the zones (`WZ_miss`)
- one wire to indicate if the data bus has been able to be encoded using the offset (`dbus_WZ_encoded`)
- one wire to indicate, in the case of a miss in the working zones, whether the data bus is coded with the bus-invert technique (`dbus_BI_encoded`).

Therefore, $m = n_a + n_d + \lceil \log_2(H + M) \rceil + 3$ wires are required.

3 Reducing the delay

The decoder introduces some delay in the bus access. Since this might be unacceptable, we now describe a method to overlap this delay with the virtual to physical address translation.

When an address translation is required, the most direct approach would be to perform the translation and then apply the encoding to the resulting physical address. However, this would produce an increased delay for the bus access. To reduce this delay we propose the following modification of the WZE technique:

- Use the virtual address to determine whether there is a hit in a working zone. To do this, each Pref contains the virtual address of the previous access to the corresponding zone.

Since, as it is well known, the translation modifies only the most-significant bits of the address (the page number) but keeps unaltered the least-significant bits (the page offset), this procedure is correct as long as the offset does not cross page boundaries. Consequently, it is necessary to detect when a change of page occurs and, in that case, the access is not treated as an offset.

Since the data value is not translated, this is encoded as in the original method.

Because of this modification in the WZE technique, the reduction in energy might be affected. This is because now we do not use offsets which cross a page boundary. On the other hand, the internal energy overhead might be reduced because now the detection of offset uses only the page-offset bits. The simulation of the SPEC benchmarks indicate that for a page size of 1 KB or larger the effect is negligible. We observe this in Table 4, where the I/O transitions per reference on the multiplexed address bus for the `gcc` benchmark (with unified cache) and for different page sizes is shown.

Page size	I/O transitions/reference (address bus)
(no pages)	3.26
4K bytes	3.26
1K bytes	3.29
256 bytes	3.36
64 bytes	3.72
1 byte	5.32

Table 4: Effect of the pages on the activity reduction if the WZE delay is overlapped with the virtual to physical address translation. For pages larger than 1K bytes the effect is negligible. Some unrealistic page sizes are also shown for comparison purposes. The data is for the `gcc` benchmark with unified cache.

	m-wire encoded address and data bus					
	`WZ_miss` (1 wire)	`ident` ($\lceil log_2(H+M) \rceil$ wires)	`word_address` (n_a wires)	`dbus_WZ_encoded` (1 wire)	`dbus_BI_encoded` (1 wire)	`word_data` (n_d wires)
WZ **format**	0	WZ index	offset or last address value	1	don't care	offset or last data value
				0	1	$BI_{G=1}$(data value)
					0	complete data
Non WZ **format**	1	don't care	complete address	don't care	1	$BI_{G=1}$(data value)
					0	complete data

Table 3: Information assigned to each of the several fields of the encoded address and data bus when there is a hit (WZ format) and a miss (Non WZ format) in the H working zones and in the M potential working zones. The number of wires for each field is also shown.

4 Use in Multimedia Applications

In this Section we modify the WZE technique for the data bus for those multimedia applications that use data workloads composed of packets of data that may be brought from memory several in the same word. Examples of these are the image processing applications and we will evaluate the WZE technique for two particular examples. We show the results for the data-only data bus and the data-only address bus (see Figure 1). The code of the applications is assumed to be stored in an internal ROM (therefore no references to instructions are sent through the address bus and no instructions are fetched through the data bus).

In multimedia applications images are composed of pixels. These pixels consist of one or a few components, each with a relatively small number of values (for instance, in the examples we illustrate each pixel consists of three colors and each color can have 256 values). In such case, these pixels can be stored in one word which is subdivided into subwords for each component. In many applications, these components are accessed and processed simultaneously. We now describe how we modify the WZE technique for this situation.

The address bus encoding part of the technique is not modified, since the locality of reference is even more apparent in these applications with images. For the data part we do the following:

- Instead of using an offset for the whole word, we encode each of the bytes using the byte offset. In this way, the one-hot encoding allows eight possible offsets. We have determined that this number of offsets (which would correspond to offsets from -4 to +4) is insufficient to capture a significant portion of the data. Consequently, it is convenient to extend the encoding to include also k-hot encodings for $k > 1$.
- Moreover, if we allow two possibilities for each byte, namely that the data satisfies the offset range (a hit) or not (a miss), we would need individual wires per byte to indicate this hit/miss. Since this would be a significant wire overhead, we decided to code every data byte as an offset, no matter how large this offset is. Moreover, since a k-hot encoding with transition signaling generates k transitions in the bus, to reduce the average number of transitions we encode the offsets into a k-hot code, with smaller value of k for the more frequent offsets (that is, the value of k increases as the absolute value of the offset increases).
- When there is an address miss (that is, the address does not correspond to an active WZ), the most direct solution is to send the unencoded data value. However, we have found that the variation in the values of images is small enough so that it is better to send the offsets with respect to the last data value.

Note that in this modification of the WZE technique, the two extra wires (`dbus_WZ_encoded` and `dbus_BI_encoded` in Table 3) are not needed, since the data bus is always encoded and no bus-invert is done.

We have done some preliminary evaluations for two applications: *image alpha blending* and *motion estimation*. Both use color images where a pixel is composed of three bytes that specify the three main colors; a pixel is read or written per memory reference.

4.1 Image Alpha Blending

Alpha Blending [1] is used for imaging effects to merge two images together, weighting one image more than the other. Thus, alpha blending may be used for fading from one image to another and this is the case shown here: given two images of two different human faces, six images are generated so that the first image is transformed into the second one. This is done for three different sets of human faces [2].

The results are reported in Table 5, where the WZE technique is compared to the unencoded case and to the best of the rest of the techniques (bus-invert with three groups, one per byte). The WZE technique uses four active and two potential working zones.

4.2 Motion Estimation

The Motion-estimation algorithm [9] is used in video transmission to lower the bandwidth of the network where the video is being transmitted. The frame to be transmitted is divided into blocks which are compared to several blocks in the previous frame and the best match is selected.

The basic motion-estimation algorithm is applied to two different sets of images in motion [3, 2] (weather satellite, human face and football images). Results are shown in Table 6; the best of the rest of the techniques is the dbm-vbm with three groups. The WZE technique uses three active working zones and no potential working zones.

Images	**Data-only Data Bus**		
	non encoded	**WZE**	**$BI_{G=3}$**
`claire` → `missa`	10.07	3.00	8.68
`claire` → `susie`	10.68	3.23	8.58
`missa` → `susie`	10.43	3.83	8.97

Table 5: Data-only data bus I/O transitions per reference for the alpha-blending application. The best of the rest of the techniques is the bus-invert with three groups.

Application	Data-only Data Bus Ratio		Data-only Address Bus Ratio		Both Buses Ratio	
	vs. non encoded	vs. best of rest	vs. non encoded	vs. best of rest	vs. non encoded	vs. best of rest
Image Blending	0.32	0.38	0.17	0.23	0.26	0.32
Motion Estimation	0.58	0.81	0.36	0.41	0.49	0.67

Table 8: Results summary for the two multimedia applications. Smaller ratio means fewer I/O transitions.

Image Sequences	Data-only Data Bus		
	non encoded	WZE	dbm-vbm$_{G=3}$
`weather satellite`	10.05	5.93	7.58
`human face`	9.13	5.14	5.89
`football`	10.79	6.22	7.79

Table 6: Data-only data bus I/O transitions per reference for the motion-estimation application. The best of the rest of the techniques is the dbm-vbm with three groups.

Application	Data-only Address Bus		
	non encoded	WZE	best of rest
Image Blending	7.78	1.33	5.78
Motion Estimation	6.71	2.40	7.21

Table 7: Data-only address bus I/O transitions per reference for the two example multimedia applications.

Table 7 shows the data-only address bus results for both applications. For the image blending the best of the rest of the techniques is the dbm-vbm whereas for the motion estimation is the bus-invert. In any case, the WZE technique clearly outperforms any previously proposed technique.

The averaged results for both applications are shown in Table 8. We show the ratio of the I/O transitions with respect the unencoded case and to the best of the rest of the techniques for the data-only data bus, data-only address bus and both buses. The I/O activity when coding both buses is reduced by 74% and 51% with respect the unencoded case, and by 68% and 33% with respect the best of the rest of the encoding techniques.

5 Conclusions

This paper gives an overview the Working Zone Encoding (WZE) method for encoding an external address and data bus, based on the conjecture that programs favor a few working zones of their address space at each instant.

For the general-purpose microprocessor multiplexed address bus and multiplexed instruction/data bus (with unified cache), the WZE significantly reduces the I/O activity, around 30% with respect to the unencoded case and 15% with respect the best of the rest of the encoding techniques. When no caches are present, the reductions are even larger.

For two multimedia applications using images as the workload, the activity in the data-only data bus and the data-only address bus is reduced by 74% and 51% with respect the unencoded case, and by 68% and 33% with respect the best of the rest of the encoding techniques. These results are promising and make it worthwhile to pursue the development of this approach.

References

[1] http://developer.intel.com/drg/mmx/appnotes/ap554.htm.

[2] http://ipl.rpi.edu/sequences/sequences.html.

[3] http://meteosat.e-technik.uni-ulm.de/meteosat.

[4] H. Bakoglu. *Circuits, Interconnections and Packaging for VLSI.* Menlo Park, CA, 1990.

[5] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano. Address bus encoding techniques for system-level power optimization. In *Design, Automation and Test in Europe*, pages 861–866, February 1998.

[6] A.P. Chandrakasan and R.W. Brodersen. *Low Power Digital CMOS Design.* Kluwer Academic Publishers, 1995.

[7] R.M. Owens H. Mehta and M.J. Irwin. Some issues in gray code addressing. In *Great Lakes Symposium on VLSI*, pages 178–180, March 1996.

[8] T. Lang, J. Cortadella, and E. Musoll. Extension of the Working-Zone Encoding method to reduce the energy on the microprocessor data bus. Technical Report http://www.eng.uci.edu/numlab/archive/pub/nl98b/02.ps.Z, University of California, Irvine, May 1998. To be published in the next Int. Conference on Computer Design (ICCD'98).

[9] C. Lin and S. Kwatra. An adaptive algorithm for motion compensated colour image coding. *IEEE Globecom*, 1984.

[10] E. Musoll, T. Lang, and J. Cortadella. Exploiting the locality of memory references to reduce the address bus energy. In *Int. Symp. on Low Power Design and Electronics*, pages 202–207, August 1997.

[11] E. Musoll, T. Lang, and J. Cortadella. Working-Zone Encoding for reducing the energy in microprocessor address buses. Technical Report http://www.eng.uci.edu/numlab/archive/pub/nl98b/01.ps.Z, University of California, Irvine, March 1998. To be published in the next special issue of Transactions on VLSI on low-power electronics and design.

[12] S. Ramprasad, N.R. Shanbhag, and I.N. Hajj. Coding for low-power address and data busses: a source-coding framework and applications. In *Proc. of the Int. Conf. on VLSI Design*, pages 18–23, January 1998.

[13] M.R. Stan and W.P. Burleson. Bus-invert coding for low power I/O. *IEEE Trans. on VLSI Syst.*, pages 49–58, 1995.

[14] M.R. Stan and W.P. Burleson. Low-power encodings for global communications in CMOS VLSI. *IEEE Trans. on VLSI Syst.*, pages 444–455, 1997.

Reduced Address Bus Switching with Gray PC

FORREST JENSEN
AKHILESH TYAGI*

Iowa State University
Ames, IA 50011

Abstract

Reduced switching on the address bus saves on the energy incurred in PC increment, the propagation of PC value across the pipeline, instruction cache access, and memory access. We estimate the savings in bit switchings in PC by adopting the Gray sequence for program sequencing. There is a 40% reduction in address bit switchings with a Gray PC over the traditional lexicographic PC over a collection of SPEC '95 integer and FP benchmarks. We also assess the adverse impact of Gray PC on the other processor units, particularly I-cache locality. The cache miss rates are indistinguishable between the Gray and lexicographic sequencing. These experiments were conducted with the SimpleScalar tool set. We also propose a design for a Gray counter for the PC. We have developed optimization algorithms for the loader so that the expected address bus switching is minimized.

1 Introduction

The need for reduced energy in processors has been emphasized elsewhere [Sin94]. Portable computing platforms (general purpose computing engines such as laptops or special purpose embedded processors) strive to reduce the energy of computation in order to prolong the battery life. The packaging and heat dissipation limitations are another driving force behind the low energy processor architecture and implementation trend.

In this paper, we assess the effectiveness of non-lexicographic instruction ordering, in particular Gray ordering, in reducing address bus switching and system energy. There are several architecture level side-effects of Gray address ordering that need to be evaluated. Specifically, an adverse impact on cache locality can negate any energy gains derived from the lower address bus switching.

This change in the implementation is transparent to the software. The only system component that needs to be modified is the loader. A simplistic loader can load an instruction originally at address A at the address $gray(A)$. However, this poses a new optimization problem for the loader, to minimize the expected switching over the program graph. We provide some algorithms for this optimization problem as well, which have not been implemented yet (Section 4).

Note that we will use terms *energy* and *power* interchangeably since for a processor power is just the energy per cycle times the clock frequency. Hence any technique not explicitly changing clock frequency affects both energy and power identically. What fraction of processor and system energy is affected by reduced address bus switching? Burd and Peters [BP94] and Gonzalez and Horowitz [GH96] have profiled the energy distribution of MIPS R3000 processor. Both the studies are based on simulations of a VLSI design for the MIPS R3000 architecture. Burd and Peters [BP94] compute the switched capacitance per cycle as an estimate of energy. The switching frequency of capacitances depends on the control signal, instruction and data correlations. They simulate the MIPS R3000 architecture with several real benchmark programs (from SPECint '92 benchmark suite) to derive a probability for switching for all the internal capacitive nodes. The total expected switched capacitance per clock cycle is 317 pF. The breakdown is as follows: *instruction cache (I-Cache) memory: 30%; I-Cache control: 15%; Datapath: 28% with register file accounting for 10%; and ALU & shifter for another 12%; Global buses: 3-4%; Controller: 10%; DCache: 8-9%*. The program counter logic (next-PC) takes up another $\approx 4\%$ energy. Note that the proposed Gray PC (program counter counting in Gray sequence) reduces energy for ICache address decoder and PC logic (and some for global buses) which accounts for $\approx 21\%$ processor energy.

These numbers did not include the switched capacitance for off-chip memory (address bus on the backplane). Assuming reasonable cache *hit rates*, and given a 25pF external load, the external switched capacitance per cycle is 272 pF! This is almost as high as the processor average switched capacitance of 317 pF. Hence, the proposed Gray PC can potentially have a bigger impact on the input/output energy than on the intra-processor energy. We discuss in Section 5 potential impact of Gray PC on the L1 instruction cache and L2 cache interface.

The literature on logic design for a Gray counter and/or adder is sparse. We propose a design for a Gray counter in Section 3. This design is being implemented so that the counting energy of a Gray PC and a lexicographic PC can be compared. Doran [Dor93] is the most comprehensive reference on Gray adder design. Altera corporation [Cor94] has an application brief dealing with a Gray counter design. Note that in the rest of the paper, we refer to binary reflected Gray code sequence by the term Gray sequence. We also discuss a design for instruction cache decoder in order to better exploit the low Hamming distances of Gray sequences in Section 5. Section 2 describes the experimental setup

*This work was supported in part by NSF grant #MIP9703702.

and results.

Su *et al.* [STD94] also use a Gray PC. They report on the average address bus switching on an internal set of programs, but do not consider other aspects of Gray PC.

2 Experiments and Results

Burd & Peters [BP94] note that the average switching per program counter (PC) bit is a little over 7% over a variety of SPECint '92 benchmarks. Hence, the expected number of bit switchings over a 32 bit address are 2.5. This is consistent with the observation that an average basic block's size is 4-5 instructions, which accounts for about 2.3 least-significant bit switchings. The remaining switching comes with 20-25% frequency when a non-sequential instruction such as a branch or procedure call is encountered. Gray PC ensures that within a basic block (and between basic blocks as well, as often as possible), exactly one address bit switches through the use of Gray sequencing in the PC. Assuming, we can ensure Gray sequencing within all basic blocks, the expected address bus switching based on back-of-the-envelope calculations would be approximately 1.2 ($.2$ accounts for switching due to non-sequential instructions).

The cache placement of instructions is based on the lexicographic addresses. The locality of program execution is exploited to achieve high hit rates. The cache blocks and sets are organized on the basis of contiguous bits. For instance, the least significant k_0 bits give block-offset, the next k_1 bits give the set address and the remaining bits give the tag. This type of mapping reflects locality based on lexicographic ordering. It is possible that two instructions residing in conflicting blocks in lexicographic ordering may not conflict in Gray ordering or *vice versa*. Hence we need to calibrate the *hit rates* of the instruction cache under lexicographic and Gray PCs. If the hit rate goes down significantly with a Gray PC, the increased energy cost of memory accesses might offset any gains derived from reduced address bus switching.

To determine the number of PC bits that switch in a Gray code PC versus a lexicographic PC and to determine any difference in cache performance between the two encoding methods, we conducted our experimental work with a modified version of *sim-cache*, one of the simulation tools available in the SimpleScalar suite of simulators [BAB96]. In addition to determining the total number of switched bits with each encoding method, we also determined the number of times each of the eleven least significant bits of the PC switch. Sim-cache uses a 32 bit address to access 8 byte instructions. As a result, the three least significant bits are always zero. For the remainder of this discussion we ignore bits 0-2 and think of bits 3-10 as the eight least significant bits. To measure differences in the cache performance, we calculated miss rate for the level 1 instruction cache. For purposes of simulation, we maintained the PC as a lexicographic binary number. We incremented the counter and added offsets for jumps and branches in the usual fashion. Before each instruction memory access, including program load and instruction fetch, we converted the PC value to Gray. With SPEC95 integer and FP benchmarks, a Gray code PC produced approximately 40% fewer instruction address bit transitions. Note that the address bus transition frequency is similar over both the integer and FP benchmarks. In Table 1 we present the address bit switching results. For both lexicographic and Gray encoding we show the average number of switched bits per instruction for the entire address and also for the eight least significant bits. The final column shows the percentage change in the total number of switched bits when going from lexicographic encoding to Gray encoding.

The reduction in instruction address bit switches confirms the observations made by Su *et. al*. In addition, we observe that the eight least significant bits account for 90-93% of all switched bits. This suggests the possibility of implementing a PC that is partially lexicographic and partially Gray. The level 1 instruction cache miss rates produced by the two encoding methods were identical. We present these values in Table 2. We used separate level 1 instruction and data caches. Both were 8 KB with 256, 32-byte direct mapped blocks. The 1 MB unified level 2 instruction and data cache had 4096, 4-way associative sets. We used 64 KB blocks and the LRU replacement scheme.

3 Gray Counter Design

There are two general operations performed on the contents of the PC, increment and addition of an offset for jump or branch instruction. We begin by presenting a Gray adder and follow with a Gray counter, which is a specialized adder. Conceptually, addition of Gray numbers is performed by first converting from Gray to lexicographic, adding the numbers, and then converting back to Gray. The disadvantage of this approach is that the Gray to lexicographic conversion is performed beginning with the MSB and working towards the LSB, while addition is performed in the opposite direction. The result is that two passes must be made through the bits instead of the usual single pass. To reduce the time necessary for the Gray to lexicographic conversion, we limit the Gray implementation to the eight least significant bits of the PC and use lexicographic for the remaining high order bits. The interface between the Gray and lexicographic sections is trivial. In addition, we include a parity bit that indicates the parity of the Gray portion of the PC and allows us to convert from Gray to lexicographic beginning with the LSB. The Gray to lexicographic conversion, rather than the addition and conversion back to Gray, is still the limiting factor of this design. We have therefore divided the Gray portion into two nibbles. We convert the high order nibble to lexicographic beginning with its MSB, progressing towards the LSB. For the low order nibble, we work in the opposite direction beginning with its LSB. The conversion to lexicographic of the two nibbles is therefore performed in parallel. Unlike Su *et. al.*, we do not modify the offsets before loading the program into memory. As a result, we need to convert only the PC to lexicographic before adding. Although this does not affect the speed of the conversion, it does reduce the complexity of the adder. For lexicographic addition, a full adder generates sum and carry bits. These are calculated with the following equations:

$s_i = a_i \oplus b_i \oplus c_i$

$c_{i+1} = a_i b_i + c_i(a_i + b_i)$

The symbols are understood as follows:

a_i and b_i are the two bits being added. s_i is the sum

Benchmark	Instr. count	Lexicographic		Gray		% change
		Switches per inst	Bits 3-10	Switches per inst	Bits 3-10	
int:						
cc1	264,897,677	2.3034	2.1832	1.4222	1.3126	-38.25
li	173,965,506	2.3358	2.1718	1.4246	1.3217	-39.01
go	132,917,038	2.2685	2.1504	1.3195	1.2338	-41.83
compress95	35,684,602	2.2812	2.1939	1.3860	1.3044	-39.24
m88ksim	494,917,870	2.3333	2.2338	1.3347	1.2681	-42.80
vortex	404,996	2.2661	2.1858	1.3242	1.2519	-41.56
FP:						
swim	796,527,564	2.2343	2.1244	1.2327	1.1241	-44.83
wave5	4,515,144,715	2.1081	2.0595	1.1763	1.1538	-44.20

Table 1: Average number of switched bits per instruction for lexicographic and Gray program counters

Benchmark	Lexicographic	Gray
int:		
cc1	0.0986	0.0986
li	0.0260	0.0260
go	0.0854	0.0854
compress95	0.0005	0.0005
m88ksim	0.0867	0.0867
vortex	0.0715	0.0715
FP:		
swim	0.0132	0.0132
wave5	0.0179	0.0179

Table 2: Level 1 Instruction Cache Miss Rates

produced at position i. c_i and c_{i+1} are the carry in and carry out of position i.

c_0 = 0 for addition and c_0 = 1 for subtraction.

Figure 1 illustrates the dataflow within a Gray adder. In the top sequence (Figure 1 (a)) the conversion to lexicographic is completed before the addition can begin. In the middle sequence (Figure 1 (b)), the conversion to lexicographic is right to left, the addition runs in parallel but slightly behind. In the bottom sequence (Figure 1 (c)), the conversion to lexicographic of the two nibbles is performed in parallel. The addition follows behind the low order nibble conversion but still finishes ahead because it is not delayed by the conversion of the high order nibble.

The equations for a Gray adder are more complicated. This reflects the need for code conversions before and after the addition. For the Gray adder, we are adding a lexicographic number to a Gray number. In the following equations we represent the bits of the lexicographic value with a, the Gray value with g, and the lexicographic equivalent of the Gray value with b. We use p to represent the parity bit. In addition to sum and carry bits for each position, we must also determine the new value of p. These are calculated with the following equations:

$s_i = a_i \oplus a_{i+1} \oplus c_i \oplus c_{i+1} \oplus g_i$

$s_7 = a_7 \oplus a_8 \oplus c_7 \oplus c_8 \oplus b_7 \oplus b_8$

$c_{i+1} = a_i b_i + c_i(a_i + b_i)$

$p_{new} = a_0 \oplus c_0 \oplus p_{old}$

c_0 = 0 for addition and c_0 = 1 for subtraction. We use a different equation for s_7 because b_7 is not derived from b_8.

For the low order nibble

$b_i = b_{i-1} \oplus g_{i-1}$

$b_0 = p$

For the high order nibble

$b_i = b_{i+1} \oplus g_i$

$b_7 = g_7$

Note that we have assumed the bits are labelled 0-7 with 0 being the LSB, so g_7 is the MSB of the Gray number and c_8 is the carry out of the most significant position. c_8 serves as the carry in for the lexicographic adder used to sum the higher order bits. It is possible with a Gray adder, as with a lexicographic adder, to implement carry lookahead across a nibble. The only difference is that we must convert to lexicographic before computing the carry generate and carry propagate signals.

With a lexicographic counter, increment is generally performed by a half-adder. This is possible because, for increment, the number being added to the PC consists of a 1 for the LSB and 0's for all higher bits. Because of this, the equations for both lexicographic and Gray adders are a simplification of those given above. For a lexicographic counter, the equations are as follows:

$s_i = b_i \oplus c_i$

$c_{i+1} = c_i b_i$

c_0 = 1

For the Gray counter, they are as follows:

$s_i = (c_i \overline{b_i}) \oplus g_i$

$c_{i+1} = c_i b_i$

c_0 = 1

$p_{new} = \overline{p_{old}}$

For the low order nibble

$b_i = b_{i-1} \oplus g_{i-1}$

$b_0 = p$

For the high order nibble

$b_i = b_{i+1} \oplus g_i$

$b_7 = g_7$

4 Loader

Loader's task is to assign instructions from the binary executable to the memory address space. Traditionally, it is a straightforward task since the mapping from the original program sequence to the address sequence is an identity (with a linear shift). However, with Gray sequencing, the loader must perform extra work.

A simple loader can take an instruction from address i in the program (ith instruction in the static program order) and load it at address $base + gray(i)$ where $base$ is the base address of the text segment, and $gray(i)$ is

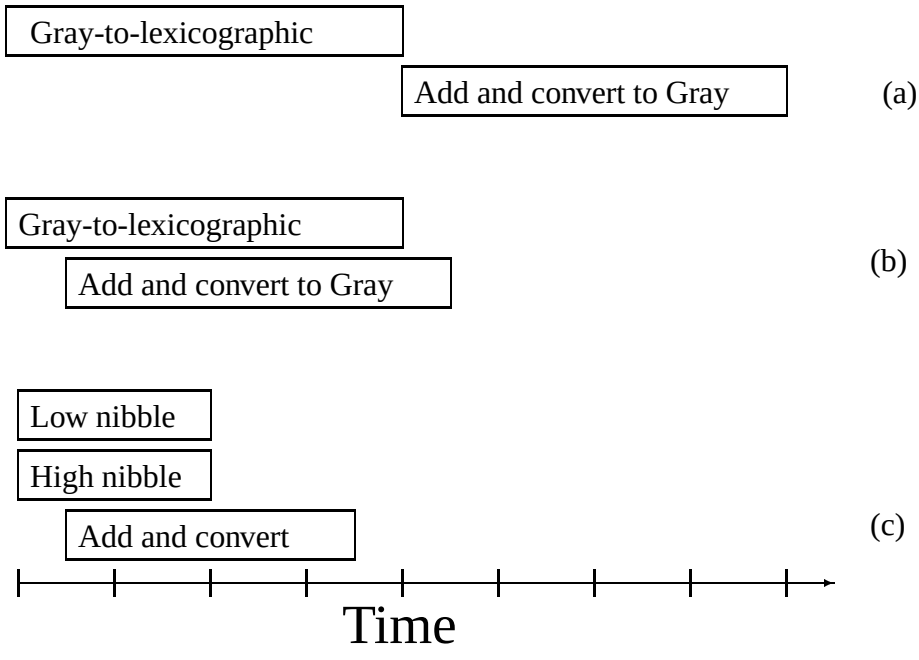


Figure 1: Illustration of Dataflow in Gray Counter

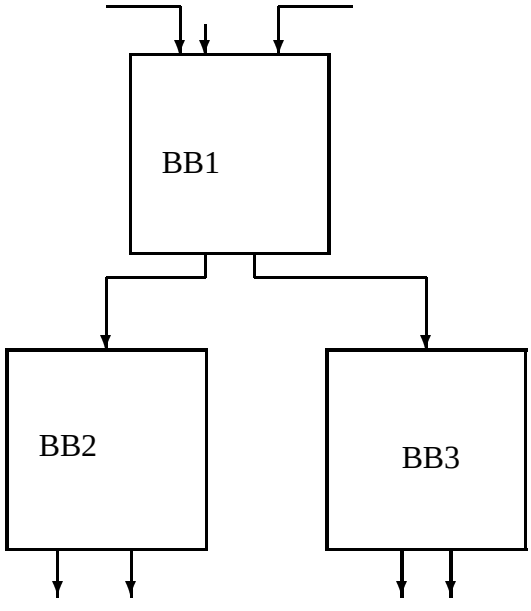


Figure 2: Program Flow Graph with Basic Block Nodes

the binary string in the ith position in a Gray sequence. This is what we have currently implemented.

However, there is a bigger opportunity for address bit switching reduction at the loader stage. The simplistic approach outlined above results in a single bit switch for the program flow within a basic block. The number of switched bits is not controlled (is comparable to the lexicographic PC scheme) for control flow out of basic blocks (inter basic block transitions). An example of a program flow graph is shown in Figure 2 where nodes represent basic blocks. Each of the edges can be assigned a probability based on static program profiling. Let $p(BB_i, BB_j)$ be the probability of the transition from the basic block BB_i to the basic block BB_j. Let the address assigned to a basic block BB be denoted by $A(BB)$. Let $h(x,y)$ be the Hamming distance of two binary strings x and y. Then the following optimization problem models the loader's task.

Input: A directed graph.

Objective: Find a memory address (for relocatability, a relative address) $A(BB)$ for each Basic block BB such that

$$\sum_{\text{over all BB}_i, \text{BB}_j} h(A(BB_i), A(BB_j)) * p(BB_i, BB_j)$$

is minimized.

If profiling is not feasible, we can minimize $\sum_{\text{over all BB}_i, \text{BB}_j} h(A(BB_i), A(BB_j))$.

We have represented the Gray chain of addresses assigned to the instruction in a basic block BB by a single address $A(BB)$ in order to bring out the similarities in this optimization problem and the low energy/power state machine synthesis. A common formulation of state assignment problem for low energy is to minimize $\sum_{s_i, s_j \in S} h(s_i, s_j) * p(s_i, s_j)$, where $p(s_i, s_j)$ is the steady state probability of the transition between states s_i and s_j, $h(s_i, s_j)$ is the Hamming distance between the codes for the two states. We have developed several algorithms [Tya96], [SCT97] for low power state assignment. Our intent is to modify one of them [Tya96] to handle the address assignment problem for the loader. The main problem with this approach is the size of the input problem. The state assignment methods take time typically close to quadratic (or close to $N^2 \log N$) in number of states N. A typical state machine has about 50-100 states. These approaches can turn out to be impractically expensive for large programs with hundreds of thousands of basic blocks. Hence, we

are also looking into heuristics linear in the number of basic blocks. Dichotomy based state assignment methods such as [TPCD94] may be adaptable for an efficient loader algorithm.

5 Future Work and Conclusions

The work reported is still preliminary. There are several interesting threads that we plan to pursue in the future. The reduced address bus switching derived from Gray sequencing is directly visible at the I-cache. However, it is the L1 and L2 cache/memory interface that provides high capacitances for the address bus. But the addresses that appear at the L1/L2 interface are already filtered based on the cache locality characteristics. Are the Gray address sequences any better than the lexicographic address sequences at the L1/L2 interface? Are there any additional constraints on the loader optimization that can reduce the Hamming distance of upper address bits for the sets of addresses likely to collide in the cache? An additional complexity is that at the L1/L2 interface the instruction and data addresses are unified. This effect will further increase the average Hamming distance. Is Harvard architecture a better choice in order to separate the instruction and data data address sequences and retain their low hamming distances?

The decoder in the instruction cache accounts for a significant fraction of energy consumed in the instruction cache [BP94]. However, a typical decoder design is precharged. We plan to experiment with a hybrid static and dynamic decoder design that leverages the low hamming distance in the addresses to reduce energy.

References

[BAB96] D. Burger, T. M. Austin, and S. Bennett. Evaluating Future Microprocessors: The SimpleScalar Tool Set. Technical Report CS-TR-96-1308, University of Wisconsin, Madison, 1996.

[BP94] T. D. Burd and B. Peters. A Power Analysis of a Microprocessor: A Study of an Implementation of the MIPS R3000 Architecture. Technical report, ERL, University of California, Berkeley, 1994. URL: http://infopad.EECS.Berkeley.edu/∼burd/gpp/gpp.html#published_papers.

[Cor94] Altera Corporation. Ripple-Carry Gray Code Counters in FLEX 8000 Devices. Technical Report Application Brief 135, Altera Corporation, May 1994.

[Dor93] R. W. Doran. The Gray Code. Technical report, Dept. of Computer Science, University of Auckland, 1993. URL: http://www.cs.aucland.ac.nz/∼techrep/TR131.

[GH96] R. Gonzalez and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid State Circuits*, 31:1277–1284, 1996.

[SCT97] P. Surthi, L. Chao, and A. Tyagi. Low-Power FSM Design using Huffman-Style Encoding. In *Proc. of European Design and Test Conference, IEEE Computer Society Press*, pages 521–525, 1997.

[Sin94] D. Singh. Prospects for Low-Power Microprocessor Design. talk delivered at ACM/IEEE International Workshop on Low Power Design, Napa Valley, CA, 1994.

[STD94] Ching-Long Su, Chi-Ying Tsui, and Alvin Despain. Low power architectural design and compilation techniques for high-performance processor. In *'Proceedings of COMPCON 94*, pages 489–498, February 1994.

[TPCD94] C. Y. Tsui, M. Pedram, C. Chen, and A. M. Despain. Low Power State Assignment Targeting Two- and Multi-level Logic Implementations. In *Proc. of ICCAD*, pages 82–87. ACM/IEEE, 1994.

[Tya96] A. Tyagi. Integrated Area-Power Optimal State Assignment. In *Proceedings of the Synthesis and System Integration of Mixed Technologies, SASIMI '96*, pages 24–31. SASIMI, Seiei Insatsu, Osaka, Japan, November 1996.

Instruction Scheduling for Low Power Dissipation in High Performance Microprocessors

Mark C. Toburen Thomas M. Conte
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina 27695-7911
{mctobure, conte}@eos.ncsu.edu

Matt Reilly
Digital Equipment Corporation
Shrewsbury, Massachusetts
reilly@rock.enet.dec.com

Abstract

Power dissipation is rapidly becoming a major design concern for companies in the high-end microprocessor market. The problem now is that designers are reaching the limits of circuit and mechanical techniques for reducing power dissipation. Hence, we must turn our attention to architectural approaches to solving this problem. In this work we propose a method of instruction scheduling which limits the number of instructions which can be scheduled in a given cycle based on some predefined per cycle energy dissipation threshold. Through the use of a machine description [8], [9] we are able to define a specific processor architecture and along with that an energy dissipation value associated with each functional unit defined therein. Through careful inspection, we can define the cycle threshold such that the maximal amount of energy dissipation can be saved for a given program while incurring little to no performance impact.

1 Introduction

Power dissipation is becoming a vital design issue in today's high-end microprocessor industry. Two examples of high-end processors that suffer from high levels of power dissipation are the DEC 21164a and 21264. The 21164a runs at a clock speed of 600 MHz while dissipating 45.5 Watts of power. The 21264 suffers even worse with internal clock speeds which can reach 666 MHz while dissipating 72 Watts. The problem now is that as power dissipation continues to rise, we are rapidly approaching a point where we will be forced to use cooling techniques which are not suitable for today's personal computers, workstations, and low-end to mid-range servers such as liquid immersion or jet impingement. Circuit designers and thermal engineers have produced some excellent techniques for keeping power dissipation to a minimum in recent years. Clock gating, power supply reduction, smaller process technology, and state of the art packaging are all examples of the approaches that have been used to date to eliminate the power dissipation problem. However, these approaches are reaching their limitations as the demand for processors with higher clock speeds and denser transistor counts continues to rise.

In this work we propose an architectural/compiler approach towards solving this problem. One way we can reduce peak power dissipation is by preventing the occurrence of current spikes during program execution. If a region of code has a heavy profile weight and contains one or more instructions which require significantly more energy than others, then the execution of this region will lead to repeated current spikes in the processor which results in increased power dissipation. Our goal is to prevent this from occurring by limiting the amount of energy that can be dissipated in any given cycle. Because of schedule slack, this often results in little or no performance impact. In our scheduling model, we schedule as many instructions as possible in a given cycle until the cycle threshold is violated. Once that point is reached, we move on to the next cycle and resume scheduling with the instruction that caused the violation in the previous cycle.

1.1 Previous Work

There have been previous attempts at using scheduling techniques to reduce total power consumption. Su, Tsui, and Despain proposed a technique which combined Gray code addressing and a method called *cold scheduling* to reduce the amount of switching activity in the control path of high performance processors, [1]. Used in conjunction with the traditional list scheduling algorithm, cold scheduling schedules instructions in the ready list based on highest priority. Priority of an instruction is determined by the power cost when the instruction in question is scheduled following the last instruction. The power cost is taken from a power cost table

which holds power entries, S(I,J), corresponding to the switching activity caused by the execution of instruction I followed by instruction J. Instructions in the ready list with lower power costs have higher priority. After each instruction is scheduled, the power cost of the remaining instructions in the ready list has to be recalculated before scheduling the next instruction. The drawbacks to the cold scheduling approach are obvious. First, a large table is required to hold power costs for all possible instruction combinations. For a high performance processor with a complex instruction set, this table can be extremely large. Second, this table must be accessed for all instructions in the ready list after each new instruction is scheduled. This will make the scheduling process itself slower. However, Su, Tsui, and Despain show that the combination of Gray code addressing and cold scheduling results in a 20-30% reduction in switching activity for the control path.

Another scheduling technique for reducing power consumption was presented by Tiwari, Malik, and Wolfe, [2], [3]. The goal in these works is to schedule code such that instructions are more judiciously chosen as opposed to instruction reordering. In this approach, actual current measurements were taken on general-purpose processors and DSP processors. Current was measured for each instruction and a power table built for single instruction values as well as values for common paired instructions. Then based on these measurements, test programs were rescheduled to use instructions which result in less power consumption. This selection is based on a number of issues such as register accesses as opposed to memory accesses and lower latency instructions. Tiwari, et al., also take into consideration what they term *circuit-state overhead* which is the switching activity between a pair of specific instructions. In [2] and [3], they argue that for the processors tested that circuit-state overhead is insignificant. However, a detailed analysis of another DSP processor [4], found that circuit-state overhead was much more significant in determining the energy consumption of a pair of instructions. Through this approach of physical measurement and code rescheduling, energy savings up to 40% were achieved on the benchmarks used. Again the problems with this approach are glaring. The process of hand measuring current for all instructions and instruction pairs, while extremely valuable, is extremely time consuming especially in light of the enormous instruction sets used in some of today's high-performance processors. Also, like Su, et al., there is a large table needed to store all power values. This can lead to costly accesses during the scheduling process.

The scheduling approach proposed here is focused on reducing power dissipation as opposed to power consumption. One advantage that our approach provides is that we can explicitly control the amount of power dissipation allowed for any given schedule. The two prior works simply limit power consumption as best they can. In our approach we determine how much power is dissipated which gives us tremendous flexibility in terms of being able to control dissipation for different architectures.

The remainder of this work presents our method of low-power scheduling. Section 2 presents the algorithm itself along with a discussion of the machine description mechanism. Section 3 presents the results of our preliminary investigation into this approach, and in Section 4 we conclude the paper and present plans for future work. All studies presented in this paper were performed using the experimental LEGO compiler designed by the TINKER Research Group at North Carolina State University.

2 Low-Power Scheduling

The goal of traditional scheduling algorithms is to improve performance in terms of execution time. This can be done in a number of ways. Such modern approaches as superblock scheduling [5], hyperblock scheduling [6], and treegion scheduling [7] focus mainly on increasing performance through increasing the amount of instruction-level parallelism in program code. In order to schedule for reduced power dissipation, we are forced to sacrifice some of the performance gains provided by these scheduling algorithms in order to obtain the desired reduction in power dissipation. However, the scheduling approach presented here has shown that significant energy savings can be obtained with minimal reduction in program performance.

In this section the method behind out approach will be presented. First we will discuss the machine description mechanism and how energy values are defined therein. Following will be a discussion of the scheduling algorithm itself.

2.1 MDES Machine Description

We use the MDES machine description language developed at the University of Illinois [8], [9] as the basis for defining the architecture for which we are scheduling. We have built the MDES environment into the LEGO compiler which allows us great flexibility in defining new, experimental architectures. In addition it provides a nice mechanism for defining new machine-specific parameters such as energy dissipation values. In the MDES description, we are able to define hardware resources such as registers, register files, different types of functional units, etc. In addition we can define each operation and the functional units that it

can be executed on. It is in this description that we define the energy dissipation values associated with each of the machine's functional units. Once these numbers are defined, the MDES environment builds a data structure that contains the energy information. Then, for each instruction, the scheduler queries the MDES to determine which FU the instruction is to be scheduled on. Once the scheduler knows which FU to scheduler on, it queries the MDES again to get the energy dissipation value associated with the specified FU.

For this particular study the energy values used in the MDES descriptions were obtained from actual power simulations run on different function unit types designed at Digital Equipment Corporation. The numbers provided were abstracted a bit in order not to reveal proprietary information, but are accurate enough to provide reliable results.

2.2 Scheduling Algorithm

The scheduling algorithm presented here is based on the traditional list scheduling algorithm. Once the DAG has been built for a specific region, the list scheduler builds the ready list and begins scheduling instructions based on dependence height. Currently we are only scheduling instructions at the basic-block level. Once an instruction has been cleared to be scheduled and assigned to the proper FU, the FU's energy dissipation value is queried by the list scheduler from the MDES. The list scheduler then adds the value provided to the energy total for the current cycle. If the total exceeds the threshold defined, then the scheduler quits scheduling for the current cycle and begins scheduling for the next cycle with the instruction that caused the violation in the previous one. If the list scheduler does not detect a violation, it proceeds normally.

The results presented in the following section show that this is a powerful technique for reducing power

The reduction in savings in total energy violations and total energy is inversely proportional to the amount of performance improvement achieved by increasing the energy threshold. Figures 1 and 2 clearly demonstrate this relationship. For the 12nJ threshold, Figures 1 and 2 show that the amount of violations and total energy saved falls off a bit from the 10nJ threshold. However, we feel that this reduction is acceptable given the performance gain we achieve by increasing the threshold value.

dissipation. We are currently in the process of investigating further enhancements to the current implementation.

3 Experimental Results

The studies performed so far with our scheduler have been run entirely on basic-block code on an 8-issue VLIW architecture defined as Tinker-8 which contains three integer ALU units, 2 general purpose floating-point units, two load/store units, and one branch unit. All results given in this section are for the SPECint95 suite of benchmarks. So far, we have found that careful determination of the per cycle energy threshold can result in significant savings in terms of overall energy violations and total energy saved over a given benchmark. The threshold can take on any value equal or greater to the largest energy value associated with any defined FU. At the low end of the threshold spectrum, we see the optimum amount of energy savings. However, choosing extremely low power thresholds will result in a large impact on program performance in terms of total cycle count. In contrast, selecting extremely high threshold values can result in little to no energy savings. The ideal is to find the point at which energy savings are significant while maintaining program performance. Figures 1 and 2 show results for two test cases in which we chose energy thresholds of 10 and 12 nJ per cycle. Figure 1 shows the total amount of energy violations saved by using the proposed low-power scheduling technique. In general, as the threshold decreases, the number of total violations saved increases. However, the cost is increased execution time. We found that by increasing the threshold from 10nJ to 12, that we could significantly reduce the performance impact while maintaining significant overall energy savings. The overall improvement in execution time ranged from 10.2% faster for 132.ijpeg to 1.3% faster for 130.li.

In addition to the results presented above for total energy violations and energy savings, we measured the extra energy cost in each benchmark program for every 100 cycles scheduled. For every 100 cycle segment, we calculate the total amount of excess power dissipated. Table 1 shows that excessive power is typically in the range of 10.0 to 100.0 nJ for every 100 cycles. Over large programs, this results in a large amount of excessive power dissipation which can be saved using the low-power scheduling technique proposed here.

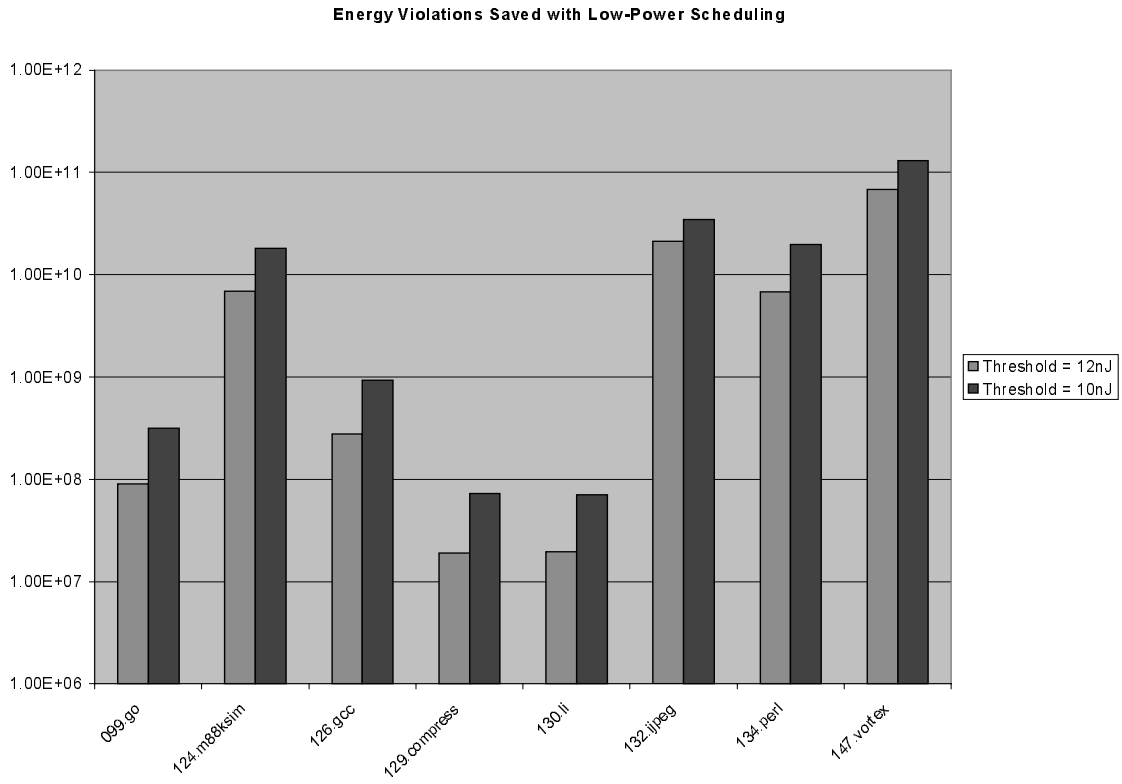


Figure 1

In Table 1, the data represents the number of times that the total power difference was in the specified range during 100 cycle spans.

4 Conclusions

In this paper, we have proposed a new method for scheduling instructions which helps reduce power dissipation in high performance processors. The approach is based on a per cycle energy threshold which may not be violated in any given cycle. Instructions are scheduled based on the list scheduling algorithm until the threshold for the current cycle is reached. Once the threshold has been exceeded, the scheduler begins scheduling for the next cycle. We have shown that this method can result in significant energy savings over a given program with little to no performance impact.

Future plans for this research are to extend the energy model to contain a more concise representation of the desired architecture and to investigate further enhancements to the scheduling algorithm to allow further increased savings.

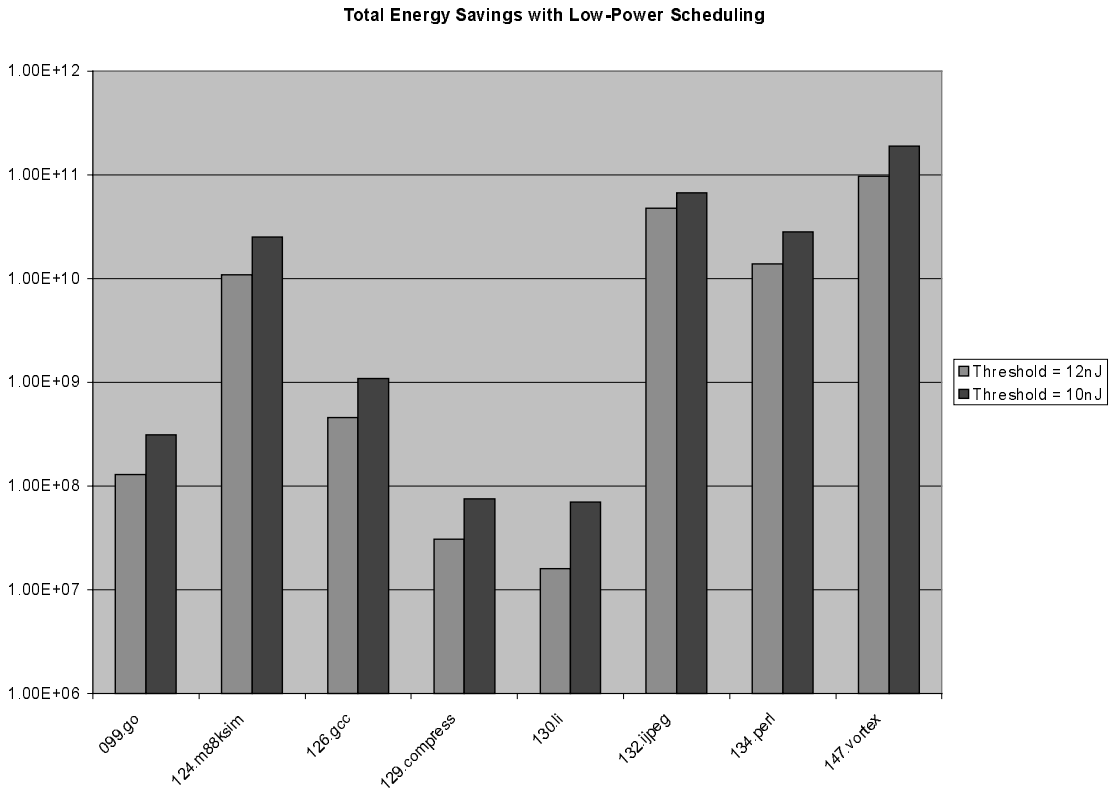


Figure 2

Table 1 – Excessive Power Distribution per 100 Cycles

	0 < X < 1.0	1.0 < X < 10.0	10.0 < X < 100.0	100.0 < X
099.go	1330	3272	2702	0
124.m88ksim	55	160	236	8
126.gcc	5752	17566	15870	255
129.compress	15	26	29	0
130.li	214	290	191	0
132.ijpeg	78	309	461	30
134.perl	754	1516	772	4
147.vortex	116	1764	6754	63

5 References

[1] Su, C-L., Tsui, C-Y., and Despain, A.M., "Low Power Architecture and Compilation Techniques for High-Performance Processors," in *Proc. of the IEEE COMPCON*, pp.489-498, 1994.

[2] Tiwari, V., Malik, S., and Wolfe, A., "Compilation Techniques for Low Energy: An Overview," presented at the *1994 Symposium on Low-Power Electronics*.

[3] Tiwari, V., Malik, S., and Wolfe, A., "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," in *IEEE Trans. on Very Large Scale Integration Systems*, pp.437-445, 1994.

[4] Lee, M.T-C., Tiwari, V., Malik, S., and Fujita, M., "Power Analysis and Low-Power Scheduling Techniques for Embedded DSP Software," presented at the *1995 Int. Symposium on System Synthesis*.

[5] Hwu, W.W., Mahlke, S.A., Chen, W.Y., Chang, P.P, Warter, N.J., Bringman, R.A., Ouelette, R.G., Hank, R.E., Kiyohara, T., Haab, G.E., Holm, J.G., and Lavery, D.M., "The Superblock: An effective structure for VLIW and superscalar compilation," in *The Journal of Supercomputing*, vol. 7, pp.229-248, Jan. 1993.

[6] Mahlke, S.A., Lin, D.C., Chen, W.Y., Hank, R.E., and Bringman, R.A., "Effective compiler support for predicated execution using Hyperblock," in *Proc. of the 25th Ann. Int'l. Symposium on Microarchitecture*, pp45-54, 1992.

[7] Havanki, W.A., *Treegion scheduling for VLIW processors*. MS Thesis, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 1997.

[8] Gyllenhaal, J.C., *A machine description language for compilation*. MS Thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1994.

[9] Gyllenhaal, J.C., Hwu, W.W., and Rau, B.R., "HMDES Version 2.0 Specification," Technical Report IMPACT-96-3, University of Illinois at Urbana-Champaign, Urbana, IL, 1996.

Code Transformations for Embedded Multimedia Applications: Impact on Power and Performance

Nikos D. Zervas
University of Patras
Dep. of Electrical & Computer Engineering, Rio 26500, Greece
Tel.: (+) 30 61 997324
E-mail:zervas@ee.upatras.gr

Kostas Masselos
University of Patras
Dep. of Electrical & Computer Engineering, Rio 26500, Greece
Tel.: (+) 30 61 997324
E-mail:zervas@ee.upatras.gr

C. E. Goutis
University of Patras
Dep. of Electrical & Computer Engineering, Rio 26500, Greece
Tel.: (+) 30 61 997324
E-mail:goutis@ee.upatras.gr

ABSTRACT

A number of code transformations for embedded multimedia applications is presented in this paper and their impact on both system power and performance is evaluated. In terms of power the transformations move the accesses from the large background memories to small buffers that can be kept foreground. This leads to reduction of the memory related power consumption that forms the dominant part of the total power budget of such systems. The transformations also affect the code size and the system's performance which is usually the overriding issue in embedded systems. The impact of the transformations to the performance is analyzed in detail. The code parameters related to the performance of the system and the way they are affected by the transformations are identified. This allows for the development of a systematic methodology for the application of code transformations that achieve an optimal balance between power and performance.

Keywords

Embedded, Multimedia, Code Transformations, Power, Performance.

1. Introduction

Image and video coding rapidly became an integral part of information exchange. The number of computer systems incorporating multimedia capabilities for displaying and manipulating image and video data is continuously increased. The rapid advances in multi-media and wireless technologies made possible the realization of sophisticated portable multi-media applications such as portable video phones, portable multimedia terminals and portable video cameras. Real time image and video processing are required in such applications. Low power consumption is of great importance in such systems to allow for extended battery life. Low power portable multimedia systems are described in [1-2]. Portability is by no means the only reason for low power consumption [3]. Low power consumption is of utmost importance in non-portable applications as well. For this reason there is great need for power optimization strategies especially in the high levels where the most significant savings can be achieved [3]. Power exploration and optimization strategies for image and video processing applications are described in [4-10].

There are two general approaches for the implementation of multimedia systems. The first is to use custom hardware dedicated processors. This solution leads to smaller area and power consumption however it lacks flexibility since only a specific algorithm can be executed by the system. The second solution is to use a number of instruction set processors. This solution requires increased area and power in comparison to the first solution however it offers increased flexibility and allows implementation of multiple algorithms by the same hardware. Mixed hardware/software architectures can also be used.

In multimedia applications, memory related power consumption forms the major part of the total power budget of a system [7-8]. A systematic methodology for the reduction of memory power consumption has been proposed in [7-8]. This methodology includes the application of loop and data flow transformations. However it mainly targets custom hardware architectures and the impact of the transformations on the performance of an implementation based on instruction set processors is not addressed.

In this paper a number of code transformations is presented. The effect of the transformations on three basic parameters of embedded multimedia systems namely power, performance and code size is illustrated. The way in which these transformations affect power and performance is analyzed. As test vehicles four well-known motion estimation algorithms are used. The aim of the research under consideration is to develop a methodology for the effective application of code transformations in order to achieve the optimal balance between power consumption and performance.

The rest of the paper is organized as follows: In section 2 a brief description of the motion estimation algorithms is given. In section 3 the applied transformations are described in detail. In section 4 the way in which the transformations affect the power consumption and the performance is described. Finally in section 5 some conclusions are offered.

2. Motion estimation algorithms

Four typical [11] motion estimation algorithms were used as test vehicles, namely full search, three level hierarchical search (hierarchical), parallel hierarchical one dimensional search (phods) and two dimensional three step logarithmic search (log). Simulations using the luminance component of QCIF frames were carried out. The dimension of the luminance component of a QCIF frame is 144x176 $(N \times M)$. The reference window was selected to include 15×15 ($(2p+1)\times(2p+1)$ and p=7) candidate matching blocks. Blocks of 16x16 $(B \times B)$ pixels were considered. The general structure of the above algorithms is described in figure 1.

```
for(x=0;x<N/B;x++)  /*for all blocks in the */
for(y=0;y<M/B;y++) /*current frame */
{
 for(i=-p;i<p+1;i++ ) /*for all candidate blocks */
  for(j=-p;j<p+1;j++) /*in the reference window*/
  {
  for(k=0;k<B;k++ )  /*for all pixels in the block */
   for(l=0;l<B;l++)
   {
    if ((B*x+i+k)<0 ||  (B*x+i+k)>N-1 || (B*y+j+l)<0 || (B*y+j+l)>M-1)
    /* Conditional statement for the pixels of the candidate blocks */
```

Figure 1: General structure of motion estimation algorithms.

The algorithms consist from 3 double nested loops. Each block of the current frame (outer loop) is compared to a number of candidate blocks included in the reference window (middle loop). Computing a distortion criterion (inner loop) using all the pixels of both the current and the candidate blocks performs the comparison. A conditional statement inside the nested loops checks whether the pixels of the candidate blocks are inside the previous frame or not.

3. Description of the applied transformations

a) Transformation 1

The first transformation applied was a loop interchange [12] between the loop related to the candidate blocks and the loop related to the pixels of each block. In this way each pixel of the current block was accessed only once (instead of (2p+1)x(2p+1) times in the original description) and its contribution to the distortions of the candidate blocks was computed. An array signal of size (2p+1)x(2p+1) was introduced to store the intermediate values of the candidate block distortions.

a) Transformation 2

The second transformation introduced a new array signal for the storage of the current block. This array signal was initialized from the current frame array signal before the loop related to the candidate blocks. This transformation is a loop distribution of the middle loop i.e. an insertion of an extra loop with the same limits and step, combined with a node splitting of the outer loop. As a consequence the inner loop accessed the current block array signal, instead of the current frame array signal, for the distortion computation.

b) Transformation 3

The third transformation introduced an array signal for the reference window. This array signal was initialized from the previous frame array signal before the loop related to the candidate blocks. This transformation is similar to transformation 2. The only difference was that the introduced array signal contained the reference window instead of the current block. So the inner loop accessed the reference window array signal, instead of the previous frame array signal, for the distortion computation.

d) Transformation 4

The fourth transformation was based in the same idea as the previous transformation but it also exploited data reuse [13]. Data reuse was feasible because of the overlapping reference windows of neighboring blocks (Figure 2). Specifically an overlapping of 2px(2p+B) pixels exists for the reference windows of two neighboring blocks. For every current block shifting its last 2p columns to the left and transferring the new non-overlapping B columns from the previous frame array signal initialized the reference window array signal.

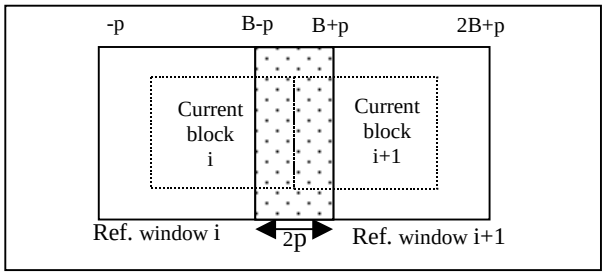


Figure 2: Overlapping of reference windows of neighboring blocks.

e) Transformation 5

The fifth transformation introduced two new array signals, one for the reference window, and one for the candidate block. The first array signal storing the reference window was initialized from the previous frame array signal before the loop related to the candidate blocks without exploiting the data reuse as in transformation 4.

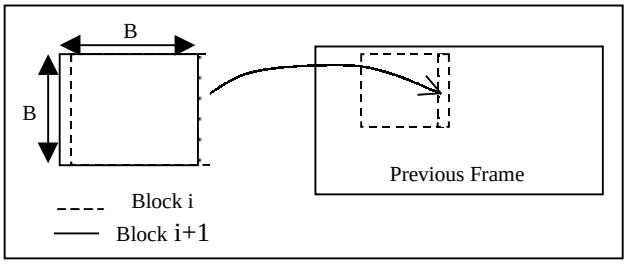


Figure 3: Overlapping of neighboring candidate blocks.

The second array signal storing the candidate block was initialized from itself for the overlapping part ((B-1)xB pixels) and from the reference window array signal for the non-overlapping part (Bx1 pixels), before the inner loop. In this way the data reuse for the previous block array signal was optimally exploited (Figures 3, 4).

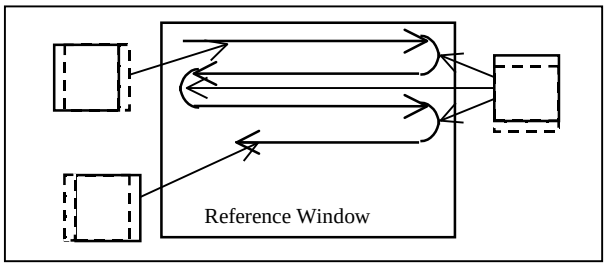


Figure 4: Optimal data reuse for the candidate blocks in a reference window.

This transformation is a node splitting of the outer loop and a loop distribution of the loop related to the candidate blocks.

f) Transformation 6

This is a data flow transformation and was applied only to the hierarchical motion estimation algorithm. The transformation produces the sub-sampled by four versions of both current and previous frames from the sub-sampled by two version of each frame, instead of producing them from the original frames as defined by the initial algorithm.

4. Effect of transformations on power and performance

As already stated memory related power consumption forms the major part of the total power budget of an image or video

processing system. Thus for the evaluation of the transformations' effect on the power consumption only the memory power consumption was considered. The transformations described in the previous section reduce the number of accesses to the large background memories storing the current and the previous frame which are the most power costly and introduce accesses to small arrays (also introduced by the transformations) that can be stored foreground requiring smaller power per access. In this way the power consumption is heavily reduced. To better illustrate the effect of the transformations on memory accesses and on power consumption let assume that the accesses required by an algorithm in its original are given by the following equation.

$$Total\ number\ of\ accesses\ (org) = A(bkg) + B(fg) \quad (1)$$

Small buffers, caches, register files and registers are considered as foreground memories. Large buffers are considered as background memories. The application of the transformations results in an algorithm description with number of memory accesses given by the following equation.

$$Total\ number\ of\ accesses\ (trnsf) = C(bkg) + D(fg) \quad (2)$$

where

$$Total\ number\ of\ accesses\ (org) < Total\ number\ of\ accesses\ (trnsf)$$

$$A >> C$$

$$B << D$$

Obviously the real effect of the transformations is the transfer of the larger part of the background memory accesses to foreground memories. The total number of accesses to memory elements is increased after the application of the transformations since extra accesses are required for transferring the data from the background memories to foreground memories. A first order metric of the transformation effectiveness (in terms of power consumption reduction) can be the relative decrease of the background memory accesses i.e. the ratio *(A-C)/A*. This first order metric of the power consumption reduction for the different transformations is presented in figure 5. In figure 6 the real power savings achieved by the application of the transformations are illustrated.

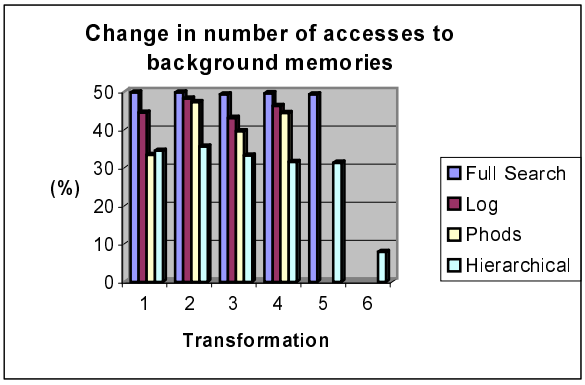


Figure 5: First order metric of power consumption reduction.

The power savings introduced by the application of the transformations described in the previous section ranged from 30% to 50% for transformations 1-5 while transformation 6 succeeded almost 8% power reduction but it affected only a small part of the complete motion estimation algorithm. For the evaluation of the power consumption of the data memory the model of Landman [14] was used. For the estimation it was assumed that the current and previous frames are stored in two separate background on-chip buffers while all the small arrays introduced by the transformations were assumed to be stored in an intermediate small buffer (data cache). All buffers were assumed to be single port read/write and 8 bits wide.

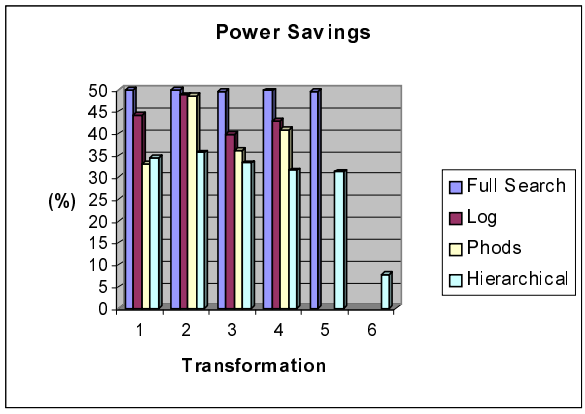


Figure 6: Power savings after the application of code transformations.

The code transformations described in the previous section affect also the system performance i.e. the number of cycles required for the execution of the code. In an abstract level, performance can be considered as a function of four code parameters namely number of memory accesses, number of computing operations, number of control operations and number of address operations as described by the following equation.

$$performanc\ e = a \times \#\ memory\ \ accesses\ + c \times \#\ control\ \ op\ /\ tions + \\ + b \times \#\ computing\ \ op\ /\ tions + d \times \#\ address\ \ op\ /\ tions \quad (3)$$

Where a, b, c, d are parameters determined by the instruction set processor on which the code is executed and reflect the number of cycles required for each one of these operations. A more refined model should distinguish the accesses with respect to the memory to which they are performed and the different computing operations with respect to their complexity. In general evaluation of the above factors before and after the application of code transformations may give an estimate of the transformation effect on performance. The effect of the transformations on the performance-number of cycles is illustrated in figure 7.

To estimate the transformation effect on performance the motion estimation codes were simulated on ARM 7 processor that was used as an embedded core. ARM is considered as the state-of-the-art core for embedded telecommunication applications. However it is still a general-purpose processor and thus it is not the most suitable for multimedia applications like motion estimation. Since only the relative effect of transformations on performance (in number of cycles) is of interest the use of ARM suffices. The effect of code transformations on the number of cycles required for program execution is not the same in all cases. This is the number of cycles either increases or decreases depending on the effect of transformations on the parameters of the model described in equation 3.

Transformation 1 increased the number of cycles for all motion estimation algorithms. The increase ranged from 1,6% (full search-simplest structure) to 60% (parallel one-dimensional hierarchical). This is due to the increase of total number of memory accesses. The number of computing and control operations remained stable and the addressing operations were reduced.

Transformation 2 decreased the accesses to background previous buffer, which has more complex address equations, while increased the number of accesses to the foreground candidate block buffer, which has simple address equations.

That way the reduction of address operations led to decrease of the number of cycles for all motion estimation algorithms of 0,2% (phods)-10%(full search) although the total number of memory accesses increased and the computing and control operations remained stable.

In the same way transformation 3 reduced the address operations. Performing check for each reference window, instead of each candidate block, lead to an 81%-98% reduction of control operations. So although the total number of memory accesses was increased, the number of cycles was reduced from 27% (full search) to 37% (log).

For transformations 4 and 5 multiple control actions for the same pixel were eliminated because of the data reuse resulting in a reduction of the number of control operations from 85% up to 99%. Transformation 4 also reduced the address operations for the same reasons as transformations 2 and 3. So although the total number of memory accesses increased, the number of cycles reduced from 27% (full search) up to 37% (log). Transformation 5 achieves optimal data reuse at the expense of complex addressing. This fact caused an 8%-63% increase of the address operations and combined with the increase of total memory accesses lead to an increase of cycles from 30% (hierarchical)-101% (full search).

It must be noted that no data caching was taken into account during simulation. This means that the performance of the codes (in number of cycles) is better than described above especially for the cases where many accesses to the faster data cache occurred. Another system parameter affected by the code transformations is the code size. Transformations usually make the code more complex and thus increase the code size. This implies an indirect effect on the system's power consumption since increase of the code size leads to an increase of the program memory. Increased size of the program memory leads to increased effective capacitance per accesses i.e. capacitance per instruction fetching. Since the motion estimation codes were mapped on ARM that is a general-purpose processor, the final code size and the number of instructions required for the program execution (effectively the number of times the program memory is accessed) are not very realistic. For this reason the power related to program memory was not evaluated. Furthermore no instruction caching was taken into account. The presence of an instruction cache reduces significantly the program memory related power consumption especially in data dominated applications where cache misses do not occur frequently. The effect of the transformations on the code size is illustrated in figure 8. Code size was increased for all the transformations and for all algorithms from 7% up to 141%, due to the extra array signals and the complexity introduced by all the transformations.

5. Conclusions

In this paper code transformations for power consumption reduction of embedded multimedia applications were presented. The transformations achieve power consumption reduction by moving the main part of the background memory accesses to small foreground memories. The code transformations affect also the system's performance. The effect of transformations on the performance is described analytically. Abstract models based on high-level code parameters for both power and performance were also described. These models can be used to evaluate the power and performance effects independently of the instruction set processor on which the code will be executed. Another system parameter affected by the code transformations is the code size. It was demonstrated that the code size increases after the application of code transformations affecting the size of the code memory and indirectly the system's power consumption. This is because increase of the size of the program memory corresponds to increase of the effective capacitance per instruction set. A model for evaluating the power consumption of the program memory was also included.

The research described in this paper is part of on-going research. The aim of our research is the derivation of models for fast and accurate evaluation of the effect of code transformations on power and performance and the development of a methodology for efficient application of transformations in order to achieve the optimal power performance balance.

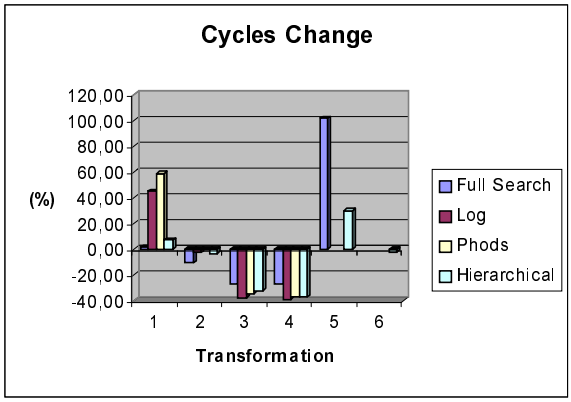


Figure 7: Transformation effect on performance (# cycles).

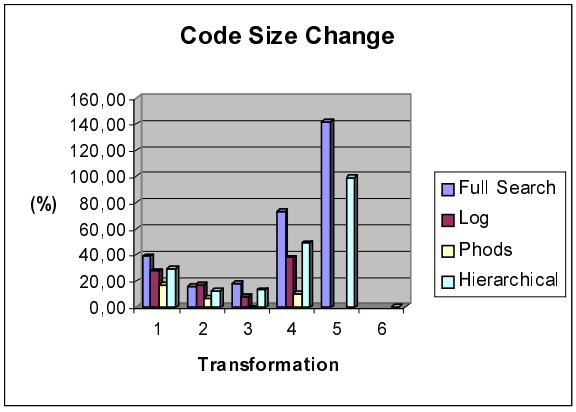


Figure 8: Effect of transformations on code size.

6. References

[1] A. P. Chandrakasan, A. Burstein, R. W. Brodersen, "A Low-Power Chipset for a Portable Multimedia I/O Terminal", IEEE Journal of Solid-State Circuits, Vol. 29, No. 12, December 1994, pp. 1415-1428.

[2] T. H. Meng, B. M. Gordon, E. K. Tsern, A. C. Hung, "Portable Video-on-Demand in Wireless Communications", Proceedings of the IEEE, Vol. 83, No. 4, April 1995.

[3] J. M. Rabaey, M. Pedram, "Low Power Design Methodologies", Kluwer Academic Publishers 1995.

[4] D. B. Lidsky, J. M. Rabaey, "Low-Power Design of Memory Intensive Functions", 1994 IEEE Symposium on Low Power Electronics, Digest of Technical Papers, pp. 16-17.

[5] M. Tartagni, A. Leone, A. Pirani, R. Guerrieri, "A Block-Matching Module for Video Compression", 1994 IEEE Symposium on Low Power Electronics, Digest of Technical Papers, pp. 24-25.

[6] K. K. Chan, C. Y. Tsui, “Exploring the Power Consumption of Different Motion Estimation Architectures for Video Compression”, proc. of the 1997 IEEE Intl. Symposium on Circuits and Systems, pp. 1217-1220.

[7] L. Nachtergaele, F. Catthoor, F. Balasa, F. Franssen, E. De Greef, H. Samsom, H. De Man, “Optimization of memory organization and hierarchy for decreased size and power in video and image processing systems”, IEEE Intl. Workshop on Memory Technology, Design and Testing, August 7-8 1995, San Jose, pp. 82-87.

[8] S. Wuytack, F. Catthoor, L. Nachtergaele, H. De Man, “Power Exploration for Data Dominated Video Applications”, proc. of the 1996 Intl. Symposium on Low Power Electronics and Design, Monterey, California, pp.359-364.

[9] F. Catthoor, M. Janssen, L. Nachtergaele, H. De Man, “System-Level Data-Flow Transormations for Power Reduction in Image and Video Processing”, proc. of the 1996 Intl. Conference on Electronics Circuits and Systems (ICECS’96), Rhodos, Greece, pp. 1025-1028.

[10] L. Nachtergaele, F. Catthoor, B. Kapoor, S. Janssens, “Low power storage exploration for H.263 video decoder”, in VLSI Signal Processing IX, IEEE Press, pp.116-125, 1996.

[11] V. Bhaskaran, K. Konstantinides, “Image and Video Compression Standards”, Kluwer Academic Publishers, 1994.

[12] D. Kulkarni, M. Stumm, “Loop and Data Transformations: A tutorial”, Technical Report CSRI-337, Computer Systems Research Institute, University of Toronto, June 1993.

[13] J. P. Diguet, S. Wuytack, F. Catthoor, H. DeMan, “Hierarchy Explorartion in High Level Memory Management”, in proc. of the 1997 International Symposium on Low Power Electronics and Design, Monterey CA, August 18-20.

[14] P. Landman, “Low power architectural design methodologies”, Doctoral Dissertation, U. C. Berkeley, Aug. 1994.

Modeling Inter-Instruction Energy Effects in a Digital Signal Processor

Ben Klass, Donald E. Thomas, Herman Schmit, David F. Nagle
Department of ECE, Carnegie Mellon University
Pittsburgh, PA 15213
benk@ece.cmu.edu, thomas@ece.cmu.edu

Abstract

This paper explores techniques for creating accurate instruction-level energy models for digital signal processors (DSP). Our initial results confirm previous work showing that inter-instruction effects can become a significant component of power consumption for many programs. To overcome limitations of previous models, we develop a straightfoward method (the NOP model) that models transitions between any two instructions. Measurements show that our method accurately models inter-instruction effects without a quadratic increase in the size of energy tables. Complex instructions are handled by treating functional units within the processor separately.

1 Introduction

Instruction-level energy models can be an effective tool for high-level software-based optimizations [LEE97][TIWA94]. The basic technique constructs a table that records each instruction's average energy. High-level power estimators use this table to quickly determine each software instruction's energy consumption, avoiding costly circuit-level simulation (e.g., Spice). Because instructions are the atomic units used by code generators, instruction-level energy models can be integrated with power-optimizing compilers more easily than simulation-based estimators. Further, instruction-level energy models allow chip manufacturers to provide fine-grained power information without having to disclose confidential design layout and implementation details—allowing software designers to quickly and accurately estimate a program's power consumption without understanding the underlying implementation details.

Unfortunately, accurate instruction-level energy models require more than simple per-instruction power estimates. Inter-instruction effects can significantly alter the power consumed by a given instruction, making it difficult to derive a single power number for each architectural instruction [LEE97]. Power tables could be expanded to include every pair of instructions. Unfortunately, building such tables can be very time consuming and requires $O(N^2)$ space (where N is at least the size of the instruction set). Grouping instructions into common classes [Lee97] can reduce the table size, but does not scale well for DSP-type architectures with their rich addressing modes and parallel instruction issue capabilities.

To overcome the problems of classification, we have developed a straightfoward method that requires only O(N) space while accurately estimating program energy. Our results, simulated with an implementation of a subset of the Motorola DSP56000 (56K), produce instruction-level power tables that predict program power within 8% percent of simulation-based estimates. Further, by attributing each instruction's power consumption to the various functional units, we preserve accuracy while overcoming the difficulty associated with modeling the 56K's rich addressing modes and parallel functions.

Section 2 describes our subset of the 56K DSP and our design methodology. Section 3 presents our approach to generating instruction-level power tables and compares our results with previous techniques. Section 4 further evaluates these models and describes potential limitations. Finally, in Section 5 we present our conclusions and outline future work.

2 Tools and Methodology

2.1 CMU 56000 DSP

To build accurate models and to compare our results with a real design, we designed and implemented a standard-cell based subset of the Motorola DSP56000 instruction set [MOTO90]. Synopsys and Cascade's Epoch synthesized our 56K Verilog model into a standard-cell layout (see Figure 1).

We choose the 56K because instruction-level power analysis is more complex than simple RISC cores and because the 56K's functionality is representative of many power-conscious architectures. The 56K is a 24-bit, fixed-point DSP that can encode and issue one arithmetic operation and up to two "parallel" data moves in one *packed* instruction. Our 56K core implements most of the arithmetic and basic data movement instructions and accounts for most of the logic that effects power consumption.

2.2 Mynoch Power Estimator

A variety of approaches have been used to characterize the power consumption of digital systems. For physical devices, direct-measurement of current gives the most accurate measurements [TIWA94]. However, the granularity of results is limited to device-level, multi-clock cycle measurements. Accurate, fine-grained results can be obtained with Spice-based simulators, such as Star-Sim [KRIS97], but long run-times severely limit the number of cycles/events that can be simulated. Gate-level power estimators improve simulation speed [KOJI95][PURS95][XANT97], by sacrificing accuracy to achieve faster run-times.

The initial analysis of our 56K's power consumption was done using CMU's gate level analysis tool, Mynoch [PURS95]. Mynoch estimates power by counting transitions from a Verilog

This work was supported by the Defense Advanced Research Projects Agency under Order No. A564 and the National Science Foundation under Grant No. MIP90408457.

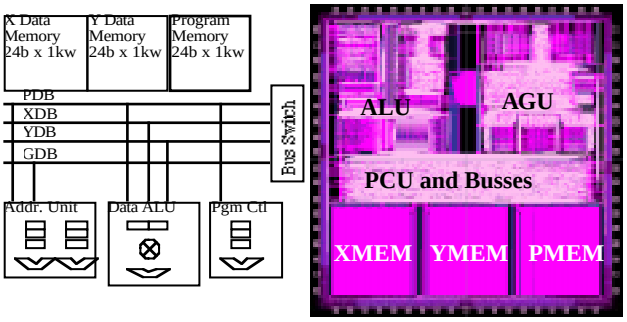


Figure 1: DSP56000 architecture and layout

Major components: 1) 3 KB of SRAM; 2) data ALU, which contains a 24x24-bit multiplier and 56-bit accumulator; 3) address generation unit (AGU), which contains three sets of eight 16-bit registers and two ALUs capable of arbitrary modulo and bit reversed arithmetic; and 4) program control unit (PCU).

simulation and calculating the dynamic energy consumed for each transition using:

$$E = \frac{1}{2} \bullet C \bullet V^2$$

Mynoch runs 450 times faster than Spice simulation, allowing us to simulate thousands of cycles for each test program. Memory is not modeled with Mynoch since Epoch uses behavioral models for memory. Spice based simulations have shown memory to be approximately 20% of the total energy.

We verified the accuracy of Mynoch by comparing Mynoch's power estimates against Avanti's Star-Sim, which uses a modified Spice engine. Eighteen iterations of a 4-tap FIR filter were simulated with both Mynoch and Star-Sim. Initially, Mynoch's power estimates showed significant error in contrast to Star-Sim. To locate the source of Mynoch's error, we compared Star-Sim's per module power estimates with Mynoch's estimates. The analysis showed that Mynoch's inability to account for intra-gate capacitance within registers (i.e., D-flip flops) was the primary source of error. To correct for this error, we used Star-Sim's power estimates to build a simple linear-regression model that included the number of registers. The resulting model had a very high degree of accuracy (r^2 factor of 0.98). This model was further verified by comparing results of Mynoch augmented by the regression model vs. Star-Sim for 120 cycles of an FFT program. The error between the two methods was less than 1% for the processor, although error on functional units was higher. All of the Mynoch results reported in this study are augmented by the regression model.

Pgm	Description	Instr	%Instr arith	Sim cycles
fir4	4-tap FIR filter (direct form)	5	57%	1760
fir64	64-tap FIR filter (direct form)	5	96%	5,000
fir4u	4 tap FIR filter, unrolled once	12	66%	1,500
FFT	256 point FFT	24	79%	8,062
LMS	64-tap least mean squares adaptive filter	13	63%	30,000

Table 1: Description of workloads

The **Instr** column lists the number of unique instructions executed in each workload's main program loop. **%Instr arith** lists the percentage of instructions executed that are arithmetic instructions. **Sim cycles** lists the number of cycles simulated for power analysis.

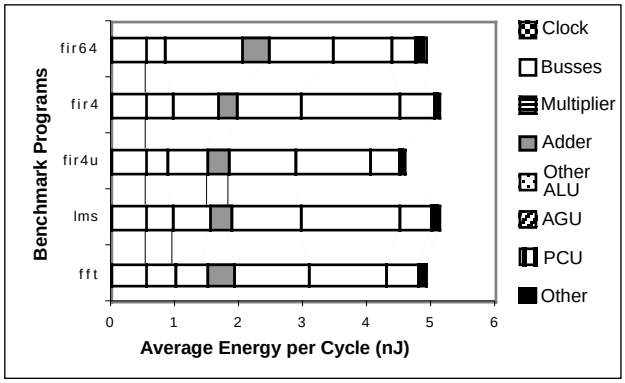


Figure 2: Energy consumption for workloads

This figure shows power consumption for the workloads. The AGU and ALU are the two largest consumers of power and almost all of the variation between programs is caused by variation in the multiplier and AGU power. The multiplier uses guard latches on the input and consumes less power with programs that have proportionally fewer multiply operations.

2.3 Test Programs and Reference Energy

In contrast to general processors, a DSP is frequently used to compute fairly simple, data intensive programs. For the workloads of our power analysis we chose five program kernels that represent those found in typical signal processing applications (see Table 1). Three of the kernels are finite-impulse response (FIR) filters, one is a Fast Fourier Transform (FFT), and one is an adaptive filter (LMS). Gaussian white noise was used as input data.

Power consumption for the benchmark programs was measured using Mynoch (see Figure 2). Average energy consumed per cycle, given in nanoJoules, provides the basic unit of measure which will be used throughout the paper. This is obtained by dividing the total energy over the program execution by the number of cycles. Power consumed by the pads and memory is not included.

3 Instruction-level Models

Although circuit-level simulations provide insight into power consumption for a given program, an instruction-level model is more appropriate for code generators. Instruction-level models typically use an ***energy table*** that describes the energy cost for each instruction in an instruction-set architecture (ISA). The challenge in building such a model is balancing accuracy and energy table size. This section presents four models with different accuracy and table size trade-offs. The first model, ***base model***, produces the smallest table size, but yields poor energy-prediction accuracy across a program run. The second model, ***pair model***, has greater accuracy, but at the cost of much larger tables. The third model, ***NOP model***, provides nearly the accuracy of the pair model with much smaller tables. The fourth model, ***general model***, is similar to the NOP model but the energy table generated is independent of the program being evaluated. The general model provides reasonable accuracy and is appropriate for use with code generators.

3.1 Base Model and Estimation

Building the Base Model's Energy Table

Creating a complete and accurate energy table for the 56K DSP requires one to account for each instruction in the ISA, each instruction's different register and immediate values, and every possible packed instruction.[1] This can require a significant amount of time and space. To make the base model more

```
DO #<50
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0   ;target instruction
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0   ;target instruction
```

Figure 3: Loop used to characterize MAC from FIR filters

This loop was used to calculate the base energy cost of the target instruction, in this case the MAC instruction as it occurs in the FIR filters. Two data values are read in from memory each cycle, so both multiplier inputs change. Two instances of the target instruction are needed to match the semantics of the DO instruction.

tractable, we only computed the energy cost for every unique instruction[2] found in our five workloads—not for every possible tuple of {`instruction, reg/immed, instr pair`}.

Each instruction's energy was estimated by constructing a tight-loop test program that included the target instruction and a zero-overhead branch instruction so that only the target instruction was executed in the core of the loop. Figure 3 shows the tight-loop test program used to characterize the `MAC` instruction. Each tight-loop test program was run through Mynoch to generate the base energy cost, B_{inst}. For measuring the `REP` instruction, we were forced to use a `NOP` inside of the loop because a loop of `REP` instructions is illegal.

Whenever possible, loops were made with the actual instructions used in the programs. Some instructions were modified slightly to ensure that different operands were used on each cycle. Data from the workload being characterized was used. Because the test programs are based on the actual instructions and data from our five workloads (Table 1), the results of this approach are optimistic in their accuracy. When generalizing this approach, parameters such as the destinations of parallel moves would be abstracted to reduce energy table's size.

```
CLR A X0,X:(r0)+ Y:(r4)+,Y0     ; A = 0;
                                 ; X0 <- X(n-1); Y0 <- B(0)
REP #<3                          ; for j = 0 to 2
MAC Y0,X0,A X:(r0)+,X0 Y:(r4)+,Y2  ; A+= X(n-j)*B(j);
                                    X0 <- X(n-j-1); Y0<- B(j+1)
MACR Y0,X0,A (r0)-               ; A += X(n-3)*B(3);
                                    update pointer
MOVE X:(r1)+,X0 A,Y:(r5)+        ; X0 <- X(n+1); Y:(out) <- A
```

Figure 4: Main loop of `fir4`

This figure shows the code used in the fir4 main loop. The `fir64` is the same, except that the MAC operation is repeated 63 times by changing the repeat instruction to "`rep #<63`." The CLR, MAC, MACR, and MOVE instructions employ parallel moves to move data into registers immediately before the data is needed.

1. Like many DSPs, the 56K allows for packed instructions, where an arithmetic instruction and a data-movement instruction are grouped together in one instruction.
2. Instructions are considered different if there is any difference in their opcode, immediate values, registers, or pairings. For example, two versions of a MAC instruction, (`MAC x0,y0,a` vs. `MAC x1,y0,b`), are considered different and will have different entries in the base model's energy table.

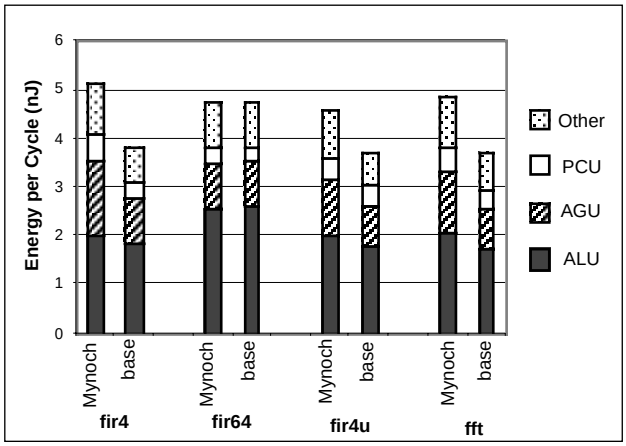


Figure 5: Mynoch vs. Base Model

Simulated energy from Mynoch and estimated energy using the base model (`base`) are given for major units. Both clock and bus power are contained in "Other." (Due to length and complexity, LMS has not been done.)

Base Model Energy Estimates

The instruction measurements described above were used to construct a base model energy table that was then used to estimate the average energy per cycle for each of our workload programs. For example, the energy per cycle for the `fir4` program was calculated by:

$$E_{fir4} = (B_{CLR} + B_{REP} + 3\,B_{MAC} + B_{MACR} + B_{MOVE}) / 7$$

Figure 5 shows the power estimates gained from the instruction energy table. The results show a very accurate power estimate for the `fir64`. However, the estimated energy for other programs is underestimated by 17% to 25%. This error is 50% higher than the error reported in [LEE97].

This error can be understood by considering the `fir64` and `fir4` programs (see Figure 4). The accuracy in the `fir64` workload is due to `fir64`'s limited number of inter-instruction effects. The `fir64` repeats the `MAC` instruction 63 times, with no intervening instructions, in a loop of 67 instructions. This behavior is very similar to the tight-loop test programs used to derive the instruction energy table. In the `fir4`, however, energy is underestimated by 25% because inter-instruction effects are significant. The `fir4` repeats the `MAC` instruction 3 times in a loop of 7 instructions, while the remaining 4 instructions are all different. Inter-instruction effects in the remaining instructions are not represented by the base model, leading to the observed error.

For each of the workloads, most of the inaccuracy for the base model occurs in the DSP's AGU (address-generation unit) and PCU (program-control unit) functional units. For the fir4, the energy for the AGU and PCU are underestimated by more than 30%. The error in these units can be understood by considering the microarchitecture (Figure 1). The AGU must generate an address by the end of pipeline stage 2, so no registers exist between the control logic and the data path (our implementation does this to increase performance). The PCU does not latch its control points for similar reasons. The lack of registers allows glitches in the control logic to propagate into the data path, causing many false transitions as an instruction word changes. In contrast, the ALU (data ALU functional unit) latches all of its control points, so glitches are confined to the control logic, making the base model more accurate.

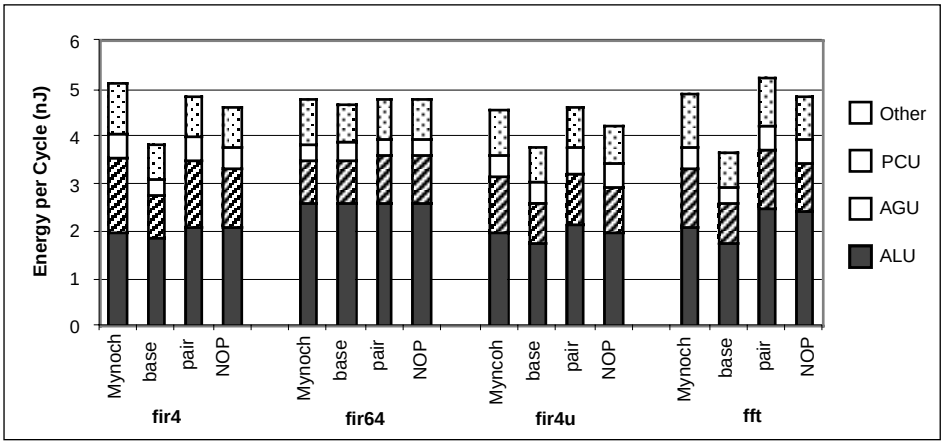


Figure 7: Different approaches to estimating inter-instruction overhead

This figure compares energy from Mynoch simulation with estimates from the base model (`base`), pair model (`pair`) and NOP model (`NOP`). The pair model measures overhead for each pair of instructions that appear in the program trace. The NOP model measures overhead for each instruction using NOP instructions

3.2 The Pair Model

The impact of inter-instruction effects on power estimation has been noted before and can be compensated for by assigning a per-instruction overhead that accounts for inter-instruction effects [LEE97]. This overhead, O_{instr}, is added to the base energy cost if an instruction is not the same as the previous instruction. For example, the energy model for the code sequence:

MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*; target instruction*

would only use B_{MAC} for the second MAC operation, while the energy model for the code sequence:

MOVE X:(r1)+,X0 Y:(r4)+,Y0	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*; target instruction*

would use $B_{MAC} + O_{MOVE,MAC}$ because the `MAC` instruction is preceded by a different instruction, `MOVE`.

Similar to the base model, we measured O_{instr}, using tight-loop test programs. Each loop consisted of the target instruction and the instruction that preceded it in the execution trace, giving average energy per cycle for the loop E_{loop}(see Figure 6). Overhead was calculated by:

$$O_{previous,target} = E_{loop} - (B_{previous} + B_{target})/2$$

DO #<50	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	
MACR Y1,X1,a (r0)-	*;target instruction*

Figure 6: Loop used to find overhead: pair model

This loop was used to calculate the overhead cost of the target instruction, in this case the MACR instruction as it occurs in the FIR filters, under the pair model. The pair of instructions that appear in the workload programs was used and the overhead for this pair was assigned to the trailing instruction. Different source registers were used for the MAC and MACR instructions to ensure that both multiplier operands change, as in the FIR programs.

Using B_{instr} and O_{instr} where appropriate, the pair model estimated the energy for each of the workloads (Figure 7). The results, labeled as `pair`, are much more accurate than the base model, with error between 1% and 10% for all programs. However, generalizing this technique would require characterizing every possible pair of instructions, requiring a table of size $O(N^2)$, where N is the number of instructions and addressing modes. For the 49 different instructions and addressing modes implemented in our 56K chip, a complete instruction energy table would contain 1176 entries. To reduce the table size, [LEE97] grouped instructions into classes and derived overhead costs between classes. This technique works well for simple machines, but is much more difficult to apply when dealing with the many complex addressing modes and instruction types found in a DSP such as the 56K.

3.3 The NOP Model

To avoid the difficulties of instruction grouping, we have developed a new approach that requires only one overhead cost for each instruction. This model is based on the assumption that the overhead cost for an instruction is not strongly dependent on the neighboring instruction, but does depend on whether the neighboring instructions are the same or different. This observation leads to the NOP model, which allows us to account for instruction changes without enumerating each pair of instructions.

Like the pair model, the NOP model calculates the energy for a particular operation with either B_{instr} or $B_{instr} + O_{instr}$, depending on the previous instruction. The NOP model differs in that we calculated one overhead cost for each instruction, O_{instr}, using loops which alternate the target instruction with `NOP` instructions (Figure 8). This techniques allowed us to capture the energy effects of changing instructions while keeping the size of the table to O(N). Power estimates using the NOP model are shown in Figure 7, labeled as `NOP`. The results show

DO #<50	
NOP	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*;target instruction*

Figure 8: Loop used to find overhead: NOP model

This loop was used to calculate the overhead cost of the target instruction, in this case the MAC instruction as it occurs in the FIR filters, under the NOP model. A target instruction is paired with an NOP to calculate its overhead cost.

```
DO #<50
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0   ; MACxy
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0   ; MACxy
DO #<50
MOVE X:(r0)+,X0 Y:(r4)+,Y0          ; MOVEx+y+
MOVE X:(r0)+,X0 Y:(r4)+,Y0          ; MOVEx+y+
```

Figure 9: Loops used for general model

These loops were used to calculate the base cost of the MACxy and MOVEx+y+ instructions under the general model. The MACxy refers to a MAC instruction where both multiplier inputs change while MOVEx+y+ refers to two parallel moves with increment.

error between 1% and 8% on programs that previously had much larger errors with the base model. Considering that grouping instructions also decreases accuracy, this should compare favorably with any model based on instruction classes.

3.4 General Instruction Model

Having established the effectiveness of the NOP model, we generalized this approach to build tables that could be used for any program—a general instruction model. Unlike our previous models, where the instruction energy tables were built to match instructions as found in the workloads as closely as possible, the general instruction model creates a single instruction energy table that can be used across all programs. Such a table could be created by processor manufacturers and then used by a code generator to optimize power.

The general instruction model is similar to the NOP model, but extends the power analysis by accounting for packed instructions. Packed instructions present a problem when building general tables because any combination of arithmetic and parallel move is allowed. Our previous models used the actual packed instructions from each workload. When generalizing, the 23 arithmetic instructions and 24 types of parallel moves lead to 552 possible combinations, making a complete table fairly large. Fortunately, the two parts of a packed instruction are largely executed by different units within the 56K. Using this architectural knowledge, we separated the two parts of a packed instruction, building tables for the energy consumed by each of the functional units rather than the entire DSP.

The general instruction model consists of four tables, corresponding to the four significant functional units: ALU, AGU, PCU, and "Other." The first three units have been described above. "Other" refers to all remaining parts of the chip, primarily the clock and bus power. The ALU and PCU were characterized by the arithmetic portion of packed instructions only, ignoring the parallel move unless no arithmetic instruction was

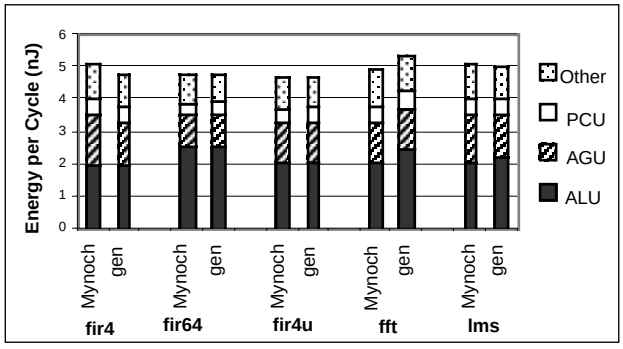


Figure 10: General Instruction Model

This figure compares Mynoch simulation energy (`Mynoch`) and the general instruction-level model (`gen`). Each component: AGU, ALU, PCU, and Other was estimated using separate tables.

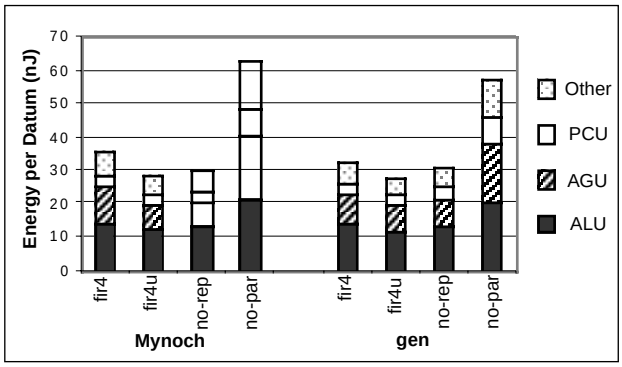


Figure 11: Implementations of a 4-tap FIR filter

This figure compares the energy to produce one datum of output for four different software implementations of the same 4-tap FIR filter. The `fir4` and `fir4u` implementations are described in Table 1. The `no-rep` implementation does not use the repeat instruction or store past inputs. The `no-par` implementation only uses one parallel move and does not use packed instructions. The number of cycles per datum for the `fir4`, `fir4u`, `no-rep`, and `no-par` programs are 7, 6, 6, and 15, respectively.

present. The AGU and "Other" were characterized by the parallel move portion, ignoring arithmetic instructions. Data was coarsely modeled in ways that would be visible to a code generator. Table entries for arithmetic operations were separated based on which operands changed value; move operations contained separate entries for the number of moves and type of update performed on address registers.

Energy costs were generated with loops similar to those described above (Figure 9). From each loop, the relevant energy costs were calculated for each unit. Uniform random data was used as input to the arithmetic unit. Under the general instruction model, the base energy cost of the instruction:

```
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0
```

was calculated by:

$$B_{ALU,MACxy} + B_{AGU,MOVEx+y+} + B_{PCU,MACxy} + B_{Other,MOVEx+y+}$$

Overhead energy costs, and whether an instruction has changed, was calculated for each unit in the same way.

Estimates based on these general instruction tables are shown in Figure 10. By making estimation automatic, we were able provide estimates for `lms` as well. Considering that program dependent information from Figure 7 has been removed, results are remarkably similar. Accuracy on all programs is within 10%.

4 Applications and Limitations

Section 3 developed an general instruction model that provides reasonable accuracy while limiting the table size. In this section we analyze this model from two perspectives. The first examines a possible use of such an energy model—evaluating the energy of code transformations within a code generator. The second looks into the importance of program data, which is not considered by the instruction-level model.

4.1 Code Transformations to Save Power

Comparing different implementations of a 4-tap FIR filter allows us to see if the energy models can recognize power savings due to code transformations (see Figure 11). While [TIWA94] looked at instruction reordering, more aggressive code transformations are used here. We implemented four versions of a 4-tap FIR filter which used the same coefficients and

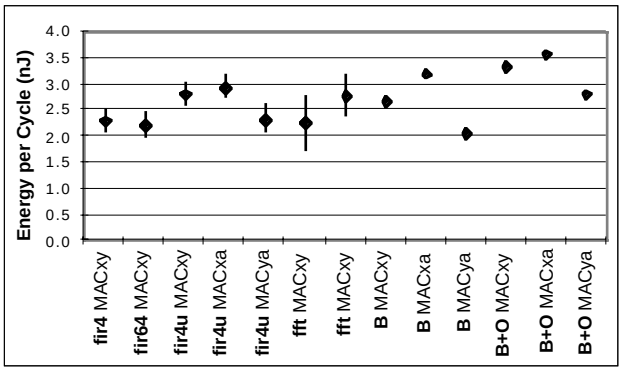


Figure 12: Energy per instruction of data ALU

The average energy for each instruction, based on Mynoch simulation of the programs, is given with standard deviation error bars to the left. The three instructions with the highest standard deviation showed a bimodal distribution. Base energy cost (B) and base plus overhead (B+O) from the general model are given to the right.

input data. Energy per datum processed is used as the metric to compare these programs to account for the different number of cycles required by different programs. Energy per datum is given by per-cycle energy multiplied by the number of cycles per datum.

Figure 11 shows that Mynoch power estimation predicts that loop unrolling, `fir4u`, consumes 20% less energy per datum than `fir4` while the version without packed instructions, `no-par`, consumes 75% more energy per datum. The difference in energy is due to both the energy per cycle and the number of cycles required to process one datum. The general model is able to recognize the difference in power, but does not show as dramatic an improvement for `fir4u` and `no-rep`. The lost accuracy comes from the general model underestimating the AGU's per-cycle energy for `fir4` while overestimating the AGU's per-cycle energy for `fir4u` and `no-rep`. While improved accuracy in AGU power estimation is needed, results show that a code generator using this model would choose the implementation using the least power under the general model we developed.

4.2 Data Dependent Variation

None of the models presented consider energy effects of program data. However, the power consumption of many units, such as the multiplier, can be highly data-dependent. Other research on DSP power consumption has noted the data dependent variation and analyzed the energy of individual functional units, such as the multiplier [KOJI95][LEE97]. Code transformations that keep one operand constant or reduce the number of "1" bits in a Booth-Encoded multiplier are ways that the compiler can change the data to reduce the multiplier's energy.

To gauge the importance of data, we used cycle accurate simulation to measure the power in the ALU when each MAC operation was active for the workloads. The average power consumed by each instruction, with standard deviation error bars, is shown in Figure 12 along with the energy costs for these instructions from the general model. Most instructions deviate between 8% and 12% from the mean, while two instructions deviate by 16% and 28%. The costs from the base model are generally within the one standard deviation of the workloads.

The `fft` has the largest standard deviation and has the least accurate ALU estimates under all models (c.f. Figure 7). The `fft` showed a bimodal distribution of energy in one of its MAC instructions, probably due to a large number of multiplications by zero. Improving accuracy for this problem would require moving to a model based on execution traces with sample program data. Increased accuracy of such a model would come at a cost of significantly longer simulation time, although techniques such as those proposed in [MARC96] could be used to reduce the trace length.

5 Conclusion and Future Work

We have presented several approaches for dealing with inter-instruction effects when building instruction-level energy models for a specific DSP design. The results show that using `NOP` instructions to model transitions between any two instructions give accuracy within 8% while reducing table size from almost 1200 to less than 100, and eliminates possible human error from other simplification methods. Using separate models on major units within the DSP avoided multiple table entries for different combinations of arithmetic and parallel move instructions and allowed us to build a general model. Such a model could allow code generators to recognize code transformations that reduce the energy consumed by programs.

Future work attempt to recognize when data dependent variation is likely to be important and include such variation within the model. Building models for other, non-DSP architectures would further validate the applicability of the ideas presented here.

6 Acknowledgments

Andrew Ryan, Chris Inacio, Jonathan Ying-Fai Tong and Bill Dougherty Jr. provided critical components for this study.

7 References

[**BAJW97]** R. S. Bajwa, N. Schumann, H. Kojima. Power Analysis of a 32-bit RISC Microcontroller Integrated with a 16-bit DSP. *Proceedings 1997 International Symposium on Low Power Electronics and Design*, pp. 137-142, 1997.

[KOJI95] H. Kojima, D. Gorny, K. Nitta, and K. Sasaki. Power Analysis of a Programmable DSP for Architecture/Program Optimization. *IEEE Symposium on Low Power Electronics, Digest of Tech. Papers*, pp. 26-27, Oct. 1995.

[KRIS97] Ram K. Krishnamurthy, *Mixed Swing Techniques for Low Energy/Operation Datapath Circuits*, Ph.D. Thesis, Carnegie Mellon University, December 1997.

[LEE97] M. T.-C. Lee, V. Tiwari, S. Malik, M. Fujita. Power Analysis and Minimization Techniques for Embedded DSP Software. *IEEE Trans. on VLSI Systems*, pp. 1-14, March, 1997.

[MARC96] Diana Marculescu, Radu Marculescu, Massoud Pedram. Stochastic Sequential Machine Synthesis Targeting Constrained Sequence Generation. *Proceedings of Design Automation Conference*, pp. 696-701, 1996.

[MOTO90] Motorola, Inc. *DSP56000/56001 Digital Signal Processor User's Manual*. 1990.

[PURS96] D. J. Pursley. A gate level simulator for power consumption analysis. M.S. Thesis, Carnegie Mellon University, May 1996.

[TIWA94] V. Tiwari, S. Malik, A. Wolfe. Power Analysis of Embedded Software: A First Step towards Software Power Minimization. *I994 ICCAD, Digest of Technical Papers*. pp. 384-390, 1994

[XANT97] T. Xanthopoulos, Y. Yaoi, A. Chandrakasan. Architectural Exploration Using Verilog-Based Power Estimation: A Case Study of the IDCT. *DAC 97*, pp. 415-420, 1997.

Power Issues in the Memory Subsystem

Split Register File Architectures for Inherently Lower Power Microprocessors *

V. Zyuban and P. Kogge

Computer Science & Eng. Department, University of Notre Dame, IN 46556, USA

Abstract

Register files represent a substantial portion of the energy budget in modern processors, and are growing rapidly with the trend towards wider instruction issue. The actual access energy costs depend greatly on the register file circuitry and technology used. However, according to a recent study, it appears that neither technology scaling nor circuitry techniques will prevent centralized register files from becoming the dominant power component of next-generation superscalar processors. This paper studies alternative methods for inter-instruction communication as to their energy efficiency and begins to lay out approaches at the architectural level that would allow inherently more energy-efficient computations.

1 Introduction

Current microprocessor design has a tendency towards wider instruction issue and increasingly complex out-of-order execution. This leads to the growth of the on-chip hardware, and dissipated power. Energy-delay product, $\frac{delay}{operation} \times \frac{energy}{operation}$, or its inverse $\frac{SPEC^2}{W}$, seem to be a reasonable metric for power efficiency of a design [8]. Smaller energy-delay values imply a lower energy solution at the same level of performance – a more energy-efficient design.

Reference [10] described and analyzed those portions of a microarchitecture where complexity, and consequently power grow with increasing instruction issue width. Among them are: register rename logic, wakeup and selection logic, data bypass logic, register files, caches and instruction fetch logic. The power growth of each of these structures can be described as $Power \sim (IPC)^{\gamma}$, where IPC is the average number of instructions completed per cycle, and γ is some constant. Every one of the above structures has its own power growth constant γ. By substituting the assumed expression for power dependence upon IPC to the energy-delay formula we see that those structures that have $\gamma > 2$ may begin to swamp the power budget as the processor issue width and IPC grow, and eventually lead to a deterioration of the energy-delay metric and thus the energy efficiency of a microprocessor.

In this paper, which is a part of a bigger project, we concentrate on one of the structures whose γ is greater than 2, namely the centralized multiported Register File (RF), and consider alternative methods for inter-instruction communication that are more energy efficient. Our goal is to find one that has $\gamma \leq 2$.

*This work was supported in part by the National Science Foundation under Grant No.MIP–95–03682.

2 Power Complexity of Centralized Register File

Register files in modern superscalar CPUs are usually centered on multiported memory macros whose storage size and the number of ports grow with increasing issue width. The silicon area of a multiported memory, built using conventional approaches, grows quadratically in the number of ports [13]. Therefore, taking into account growth both in storage needs and the number of ports, we should expect that the power portion of such multiported on-chip memories will grow rapidly in the future.

In our recent work [17] we studied the dependence of the access energy to a multiported register file upon the number of read and write ports, and the number of registers in the register file. We did a study for various circuit organizations of the RF, and tried to find the lower bound on the access energy. The study showed that even when the Port Priority Selection (PPS) technique [11] is applied, combined with double-ended reads and low-swing writes (which was found to be the most energy efficient), still the access energy grows significantly as the number of ports and the number of registers increases, Fig. 1. Here we assumed 0.95 RF read accesses and 0.6 RF write accesses per instruction (measured for the SPARC-V8), and for the number of ports $N_{read} = 2N_{write}$.

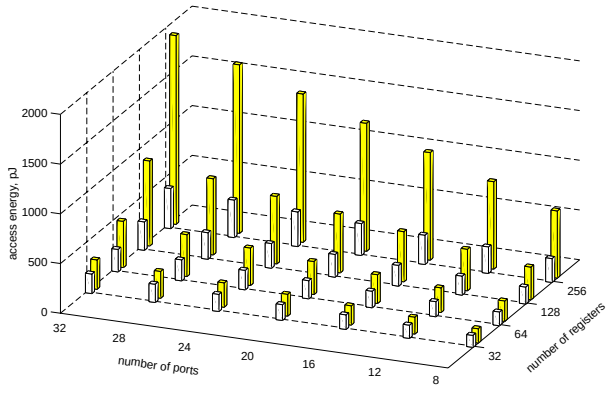


Figure 1: Average access energy per instruction of a RF using the PPS technique combined with double-ended reads and low-swing writes (1st bar) and a conventional RF architecture (2nd bar), 0.5um technology, $V_{dd} = 3.3V$

To express the RF access energy and dissipation power in terms of the issue width (IW) of a processor, we assumed for the number of read and write ports: $N_{read} = 2IW$, and $N_{write} = IW$. To estimate the number of registers we used the results in [4], where it was found that for a four-issue and eight-issue machines the performance saturates around 80 and 128 registers, respectively. Based on this data, and assuming that 40 registers is sufficient for a single issue machine, we extrapolate the dependence linearly to two-issue and 16-issue machines.

The average RF access energy per instruction $E_{average}$ versus the microprocessor issue width is plotted in Fig. 2, both for the conventional RF and the RF using the most energy efficient PPS technique. The equations listed for each case are approxi-

mate curve fits for the region $IW = 4$ to $IW = 1\ 6$

We see that there is a significant energy penalty per instruction in supporting large instruction level parallelism with a centralized register file. In order to compute the overall RF power we multiply the average energy dissipation per instruction $E_{average}$ by the number of instructions issued per cycle (IPC), and by the clocking rate f: $Power = f \times IPC \times E_{average}$. According to Fig. 2, $E_{average} \sim IW^{\beta}$, where β is between 0.96 and 1.8. Assuming that $IPC = IW^{\alpha}$ (or inversely, $IW = IPC^{1/\alpha}$), this yields $Power \sim f \times IPC \times IPC^{\beta/\alpha} = f \times IPC^{1+\beta/\alpha}$, or $\gamma = 1 + \beta/\alpha$.

If the IPC grew linearly with the issue width ($\alpha = 1$) then the use of the PPS register file architecture would result in the power dependence parameter γ close to a value of 2, while the use of the conventional register file architecture would result in $\gamma = 2.8$. However, since in real machines IPC increases less than linearly with the increase in the issue width ($\alpha < 1$), then even the use of the most energy-efficient register file architecture does not solve the problem, and leaves the power dependence parameter γ well above 2. For an Amdahl's law-like α of 0.5 [14] the γ is between 3 and 4.6!

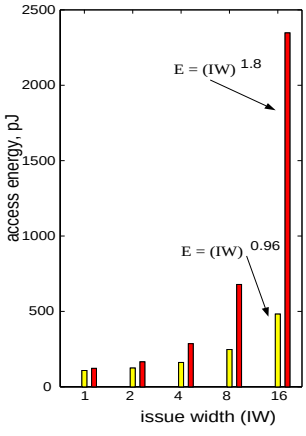


Figure 2: Average access energy per instruction for a PPS (1st bar) and a conventional RF architecture (2nd bar) versus issue width, 0.5μ feature size, $V_{dd} = 3.3V$.

It must also be stressed that all the results for the register file power described above assumed very aggressive energy management techniques, including pulse word line activation technique for reducing the bit line swing to the minimum, pulse activation of the sensing circuitry, fully cutting off precharge during reading and writing, taking advantage of the statistics of the data stored in RF memory cells, minimum transistor sizing wherever possible, use of equalizing transistors to save bit line energy during precharge. In real world CPUs the desire for speed will often not permit some of these techniques, meaning that real register file powers are even higher. The conclusion is that none of the known circuit techniques solves the problem of rapid RF power growth for machines with increasing ILP. This leads to the inescapable conclusion that the use of a centralized register file as an inter-instruction communication mechanism is going to become prohibitively expensive. Alternative techniques involving more than just circuit tricks are going to be necessary, and are a target of this work.

3 Register File Partitioning – the Basic Idea

In this section we analyze potential energy savings of replacing a centralized RF with a collection of decentralized register files. The primary advantage of accessing a collection of local register files rather than a single centralized register file is that local register files do not need to have many ports and, the number of entries in each register file is much smaller, which makes them very simple, fast and low power.

Suppose we have a superscalar processor capable of issuing IW instruction in parallel. Using the above assumptions, a conventional centralized register file would have $N_{read,centr} = 2IW$, read ports and $N_{write,centr} = IW$ write ports, and $N_{reg,centr} \approx N_0 + 1\ 2\times IW$ physical registers [4]. Ideally, we would like to partition the centralized register file into m local register files ($m< IW$) in such a way that every local register file has $N_{ports,local} = \frac{1}{m} N_{ports,centr}$ ports, and $N_{reg,local} = \frac{1}{m} N_{reg,centr}$ entries.

Under such ideal partitioning of the register file we would expect the access energy to every local register file to be $E_{local} = \frac{1}{m^{1.8}} E_{centr}$, based on the results for the conventional register file architecture, presented in the previous section. Such a partitioning would result in the total register file power of $Power = f \times IPC \times E_{local} = f \times IPC \times \frac{1}{m^{1.8}} E_{centr}$. Choosing the degree of RF partitioning to be proportional to the processor issue width, $m \sim IW$, we get for the total register file power $Power \sim f \times IPC$, with the power growth parameter $\gamma = 1$. This result is achieved *without* using complicated circuit techniques for the register file.

However, there are certain hurdles that make the described ideal partitioning impossible. First, additional paths need to be provided to pass data between local register files, resulting in extra ports to local register files. Second, the ideal $\frac{1}{m}$ partitioning of registers among local register files is not possible, rather, local register files will need more registers than $\frac{N_{read,centr}}{m}$. These effects potentially reduce the energy savings.

4 Existing Approaches to Register File Decentralization

The problem of the growth of centralized register files and the associated increase in access time has been considered by some researchers, and a few architectures with a decentralized register file have been proposed. However, thus far researchers have been primarily concerned with access times, not energy costs.

4.1 Limited Connectivity VLIWs

The ideal VLIW architecture would be a machine with many functional units connected to a single central register file [3]. This organization would enable any operation to be carried out on any available function unit, simplifying code generation. To avoid the degradation in performance caused by multi-porting a single register file researchers have suggested partitioning the register file into banks, so that each functional unit is connected to a specific bank [3, 15]. Some kind of crossover provides a limited access to the registers from other banks. Every operation specifies its destination bank at compile time.

This arrangement alleviates the problem with the number of ports, but brings about the problem of inter-bank data traffic. If value stored in one bank is required for an operation scheduled to a functional unit connected to another bank, this value must be copied between the register file banks. To minimize the additional workload on the compiler and minimize data traffic between the different register files banks, the partitioning of the register file must be done in a very careful way.

Since the assignment of an operation to a functional unit is done at compile time, VLIW compiler must carefully choose

a cluster for every operation and a register file bank where the result will be written.

4.2 Multiscalar Architecture

The Multiscalar architecture [1, 6, 7] executes code segments identified at compile time in parallel on multiple processing elements organized as a circular chain. A decentralized realization of a single register file called a Multi-Version Register File is proposed to provide a high bandwidth inter-instruction communication mechanism needed for simultaneously active multiple execution units. Each execution unit is provided with a different version of the architectural register file, called a local register file. All register reads and writes occurring in a unit are directed only to the unit's local register file. Communication between the multiple register files is done through unidirectional serial links by forwarding the last possible register instances in a task to the subsequent tasks as soon as the instances are determined to be the last possible ones. The identification of the last possible register instance is done using a compiler support.

Finding appropriate static program chunks and load balancing across the processing elements, as well as identification of the last possible register instances, are some of the issues that arise in this approach.

4.3 Trace Window Architecture

The partitioning of the physical register file in the Trace Window architecture [16] is based on the fact that not all physical registers are live beyond the trace line that produces them. Typically, around half the registers written by a small trace line (16 or 24 instructions) are not live beyond the trace line. The proportion of such local registers increases for larger trace lines [5].

In the Trace Window architecture, locally live destination registers are renamed to a physical register file local to the trace line; live-on-exit destination registers are renamed to a global register file. To identify live-on-exit registers, output-dependence checking among instructions in one trace line is done in parallel with the true-dependence checking. The local RF is flushed when the corresponding trace line is removed from the window.

One disadvantage of this approach is that in order to rename registers into local and global register files, all chunks of a trace line need to be buffered at the rename stage's output until the last chunk is renamed. As a result, register renaming of a trace line restricts dispatch bandwidth, and identification of live-on-exit registers makes instruction dispatch bursty. A mechanism for alleviating this problem is proposed which avoids full renaming on trace reuse by capturing in the trace cache renamed trace lines, obtained from the output of the register rename stage, rather than raw instructions.

5 Split Register File Architecture

The first of the above techniques makes the register file partitioning visible to the programmer and requires that all assignments of operands to register file banks be done explicitly at compile time. This approach potentially has binary compatibility problems. The two other techniques are based on the time localities of inter-instruction communication. Each bank of the register file stores those register instances that are produced and consumed by instruction that are executed close to each other in time in the dynamic instruction sequence.

In our work we study an approach to the register file decentralization based on another kind of inter-instruction communication locality. This approach is based on a hypothesis that there exist certain groups of instructions in the dynamic instruction sequence such that the inter-instruction communication is likely to be mostly local within each group. These groups do not necessarily consist of consecutive instructions in the dynamic instruction sequence.

If this is the case, then we can implement our CPU as a collection of processing unit clusters and provide each cluster with a local physical register file. At run-time, instruction despatch logic tries to steer every instructions to the cluster where instructions producing its register source operands were executed. The desired result of such partitioning would be that instructions access local register files most of the time. Additional paths are provided for inter-cluster traffic.

There are additional energy benefits we expect to get from the described organization. First, the energy efficiency of the bypass logic can be improved if we implement full bypassing within each cluster only. Second, we expect that the proposed idea should allow us to more fully take advantage of correlation between operand values of instructions in the same group. The average energy dissipation of most processing units like ALU, shifter, etc. is known to be statistically proportional to the number of data bit transitions at the inputs in a clock cycle, $E_{average} = E_{const} + N_{change} \times E_{change}$, where E_{const} is a constant energy independent of the data activity at the inputs, N_{change} is the average number of bit transitions at the inputs in one clock cycle, and E_{change} is a coefficient specific for every processing unit [9, 12]. By steering instructions that access the same registers to the same processing unit cluster we increase the probability that operand values of these instructions are correlated, and thereby reduce the average number of bit transitions at the inputs of functional units.

The suggested split register file architecture is similar to the dependence-based architecture that was proposed in [10] to simplify wakeup and select logic (which was found to be the most critical in terms of the delay complexity), and thus allow faster clocking. In the dependence-based architecture the instruction issue window is replaced with a number of FIFO buffers, each queue holding a chain of dependent instructions. Instructions from multiple queues are issue in parallel. The key difference between our approach and the dependence-based architecture is that the register file in the dependence-based architecture is not partitioned into register files local to clusters. Also, the dependence-based architecture has not been studied for energy efficiency in [10].

Before going any further the efficiency of the idea must be evaluated. The primary questions that determine the efficiency of splitting the register file are: (i) how often we need to pass data between different clusters (which is equivalent to how often accesses to remote register files will occur, or how many additional write ports might be needed for inter-register file communication), (ii) how many registers in local register files are needed, and (iii) what effect such partitioning has on correlation between operand values of instructions issued to the same functional unit.

6 Early Experiments

In this section we check the hypothesis that "there exist certain groups of instructions in the dynamic instruction sequence such that the inter-instruction communication is mostly local within each group." For this we need to find the best possible partitioning of instructions in the dynamic code sequence into groups such that inter-instruction communication across group boundaries is minimized. This would give us a theoretical lower bound on the inter-register file traffic when instructions from different groups are scheduled to access different register files. We are going to use these data to evaluate the efficiency of various reg-

ister file decentralization techniques that we are going to study later.

Our approach to identifying the best possible partitioning of instructions is as follows. First, we construct a graph for a dynamic instruction sequence. In this graph nodes represent instructions in the dynamic sequence, and edges represent inter-instruction dependencies. We take into account dependencies trough registers, through memory locations, through the the condition code and Y registers. To take into account control dependencies, we make instructions in the graph depend on all mispredicted branches executed earlier in the dynamic code sequence. Correctly predicted branches do not introduce any control dependencies. Special care is taken to account for delayed instructions.

To find the best possible partitioning of instructions into groups such that inter-instruction communication across group boundaries is minimized, we need to solve the problem of optimal partitioning of the program graph into subgraphs in such a way that the total number of crossed edges representing dependencies through registers is minimized. Unfortunately we cannot solve this problem for the whole graph because it is known to be NP complete. We overcome this difficulty by dividing the problem into subproblems of fixed size, finding the optimal solutions for these subproblems, and thus constructing a sub-optimal solution for the whole problem. To argue that our suboptimal solution is close to the optimal one we observe the dependence of the result upon the size of subproblems.

The subproblems of optimal graph partitioning are posed as follows: Assume we have M execution units and N instructions in the analysis window (subproblem size). We need to schedule these N instructions to the M execution units in such a way that the total number of crossed edges representing dependencies through registers in the whole graph from the beginning of a program to the last instruction in this analysis window is minimized. This optimization target is referred to as traffic cost minimization.

However, the traffic cost minimization contradicts the condition of exploiting maximal amount of instruction level parallelism, because the desire to minimize the number of crossed edges would result in the tendency to schedule most of the instructions to the same execution unit, leaving the rest of the hardware idle. The optimum would be achieved by scheduling all instructions to the same unit, thereby crossing no edges at all. Therefore, we need to take into account the performance requirement, namely that the total execution time from the beginning of program to the last instruction in the window must be minimized. Because of this requirement, independent instructions will tend to be scheduled to different execution units. The performance optimization target is further referred to as delay cost minimization.

There is one more requirement that we need to take into account when searching for the optimal partitioning of instructions. Namely, the number of physical registers in every register file needed for execution should be minimized. This requirement is equivalent to reducing the number of live registers in every group of instruction, and it somewhat conflicts with the performance requirement by reducing the degree of out-of order issue. Also it results in more even partitioning of instructions among execution units. The target of minimizing the number of physical registers is further referred to as storage size cost minimization.

To take into account all these requirements we take the following approach. First we fill the analysis window with incoming instructions. Then, we consider all possible assignments of the instructions in the analysis window to execution units, and calculate three costs for every assignment i.e. the traffic cost, delay cost, and storage size cost. Then we calculate the total cost of every assignment as some function of the three costs above, choose the assignment with the minimum total cost, and fill the analysis window with new instructions. We can place emphasis on any of the above requirements (either communication locality, or performance, or storage size), by constructing appropriate cost functions. The easiest way is to sum the three costs above with appropriate weights. At this point we assume that all execution units are equivalent, and every instruction can be scheduled to any of the execution units. Within every execution unit instructions are issued in-order, thus there are M^N possible combinations.

Results of applying the described optimization algorithm to a few short integer programs are presented in Figures 3, and 4 for processors with four and eight execution units, respectively. In this and other experiments in this section we used the basic two-bit branch predictor which has about 10% misprediction rate on the integer benchmarks. The branch misprediction penalty was assumed to be 5 cycles, and the instruction and data cache hit rates were assumed to be 100%.

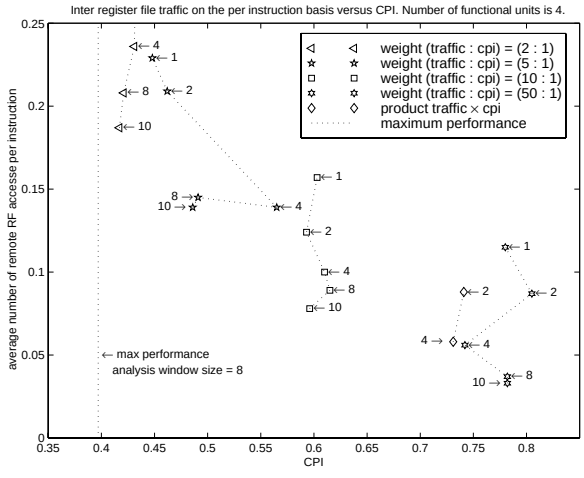


Figure 3: Inter-register file traffic on a per-instruction basis versus CPI, for different optimization targets and different sizes of the analysis window. The number of execution units is four.

Results are presented by dots, where each dot corresponds to a certain optimization target, and analysis window size. The X axis shows CPI, and the Y axis shows the average number of remote register file accesses per instruction. Thus, points closer to the origin of the coordinates represent better scheduling. Points corresponding to the same optimization target are connected with dotted lines. We see that bigger analysis windows result in a better scheduling policy. Unfortunately we could not run the simulation for larger analysis windows because of execution time constraints.

We did not observe significant saturation of improvement in scheduling with the increase in the analysis window size within the range that we could simulate, although we can see that benefit of increasing the size of the analysis window diminishes with the growth of the analysis window. We expect to see a more significant saturation of scheduling improvement for the analysis window sizes around 32 instructions, because most of the register instances are consumed within 32 instructions [5].

The results show that different optimization target result in quite different traffic-delay tradeoffs. By increasing the weight of the traffic cost we place more emphasis on minimization of the traffic cost, however, this also results in performance decrease (higher CPI). The corresponding points tend to concentrate in the lower right corners of the figures. On the other hand, by in-

creasing the weight of the delay cost we place more emphasis on performance optimization, but at the expense of higher traffic cost. The corresponding points tend to concentrate in upper-left corners of the charts. The vertical dotted lines on the charts correspond to the maximum performance scheduling, when the traffic cost is ignored. We also observed a significant space for the traffic-delay tradeoffs for configurations with 2 and 16 functional units.

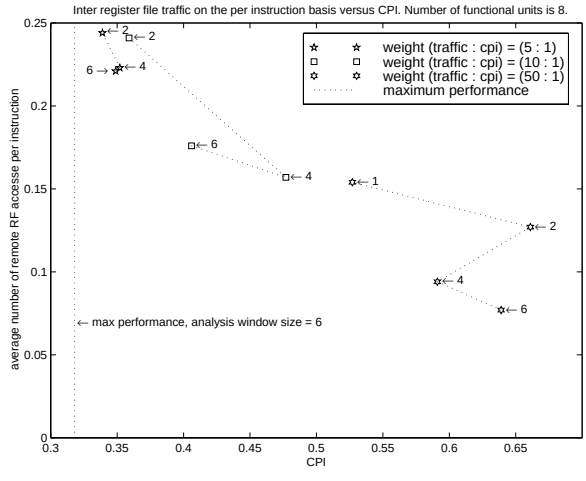


Figure 4: Inter-register file traffic on a per-instruction basis versus CPI, for different optimization targets and different sizes of the analysis window. The number of execution units is eight.

As a result of running these experiments we see that even with the existing compiler which performs optimizations for performance only, it is possible to partition instruction into groups such that inter-instruction communication tends to be mostly local within each group. By scheduling instructions in different groups to different execution units it is possible to keep the inter-unit communication low, while exploiting almost all the ILP available in the code.

We also checked the hypothesis that "steering instructions that access the same registers to the same processing unit cluster increases the probability that operand values of these instructions are correlated". Indeed, we observed a 9% to 10% reduction in the data switching activity at the inputs of functional units for the suboptimal scheduling algorithm described above. The corresponding reduction in the data switching activity at the outputs of the functional unit was from 12% to 14%. Runs with higher weight of the traffic cost resulted in more significant reductions in data switching activity. The observed reduction in the switching activity could result in up to 10% reduction in the dissipated power of the corresponding functional units [9, 12]. The reduction in the switching activities at the inputs and outputs of local register files was even more significant, up to 17%. This could be used to further reduce the access energy to local register files [17].

In the experiments described above we analyzed properties of code, avoiding implementation details as much as possible. As a next step we plan to come up with certain implementation that would allow us to take advantage of the discovered properties. We are going to analyze the inter-instruction traffic in more detail and study how the inter-group traffic could be handled. We will study the use of a global register file for storing register instances that are alive for long periods of time and extensively accessed by many instructions.

7 Opcode-Based Steering

In practice various heuristics can be used for steering instructions to clusters, and the simplest of them is the opcode-based steering, when the cluster for every instruction is chosen based on the instruction opcode. To evaluate the efficiency of the opcode-based steering we set up an experiment, based on the SPARC instruction set simulator Shade [2]. The methodology of the experiment was as follows: Each entry in the simulated physical register array has a tag that shows in which register file the data is being stored. All read accesses to register files are recorded in a matrix. Whenever unit i reads data from register file j, the entry $M[i][j]$ in the matrix is incremented. The values in the matrix are divided by the total number of instructions executed, so that diagonal and non-diagonal elements in the matrix represent local register file hit and miss rates, respectively, on a per-instruction basis. The sum of all elements in the matrix is the average number of non-$g0$ register file accesses by one instruction. Accesses to the zero register $g0$ are ignored.

In our first experiment we divided all instructions of the Sparc architecture into four groups: load/store instructions, branches, ALU instructions, and processor control instructions, including call, jmpl and return. We assumed that there is a separate cluster for every group, with a local register file. Note that early CDC and Cray machines often had multiple architecturally dedicated register files similar to this partitioning.

In this experiment we assumed that if a unit needs to access data from a remote RF, the data is moved from that remote register file to the RF of the unit requesting the data (move-on-miss policy). Therefore, non-diagonal entries in the matrix show how often such inter-register file transfers are needed (in terms of accesses per instruction). For example, M[LOAD/STORE][ALU] shows how often data needed to be transferred from the ALU register file to the LOAD/STORE register file, on a per-instruction basis.

The first approach that we simulated is a "lazy" placement policy. Every unit writes data to its local register file. If another unit needs this data, the data is moved to the corresponding register file. The results of using this policy is that too many inter-register file data transfers are needed: we observed local register file misses in up to 30% of instructions executed. Thus, this placement policy is not suitable for the split register file architecture.

The second approach we simulated is a "greedy" placement policy, when data is always written to the register file local to that unit which will need the data first (in the dynamic instruction sequence). If another unit needs this data later, the data is moved to the register file local to that unit. The results for the greedy placement policy averaged over a set of integer bench program are given in the Table 1.

Instruction Groups	Number of accesses to the local register files (per instruction)			
	LOAD/STORE	BRANCH	PROC. CONTR.	ALU
LD/ST	0.223	0.000	0.000	0.027
BRANCH	0.000	0.000	0.000	0.000
CONTR.	0.000	0.000	0.025	0.000
ALU	0.046	0.000	0.002	0.535

Table 1: Inter-register file traffic matrix for the greedy placement policy.

The third approach we simulated is an "optimal" placement policy, when data is always written to the register file local to the unit that will access this data for the most number of times. Unlike the two previous placement policies, if another unit needs

to access this data, the data is not moved from the register file. The results for the optimal placement policy averaged over the same set of integer bench program are given in the Table 2.

Instruction Groups	Number of accesses to the local register files (per instruction)			
	LOAD/STORE	BRANCH	PROC. CONTR.	ALU
LD/ST	0.213	0.000	0.000	0.037
BRANCH	0.000	0.000	0.000	0.000
CONTR.	0.000	0.000	0.023	0.002
ALU	0.026	0.000	0.000	0.557

Table 2: Inter-register file traffic matrix for the optimal placement policy.

Of course, it is not always possible for either hardware or compiler to predict exactly what unit will need the data first or which unit will access the data the most extensively, however these results show us the lower bound on the register file miss rates for the opcode-based steering scheme. The main result of this experiment is that most of the register file accesses hit local register files. The total register file miss rate is around 7.5 percent on a per-instruction basis, which means that about 7.5% of all instructions miss their local register files. An interesting observation is that the greedy and optimal placement policies result in approximately the same total miss rate. Note that this result was obtained for a code compiled with a regular optimizing compiler. If we use a compiler with proper optimizations, then the results will be better.

Most inter-register-file traffic occurs between ALU and Load/Store units. These register file misses are responsible for about 90% of the total miss rate. The percentage of these misses is higher for an optimized code than for a non-optimized code. This happens because an optimized code uses registers more efficiently for inter-instruction communication. A trivial way to eliminate this kind of misses would be to broadcast results of the instructions in these groups to both register files, however, this might not be energy efficient. In future experiments we will study the energy efficiency of such a broadcasting.

8 Conclusions and Future Work

Future growth in performance for modern CPUs is predicated on higher and higher levels of instruction issue. However, even with the best of circuit techniques known the centralized register files needed to support such machines will rapidly become a bottleneck in the power equation. This paper has suggested one path towards reducing this factor by architecturally partitioning the register file into logically and physically separate units. Early simulation results has indicated that the amount of inter-unit traffic that needs to be handled to allow such partitioning may be quite manageable - validating that the basic concept may hold substantial promise, and opening up an opportunity to trade performance for power by sharing write ports instead of adding ports dedicated to inter-register file traffic.

Near term work will involve in firming up the simulation results with more extensive benchmarks and more detailed simulations, along with more complete power models that incorporate more of the CPU. Longer term work will focus on alternative register partitioning schemes, more general that those suggested in the multi-scalar and trace window work, but again with an emphasis on inherently lower power organizations. In addition second order effects which may be significant, such as increased correlation between bits processed by function units, are also part of the simulation plans and will be investigated extensively.

The long term goal for this work is thus determination of architectural techniques, perhaps all the way at the instruction set level, that permit inherently low energy organizations which in turn are not significantly performance constrained.

References

[1] S. E. Breach, T. N. Vijaykumar, and G. S. Sohi, "The Anatomy of the Register File in a Multiscalar Processor," *Proc. 27th Annual Int'l Symp. on Microarchitecture*, pp. 181-190, December 1994.

[2] B. Cmelik, "The SHADE Simulator," Sun-Labs Technical Report, 1993.

[3] R. Colwell, et al., "A VLIW Architecture for a Trace Scheduling Compiler," *IEEE Transactions on Computers*, pp. 967–979, NO. 8, August 1988.

[4] K. Farkas, N. Jouppi, P. Chow, "Register File Design Considerations in Dynamically Scheduled Processors." Technical Report 95/10, Digital Equipment Corporation Western Research Lab, November 1995.

[5] M. Franklin and G. S. Sohi, "Register Traffic Analysis for Streamlining Inter-Operation Communication in Fine-Grain Parallel Processors," *Proc. 25th Annual Int'l Symp. on Microarchitecture*, pp. 236-245, 1992.

[6] M. Franklin and G. S. Sohi, "The Expandable Split Window Architecture for Exploiting Fine-Grain Parallelism," *Proc. 19th Annual Int'l Symposium on Computer Architecture*, May, 1992.

[7] M. Franklin, "The Multiscalar Architecture," *Ph.D. Thesis, University of Wisconsin-Madison*, Tech. Report 1196, November 1993.

[8] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 9, September 1996.

[9] H. Kojima, et al.,"Power Analysis of a Programmable DSP for Architecture/Program Optimization,"*IEEE Symposium on Low Power Electronics*, San Jose, pp. 26–27, October 1995.

[10] S. Palacharla, N. Jouppi, J. Smith, "Complexity-Effective Superscalar Processor." In: *Proceedings of the 24th Annual International Symposium on Computer Architecture*. pp. 206–218, June 1997.

[11] U.S. Patent No. 5,657,291, issued Aug. 12, 1997 to A. Podlesny, G. Kristovsky, A. Malshin, "Multiport Register File Memory Cell Configuration for Read Operation."

[12] T. Sato, et al.,"Evaluation of Architecture-level Power Estimation for CMOS RISC Processors,"*IEEE Symposium on Low Power Electronics*, San Jose, pp. 44–45, October 1995.

[13] M. Tremblay, B. Joy and K. Shin, "A Three Dimensional Register File For Superscalar Processors." In: *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*. pp. 191–201, January 1995.

[14] M. Tremblay, D. Greenley and K. Normoyle, "The Design of the Microarchitecture of UltraSPARC TM-I," *Proceedings of the IEEE*. pp. 16531–1663, December 1995.

[15] J. Turley and H. Hakkarainen, "TI's New 'C6x DSP Screams at 1,600 MIPS," *Microprocessor Report*, pp. 14–17, February 17, 1997.

[16] S. Vajapeyam and T. Miltra, "Improving Superscalar Instruction Dispatch and Issue by Exploiting Dynamic Code Sequences," *Proc. 24th Annual Int'l Symposium on Computer Architecture*, June, 1997.

[17] V. Zyuban, P. Kogge, "The Energy Complexity of Register Files," *IEEE Symposium on Low Power Electronics and Design*, Monterey, August 1998.

ENERGY EFFICIENT CACHE ORGANIZATIONS FOR SUPERSCALAR PROCESSORS*

Kanad Ghose and Milind B. Kamble
Department of Computer Science
State University of New York
Binghamton, NY 13902–6000
email: {ghose, kamble}@cs.binghamton.edu

Abstract

Organizational techniques for reducing energy dissipation in on–chip processor caches as well as off–chip caches have been observed to provide substantial energy savings in a technology independent manner. We propose and evaluate the use of block buffering using *multiple* block buffers, subbanking and bit line isolation to reduce the power dissipation within on–chip caches for superscalar CPUs. We use a detailed register–level superscalar simulator to glean transition counts that occur within various cache components during the execution of SPEC 95 benchmarks. These transition counts are fed into an energy dissipation model for a 0.8 micron cache to allow power dissipation within various cache components to be estimated accurately. We show that the use of 4 block buffers, with subbanking and bit line isolation can reduce the energy dissipation of conventional caches very significantly, often by as much as 60–70%.

1. Introduction

Most high–performance microprocessors have one or two levels of on chip caches to accommodate the large memory bandwidth requirements of the superscalar pipeline. A significant amount of power is dissipated by these on–chip caches, as exemplified in the following:

(a) The on–chip L1 and L2 caches in the DEC 21164 microprocessor dissipate about 25% of the total power dissipated by the entire chip [ERB+ 95].

(b) In the bipolar, multi–chip implementation of a 300–MHz. CPU reported in [JBD+ 93], 50% of the total dissipated power is due to the primary caches.

(c) A recently announced low–power microprocessor targeted for the low power, the DEC SA–110, medium performance market (that also leads all microprocessors available today in terms of SPECmarks/watt) dissipates 27% and 16% of the total power, respectively, in the on–chip I–cache and D–cache respectively [Mon 96].

As yet another example, the HP PA 8500 is significantly cache–rich compared to other similar high–end microprocessors and reportedly more than 70% of its die area is occupied by the on–chip L1 caches, which are likely to be a major source of power dissipation. Several factors result in significant power dissipations in on–chip caches. First, their tag and data arrays are implemented as SRAMs, to allow the cache access rate match the pipeline clock rate. Second, the cache area is more densely packed than other areas on the die, so that the number of transistors devoted to the cache are quite a significant percentage of the total number of transistors on the die. The above examples suggest that it is imperative to seek techniques that reduce the power dissipated in on–chip caches without compromising the performance of the caches.

* This work is supported in part by the National Science Foundation through award No. MIP–9504767 and by the IEEC at SUNY–Binghamton

The bulk of the energy dissipation in CMOS circuits comes from transitions in the logic levels at gate outputs. The power dissipated by a CMOS gate due to output transitions is given by $P = 0.5 \cdot C_L \cdot V^2 \cdot f$, where C_L is the capacitative load driven by the gate, V is the voltage swing per transition and f is the transition frequency. In addition to these capacitative dissipation, circuits in real caches (such as the sense amps) also have non–capacitative dissipations, that also depend on the number of transitions. Several techniques have been proposed (and actually used in some cases) for reducing the power dissipated by caches in general. These techniques can be divided into the following categories:

1) Techniques that use alternative cache organizations for reducing cache power dissipation. These include the use of subbanked caches [SuDe 95, KaGh 97a], block buffers [SuDe 95, KaGh 97a], their combination and multimodular external caches [KBN 95]. Many techniques for organizing SRAMs for lower power dissipation, as presented in [Itoh 96] and [EvFr 95] can also be used to reduce dissipation in the tag and data arrays within a cache.

2) Circuit design techniques that focus on reducing the power dissipation within the static RAM bit cells, particularly dissipations due to bit line transitions. These include bit line isolation techniques that isolate the sense amp from the bit lines once the sense amp has started making a transition, as used in the Hitachi SH–3 embedded microprocessor [HKY+ 95], clamped bit lines that limit bit line swings and other similar techniques.

3) Instruction scheduling techniques that schedules instructions in a manner that reduces power dissipation in instruction caches [SuDe 95].

Note that techniques in these three categories are generally independent from each other and can thus be used in conjunction. The focus of this paper is on techniques that fall into the first category.

In this paper we propose and evaluate the use of *multiple* block buffers, subbanking and bit line isolation to reduce the power dissipation within on–chip caches for superscalar CPUs. We show that a suitable combination of these techniques can be extremely effective in reducing the energy dissipation in all caches across the memory hierarchy, both on–chip and off–chip. (Power savings for off–chip caches and other techniques for reducing off–chip cache power are discussed in a separate paper.) Our approach is to use a detailed register–level simulator for a superscalar processor and cache hierarchy to glean transition counts within key cache components as the simulator executes SPEC 95 benchmarks. These transition counts are then fed into an energy dissipation model for the major cache components [KaGh 97b, KaGh98], with capacitative coefficients obtained from [WiJo 94] for a real cache implementation, to determine the energy dissipations are realistically as possible.

The rest of this paper is organized as follows. In Section 2, we describe the energy efficient architectural techniques such as block buffering, subbanking, multiple block buffers and bit–line isolation. These are elaborated in earlier papers [KaGh 97a,

KaGh97b, GhKa 98]. In Section 3, we elaborate on the experimental setup used for our measurements. The energy dissipations for the proposed cache configurations, as obtained from the simulated execution of SPEC 95 benchmarks are discussed in Section 4. The main conclusions are given in Section 5. The Appendix summarizes the energy dissipation model of the set associative caches.

2. Organizing Caches for Energy Efficiency

The most common cache organization employed in modern microprocessors today is the set–associative cache [Smith 82]; the direct–mapped cache and fully associative cache organizations are two extremes of the set–associative organization. In a normal *m*–way set associative cache, there are *m* tag and data array pairs, each consisting of S rows, where $S = 2^s$. Each data array location is known as a cache line, which is a group of W consecutive words, W being a power of 2 ($W = 2^w$). A cache line is thus capable of holding the contents of what is called a memory block, consisting of W consecutive memory words. Tag and data array locations at the same offset within the tag and data arrays make up what is called a *set*. The placement rules for such a cache dictates that a word at an address A in the memory (RAM), if present in the cache, can be found in any cache line within the set at offset (A div W) mod S. To uniquely identify the memory block that resides within a cache line, the tag part of the address (obtained by stripping the lower order (s+w) bits of A) are kept in the associated tag array location. The access steps for the m–way set–associative cache are thus as follows:

Step 1: Use the middle order s bits in the address of the word to be accessed to read out the set that potentially contains these words into output latches of the tag and data arrays. These latches together make up what is called a **block buffer** (aka **line buffer**).

Step 2: Compare the tag part of the address being accessed in parallel with the m outputs from the tag arrays (using m independent comparators). A match with the the output of the tag array indicates that the required data is within the output latch of the corresponding data array. If this is the case, a situation called a cache hit, the lower order w bits of the address are used to multiplex out the desired word from the data array output latch. If no match occurs, we have a cache miss, implying that the desired data is not within the cache.

In most modern caches, the two steps outlined above are implemented in 2 (or more) clock cycles. In the pipelined caches that we have designed in our simulator, the two steps mentioned above are naturally divided into 2 stages, thereby providing a theoretical throughput of one request per cache cycle (per port). The tag and data arrays need to be updated when a missing line is fetched or when a STORE request updates the contents of an existing line. This update of the arrays is performed by a third stage of the cache pipeline. On a cache miss, some replacement algorithm – implemented in hardware – is used to select a victim line from the set read out in step 1 and appropriate steps are followed to install the memory block into the victim's line frame.

For a superscalar CPU, to maximize the number of instructions to be examined for dispatch in each cycle, a facility is needed to transparently step across the line boundary to the physically next line (if that line is cached). One way of doing this is to use a deck buffer mechanism (as used in the MIPS 10K) or to use odd and even cache banks (with an automatic line address incrementation facility, as originally used in the IBM Risc/6000). Modern superscalar CPUs also tend to support multiple pipelined load/store units that can request access to the D–cache simultaneously. This requirement is met by using a multiported cache or a interleaved cache. In the power studies reported here, we assume that the L1 I–cache uses odd–even cache banks, while the L1–D cache is multiported.

2.1 Set–associative cache with a single block buffer

Figure 1 depicts a two–way set–associative cache augmented with a single block buffer (i.e., one set of tag and data latches for the various cache ways) that does not prolong the cache cycle time. The components added to the two–way set associative cache to incorporate block buffering are shown highlighted in a grey background. The cache shown in Figure 1 is accessed with a 2–cycle latency but at the rate of one access per cycle, exactly like a conventional set–associative cache. These block buffer augmented caches exploit the spatial locality of reference exhibited by all programs in general. If the current data being accessed is within the last cache line that was accessed, there is no need to fetch that line again from the data array of the cache. This not only saves accesses of the data arrays but, at the same time, also saves the access necessary to the tag arrays, since both the tag and data arrays have to be read to determine if the line containing the required data is within the cache.

The steps for accessing the set–associative cache with a single block buffer, using a 2–phase clock (phases $\phi1$, $\phi2$) are:

Cycle 1:

$\phi1$: – Precharge the tag and data arrays for a read access
– Start decoding of the set address applied to the arrays (This is exactly identical to what happens in normal set–associative cache.)

– Simultaneously, compare fields in the address being accessed with the set number of the previous access (to determine if the current access is to the same set as the previous one).

$\phi2$: – If the comparison succeeds (called a "block buffer hit"), abort the readout of the tag and data arrays. Otherwise, latch in the selected set into the array output latches.
– Move the set number for the current access into the latch that holds the set number for the last access made.

Cycle 2:

$\phi1$: – Perform the normal tag comparison, as in a normal set–associative cache.

$\phi2$: – Perform the word multiplexing on a cache hit, as in a normal set–associative cache.

Note also that the cache cycle time with block buffering does not go up from that of a normal set–associative cache; all performance characteristics (access rate, pipelining, number of pipeline stages) are maintained.

Block buffered caches were introduced by Su and Despain in a slightly different form in [SuDe 94]. In their design, the block address of the current access is first compared against the block address of the set that is currently resident in the block buffer. The normal access of the cache – including bit line precharging and row address decoding – is started only when a mismatch occurs. Consequently, this arrangement prolongs the cache access latency, a solution unattractive in practice. Second, if the cache access is pipelined in two stages, a completely new set of tag comparators are needed (along with a comparator for the set number) to allow a block buffer hit to be determined in parallel with the tag comparison step of a normal cache access for the *previous* access. Both of these problems are not present in our block buffering scheme.

2.2 Set–associative cache with multiple block buffers

The benefit of a block buffer can be further extended by using multiple block buffers, in effect using the block buffers as a miniature level 0 cache. Figure 2 depicts a set–associative cache

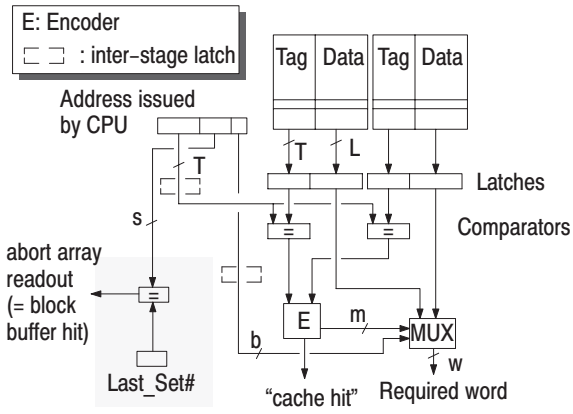


Figure 1. A 2–way Set Associative Cache with One Block Buffer

with multiple block buffers, in this case 4. Here, some additional considerations are needed due to the use of multiple block buffers. First, we need four latches to hold the number of the four most recently accessed sets. Second, four more comparators are needed to compare the set number field of the current access against the numbers of the sets sitting in the four block buffers. Third, if no block buffer hits occur, the data from the set selected by the current address applied to the cache must be retrieved into one of the four block buffers. Consequently, on a block buffer miss, a victim must be selected from one of the four block buffers. The need to write back an updated block buffer's content into the tag and data arrays, when the block buffer gets selected as a victim, is avoided by writing through updates to the block buffer into the corresponding data array on write accesses that result in a block buffer hit. Along with these, a multiplexing facility is needed to direct the outputs of the tag and data arrays into the victim block buffer. In this case, the four block buffers effectively make up a 4–entry fully–associative write–through cache, which is accessed in parallel with the normal cache without any prolongation in the cycle time or without any impact on the normal cache access pipeline.

The access steps for a cache with multiple block buffers are as follows:

Cycle 1:

ϕ1: – Precharge the tag and data arrays for a read access
– Start the decoding of the set address applied to the arrays (This is exactly identical to what happens in normal set–associative cache).

– Simultaneously, compare the set selector field in the address being accessed with the set numbers four the four sets stored in the four block buffers.
– Concurrently, identify a victim block buffer in advance to handle misses on the block buffers and start the setup of the multiplexor at the output of the tag and data arrays.

ϕ2: – If the set number of the current access matches the set number associated with any of the block buffers (as found in the set number latches), abort the readout of the tag and data arrays and steer the tag and data values from the matching buffer into the tag comparators and the word multiplexer, respectively.

Otherwise, if a hit did not occur on the block buffers, latch in the contents of the selected set from the arrays into the block buffer chosen as a victim in the previous clock phase, and move the set selector bits in the currently accessed address to the set number latch associated with this block buffer.

Cycle 2:

ϕ1: – Perform the normal tag comparison, as in a normal set–associative cache.

ϕ2: – Perform the word multiplexing on a cache hit, as in a normal set–associative cache.

The energy savings resulting in a set–associative cache with a multiple block buffers is due to reasons similar to that for a set–associative cache with a single block buffer. Multiple line buffers simply increase the probability of aborting the tag and data array accesses, so that bit line dissipations during the array readout are avoided.

In [KGM 97], Kin et al describe the use of a small "filter cache" that sits in front of a conventional L1 cache for reducing the power dissipation of the cache memory system. If a hit occurs in the smaller filter cache, data is accessed from the filter cache. The normal L1 cache access is started only after a miss has been detected in this filter cache. Consequently, the cache access latency is increased – this can have adverse impact on performance. Notice that although the multiple block buffers in our proposal behave as a fully associative cache and serves the same purpose as a filter cache, it does not impact the cache latency at all. This is because we probe the block buffers in parallel with the normal cache access. Also, unlike a fully associative cache, we need only one set of tag comparators and four set number comparators to detect a block buffer hit. (In a four entry fully associative cache four tag comparators would be needed)

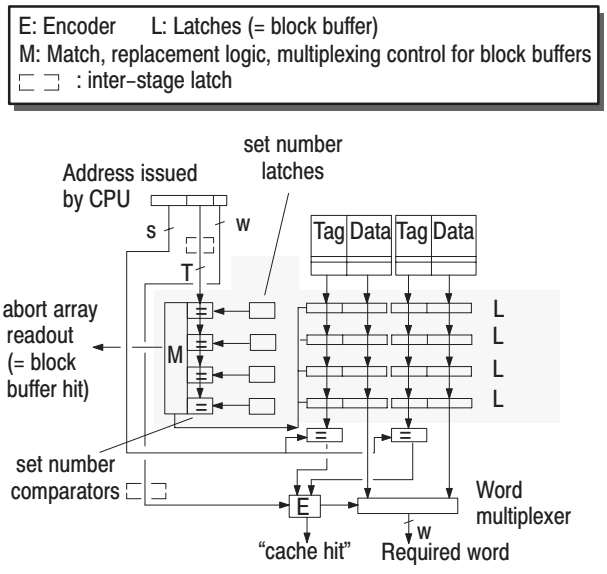


Figure 2. A 2–way Set–Associative Cache with Four Block Buffers

2.3 Bit–line isolation and subbanking

One of the major source of power dissipation in a cache has been shown to due to transitions in the bit lines of the data and tag arrays. Bit line dissipations occur when the bit lines are precharged or discharged. Bit line isolation represents one way of reducing the energy dissipations due to bit line transitions. Here, the sense amps sensing the bit lines are disconnected from the bit lines as soon as they start the transition on sensing. Consequently, when the sense amps recover the cleaner logic levels, these logic levels are not driven on the bit lines. With this arrangement, a small differential voltage on the bit lines is sufficient to trigger the full transition on the sense amp outputs, but because the sense amp output remains disconnected from the bit lines, the bit lines

themselves do not experience the full transition made by the sense amp. Considerable energy savings thus results. Bit line isolation has been used to reduce the power dissipation in the cache memory of the Hitachi SH–3 embedded microprocessor [HKY+ 95].

To achieve further savings on the bit line energy, the data arrays can be subdivided into subbanks, so that only those subbanks that contain the desired data can be readout [SuDe 95, KaGh 97a]. A subbank consists of a number of consecutive bit columns of the data array. A data line is thus spread across a number of subbanks. The size of a subbank refers to the physical word width of each subbank. Each subbank within a data array can be activated independently. By using an array of bit flags to indicate the presence/absence of subbanks in the block buffer, the array access stage can determine if a subbank needs to be read out for the current request. This again does not affect the cycle time nor the pipeline of the cache, while at the same time preventing readout of those subbanks from the data array which might never be needed.

3. Experimental Setup

We now describe the experimental setup used in our study of the energy efficiency achieved through the use of set associative caches with multiple block buffers, subbanking and bit line isolation as described in Section 2. We use a detailed register–level simulator, SCAPE, which accurately simulates at the cycle level a superscalar pipelined processor, based on the MIPS instruction set. SCAPE can be configured (using a configuration file) at run time to simulate a variety of cache hierarchies such as 2 or 3 levels of caches, split or unified caches etc. The primary cache organizational features such as size, associativity, block size and cache operating frequency can be configured individually for each cache used in the system. Apart from these, a variety of other cache organizational features that are important in the context of a superscalar processor can be individually tuned for each cache, such as the number of ports, multiple outstanding requests, writeback policy and sub–blocking. For the results presented here, we used a 3 level cache hierarchy, with split L1 caches, unified L2 cache and an unified L3 cache. The L1 and L2 caches were assumed to be on–chip and the L3 cache was assumed to be off–chip. Since superscalar CPUs need to fetch multiple instructions at a time to feed the instruction dispatch unit, the L1 I–cache was designed to supply multiple instructions per Ifetch request. To improve the I–cache bandwidth, the I–cache was designed to have *even–odd* directories (and banks) so that requests that spanned consecutive rows could still be completed in the same cycle. The presence of multiple LOAD/STORE functional units in the superscalar pipeline demanded the use of multiple ports for the L1 D–cache. All caches consisted of a 3–stage pipelined design where stage 1 performed that bit–line precharging and array access activities, stage 2 performed the tag compare, steer and miss handling activities and stage 3 performed the write activity into the array. We assume the layout and technology parameters for the 0.8µ cache described in detail in [WiJo 94].

The superscalar pipeline implements the MIPS 3000 ISA. The out–of–order execution engine uses a register update unit (RUU) which maintains the program order of the instructions to retire completed instructions. Data forwarding from completed but not retired instructions in the RUU as well as from functional units is employed to maximize instruction throughput. Multiple functional units are also used achieve dispatch of multiple instructions in the same cycle. Many parameters of the superscalar pipeline itself can be configured at run time, such as size of the RUU, number of functional units, degree of superscalar issue and number of physical registers used to implement register renaming). For our studies we used a 64–entry RUU and 4 instruction fetch and issue. We assumed that the following function units are present: 2 LOAD units, 1 STORE unit, 6 integer units, 2 Integer multiply/divide (pipelined)units and 2 Floating point (pipelined) units. SCAPE accepts any statically compiled executable runnable on a MIPS R3000 processor. We simulated the execution of the SPECInt95 and few SPECFp95 (su2cor, mgrid, applu) benchmarks to get a good mix of CPU–intensive and memory–intensive loads.

For our base case, we assume a 32 Kbyte, direct–mapped L1 I–cache and a 32 Kbyte, 4–way set–associative L1 D–cache. The line sizes for both of these caches are set to 32 bytes, a typical number, with 16 byte subblock size. The L2–cache is assumed to be a 128 Kbyte, 4–way set–associative unified (i.e., shared by instructions and data) on–chip cache with a line size of 64 bytes and 32 byte subblock size. The 64 byte line size was chosen, since L2 caches must have a line size longer than that of the L1 caches to be effective. The off–chip L3–cache is assumed to be a 1Mbyte, 8–way set–associative unified off–chip cache with a line size of 128 bytes. The interconnection bus width was 32 bytes between L2 and L3, and 16 bytes between L1 and L2 and 64 bytes between L3 and the main memory. All the caches had write–back policy except L3 which was write–through with a 16 deep write back buffers. A buddy replacement algorithm, approximating LRU, was used for all the set–associative caches, as well as for choosing the victim block buffer on a block buffer miss.

We studied a variety of cache configurations, keeping the individual capacities of L1 I, L1 D, L2 and L3 caches constant, to study the effects of multiple block buffers and other energy efficient enhancements. These configurations are as follows (see Table I):

- L1 I–caches with associativities of 1 and 4. For each of these configurations we compared the energy saving through the use of simultaneously using 4–block buffers, subbanking and bit–line isolation against a conventional cache. For these configurations of L1 I–cache, the parameters of the L1 D–cache and L2–cache were kept the same as in the base case. The L3 cache is always maintained as a conventional cache with base case configuration.

- L1 D–caches with associativities of 2 and 4. For each of these configurations we compared the energy saving through the use of simultaneously using 4–block buffers, subbanking and bit–line isolation against a conventional cache. For these configurations of L1 D–cache, the parameters of the L1 I–cache and L2–cache were kept the same as in the base case. The L3 cache is always maintained as a conventional cache with base case configuration.

For the purpose of this paper, we used the capacitances for the 0.8 micron CMOS cache described in [WiJo 94]. A supply voltage of $V_{dd} = 5$ Volts was assumed. The voltage swing on the bit lines was limited to 500mV on each side of $V_{precharge}$ which was assumed to be $V_{dd}/2$. For the bit line isolation design however, the voltage swing was assumed to be 200mV on each side of $V_{precharge}$. The CPU pipeline (and the caches) was (were) assumed to be clocked at 120 MHz. The 0.8µ pipelined cache used for the case study can sustain this clock rate for a wide range of cache sizes based on the timings reported in [WiJo 94] where the cache was not pipelined. The L2 cache is operated at half this frequency.

4. Results and Discussions

Figures 3 and 4 depict how the power dissipation varies across the SPEC 95 benchmarks for a conventionally organized L1 I–cache and a conventionally organized L1 D–cache. Figure 5 shows how the power dissipation in the conventional, unified L2 cache varies over the same benchmarks.

	CACHE CONFIGURATIONS
I	I–cache: 32Kbyte, **direct–mapped**, 32 byte line,16 byte subblocks; D–cache: 32Kbyte, **4–way**, 32 byte line, 16 byte subblocks; L2–cache: 128Kbyte unified, 4–way, 64 byte line, 32 byte subblocks; L3–cache: 1Mbyte, 8way, 128 byte line, nonsubblocked (all caches conventional)
II	I–cache: 32Kbyte, **4–way**, 32 byte line, 16 byte subblocks; D–cache: 32Kbyte, **4–way**, 32 byte line, 16 byte subblocks; L2–cache: 128Kbyte, 4–way, 64 byte line, 32 byte subblocks; L3–cache: 1Mbyte, 8–way, 128 byte line, nonsubblocked (all caches conventional)
III	I–cache: 32Kbyte, **direct–mapped**, 32 byte line,16 byte subblocks; D–cache: 32Kbyte, **2–way**, 32 byte line, 16 byte subblocks; L2–cache: 128Kbyte unified, 4–way, 64 byte line, 32 byte subblocks; L3–cache: 1Mbyte, 8way, 128 byte line, nonsubblocked (all caches conventional)
IV	same as case I except, I–cache: 4 blockbuffers, 16byte subbanks, D–cache: 4 blockbuffers, 4byte subbanks L2–cache: 4 blockbuffers, 16 byte subbanks, L3–cache: conventional
V	same as case II except, I–cache: 4 blockbuffers, 16byte subbanks, D–cache: 4 blockbuffers, 4byte subbanks L2–cache: 4 blockbuffers, 16 byte subbanks, L3–cache: conventional
VI	same as case III except, I–cache: 4 blockbuffers, 16byte subbanks, D–cache: 4 blockbuffers, 4byte subbanks L2–cache: 4 blockbuffers, 16 byte subbanks, L3–cache: conventional

Table I. Cache configurations used in the evaluation

Figure 6 shows how the average power dissipation for the L1 I–cache (over the SPEC 95 benchmarks studied) decreases as we move from a conventional organization (labelled a) to an organization with 4 block buffers and subbanking (labelled b) and an organization with 4 block buffers and subbanking that additionally uses bit line isolation (labelled c). The power reductions that result are in excess of 66%. Figure 7 shows that better power improvements also occur for the L1 D–cache when identical enhancements are made – in this case the power savings approach 70%.

In Figures 8 and 9, we depict the power dissipations across the benchmarks for the two most power–efficient organizations for the L1 I–cache and the L1 D–cache respectively. A comparison of Figures 3 and 8 shows that the power savings for the L1 I–cache are quite consistent across the benchmarks when 4 block buffers, subbanking and bit line isolation are used. Similar conclusions are to be drawn for the L1 D–cache when Figures 4 and 9 are compared.

Figure 10 depicts the extent of overall power savings when organizational enhancements are made to all of the on–chip caches. Since the L1 I–cache has a relatively higher dissipation, the specific organization chosen for the L1 I–cache was the one that had the least power dissipation. The L1–Dcache and the L2 cache configurations were then chosen to minimize the overall power dissipation in *all* of the on–chip caches.

Figures 11 and 12 are useful in understanding where the power savings come from. For the conventional organizations, the energy spent in precharging and discharging the bit lines dominate. With the use of block buffers, the number of accesses to the actual cache arrays are reduced, reducing in turn the bit line dissipation component. When subbanking is additionally deployed, the number of bit lines that have to be driven, precharged or sensed in the data array gets further reduced. Additional savings come when the capacitative loading of the bit lines, as seen by the sense amp, is reduced through bit line isolation.

5. Conclusions

On–chip caches are a major source of power dissipation in contemporary superscalar microprocessors. The bulk of the energy dissipated in conventionally organized caches is in precharging, sensing and discharging the bit lines of the tag and data arrays. We proposed the use of alternative organizations, such as multiple block buffering, subbanking and bit line isolation to reduce the power dissipation in on–chip caches without compromising the cache cycle time (and other aspects of performance, such as the cache hit ratio and cache access rate). The power saving achieved ranged from 60% to 70%.

We are currently investigating other organizational and circuit enhancements to caches and on–chip memory systems for reducing power without sacrificing performance.

References

[EvFr 95] Evans, R. J. and Franzon, P. D., “Energy Consumption Modeling and Optimization for SRAM’s”, in IEEE Journal of Solid–State Circuits, Vol. 30, No. 5, May 1995, pp 571–579.

[HKY+ 95] Hasegawa, A. et al, “SH3: High Code Density, Low Power”, IEEE Micro magazine, Dec. 1995, pp. 11–19.

[Itoh 96] Itoh, K., “Low Power Memory Design”, in *Low Power Design Methodologies*, ed. by Rabaey, J. and Pedram, M., Kluwer Academic Pub., pp. 201–251.

[KBN 95] Ko, U., Balsara, P. T. and Nanda, A.K., “Energy Optimization of Multi–Level Processor Cache Architectures”, in Proc. of the Int’l. Sym. on Low Power Design, 1995, pp. 45–49.

[KaGh 97a] Kamble, M. B. and Ghose, K., “Energy–Efficiency of VLSI Caches: A Comparative Study”, in Proc. IEEE 10–th. Int’l. Conf. on VLSI Design, Jan. 1997, pp. 261–267.

[KaGh 97b] Kamble, M. B. and Ghose, K., “Analytical Energy Dissipation Models for Low Power Caches”, in Proc. 1997 Int’l Symposium on Low Power Electronics and Design, Aug. 1997, pp. 143–148.

[KaGh 98] Kamble, M. B. and Ghose, K., “Modeling Energy Dissipation in Low Power Caches”, Technical Report CS–TR–98–02, Dept. of Computer Science, SUNY–Binghamton 1998.

[KGM 97] Kin, J., Gupta, M. and Mangione–Smith, W.H., “The Filter Cache: An Energy–Efficient Memory Structure”, in Proc. MICRO 30, 1997, pp. 184–193.

[Larus 96] Larus, J., “SPIM: A MIPS 2000 SImulator”, available form Univ. Wis., CS ftp site.

[Mon 96] Montanaro, J. et al., “A 160 MHz, 32b 0.5 W CMOS RISC Microprocessor”, in IEEE ISSCC 1996 Digest of Papers, 1996.

[Smith 82] Smith, A. J., “Cache Memories”, ACM Computing Surveys, Sept. 1982, pp. 473–530.

[SuDe 95] Su, C. and Despain, A., “Cache Design Tradeoffs for Power and Performance Optimization: A Case Study”, in Proc. of the Int’l. Sym. on Low Power Design, 1995, pp. 63–68.

[WiJo 94] Wilton, S. E., and Jouppi, N., “An Enhanced Access and Cycle Time Model for On–Chip Caches”, DEC WRL Research Report 93/5, July 1994

[WaRa+ 92] Wada t., Rajan S., and Przybylski S.A., “An Analytical Access Time Model for On–Chip Cache Memories”, IEEE Journal ofSolid–State Circuits, Vol. 27, NO. 8, Aug 1992, pp. 1147–1156.

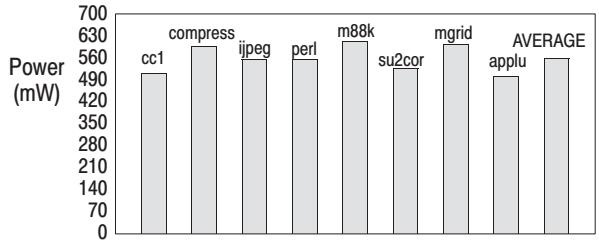


Figure 3. Power dissipations in the conventional **L1-Icache** across the SPEC95 benchmarks for Configuration I

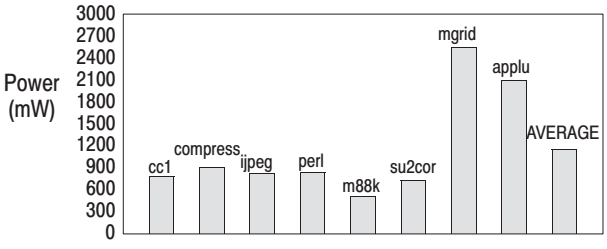


Figure 4. Power dissipations in the conventional **L1-Dcache** across the SPEC95 benchmarks for Configuration I

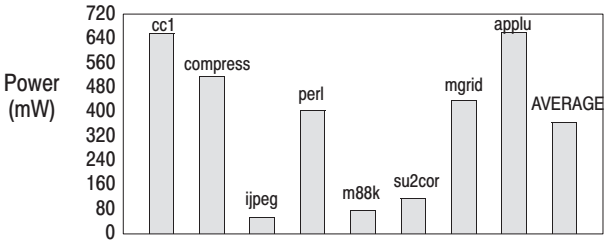


Figure 5. Power dissipations in the conventional **L2 cache** across the **SPEC95** benchmarks for Configuration I

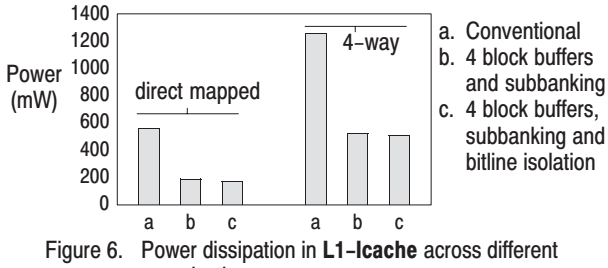


Figure 6. Power dissipation in **L1-Icache** across different organizations

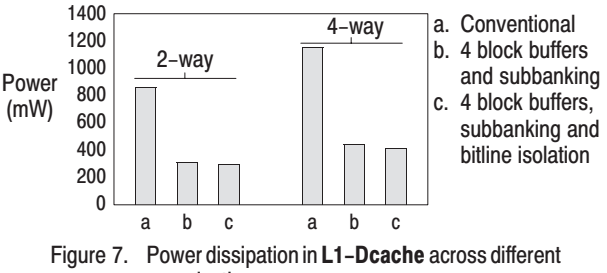


Figure 7. Power dissipation in **L1-Dcache** across different organizations

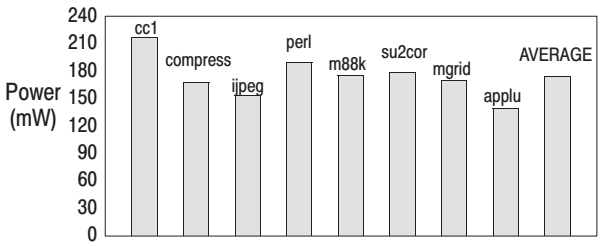


Figure 8. Power dissipations in the **L1-Icache** across the SPEC95 benchmarks for the most energy efficient Icache configuration (Configuration IV)

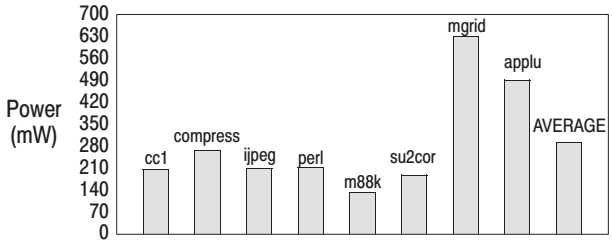


Figure 9. Power dissipations in the **L1-Dcache** across the SPEC95 benchmarks for the most energy efficient Dcache configuration (Configuration VI)

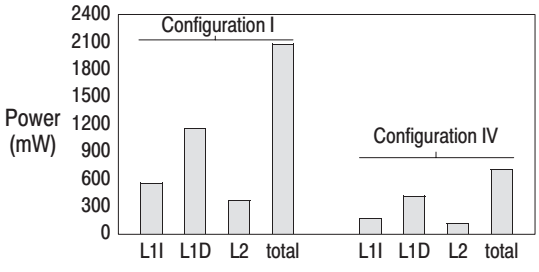


Figure 10. Power dissipations in the **on-chip caches**

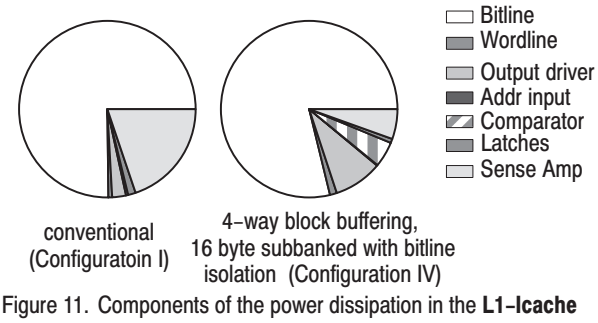


Figure 11. Components of the power dissipation in the **L1-Icache**

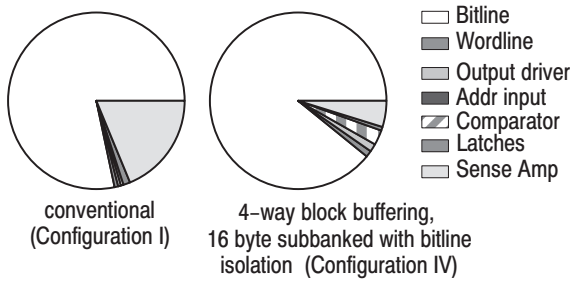


Figure 12. Components of the power dissipation in the **L1-Dcache**

Power and Performance Tradeoffs using Various Cache Configurations

Gianluca Albera [§†] and R. Iris Bahar [†]

[§] Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

[†] Brown University
Division of Engineering
Providence, RI 02912

Abstract

In this paper, we will propose several different data and instruction cache configurations and analyze their power as well as performance implications on the processor. Unlike most existing work in low power microprocessor design, we explore a high performance processor with the latest innovations for performance. Using a detailed, architectural-level simulator, we evaluate full system performance using several different power/performance sensitive cache configurations. We then use the information obtained from the simulator to calculate the energy consumption of the memory hierarchy of the system. Based on the results obtained from these simulations, we will determine the general characteristics of each cache configuration. We will also make recommendations on how best to balance power and performance tradeoffs in memory hierarchy design.

1 Introduction

In this paper we will concentrate on reducing the energy demands of an ultra high-performance processor, such as the Pentium Pro or the Alpha 21264, which uses superscalar, speculative, out-of-order execution. In particular, we will investigate architectural-level solutions that achieve a power reduction in the memory subsystem of the processor *without* compromising performance.

Prior research has been aimed at measuring and recommending optimal cache configuration for power. For instance, in [10], the authors determined that high performance caches were also the lowest power consuming caches since they reduce the traffic to the lower level of the memory system. The work by Kin [7] proposed accessing a small *filter cache* before accessing the first level cache to reduce the accesses (and energy consumption) from DL1. The idea lead to a large reduction in memory hierarchy energy consumption, but also resulted in a substantial reduction in processor performance. While this reduction in performance may be tolerable for some applications, the high-end market will not make such a sacrifice. This paper will propose memory hierarchy configurations that reduce power while retaining performance.

Reducing cache misses due to line conflicts has been shown to be effective in improving overall system performance in high-performance processors. Techniques to reduce conflicts include increasing cache associativity, use of victim caches [5], or cache bypassing with and without the aid of a buffer [4, 9, 11]. Figure 1 shows the design of the memory hierarchy when using buffers alongside the first level caches.

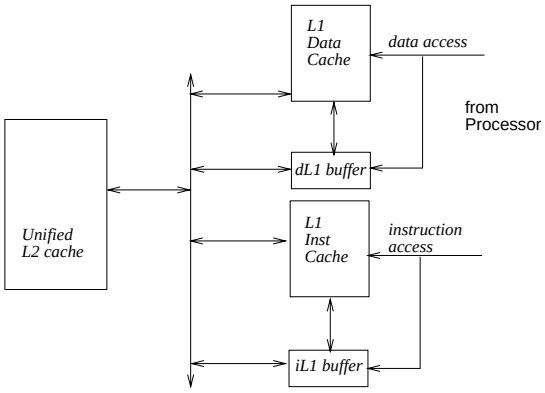


Figure 1: Memory hierarchy design using buffers alongside the L1 caches. These buffers may be used as victim caches, non-temporal buffers, or speculative buffers.

The buffer is a small cache, between 8–16 entries, located between the first level and second level caches. The buffer may be used to hold specific data (e.g. non-temporal or speculative data), or may be used for general data (e.g. "victim" data). In this paper we will analyze various uses of this buffer in terms of both power and performance. In addition, we will compare the power and performance impact of using this buffer to more traditional techniques for reducing cache conflicts such as increasing cache size and/or associativity.

2 Experimental setup

This section presents our experimental environment. First, the CPU simulator will be briefly introduced and then we will describe how we obtained data about the energy consumption of the caches. Finally we will describe each architectural design we considered in our analysis.

2.1 Full model simulator

We use an extension of the *SimpleScalar* [2] tool suite. SimpleScalar is an execution-driven simulator that uses binaries compiled to a MIPS-like target. SimpleScalar can accurately model a high-performance, dynamically-scheduled, multi-issue processor. We use an extended version of the simulator that more accurately models

all the memory hierarchy, implementing non-blocking caches and complete bus bandwidth and contention modeling [3]. Other modifications were added to handle precise modeling of cache fills.

Tables 1, 2, and 3 show the configuration of the processor modeled. Note that first level caches are on-chip, while the unified second level cache is off-chip. In addition we have a 16-entry buffer associated with each first level cache; this buffer was implemented either as a fully associative cache with LRU replacement, or as a direct mapped cache. Note that we chose a 8K first level cache configuration in order to obtain a reasonable hit/miss rate from our benchmarks [13]. In Tables 2 and 3 note that some types of resource units (e.g., the FP Mult/Div/Sqrt unit) may have different latency and occupancy values depending on the type of operation being performed by the unit.

Table 1: Machine configuration parameters.

Parameter	Configuration
L1 Icache	8KB direct; 32B line; 1 cycle lat.
L1 Dcache	8KB direct; 32B line; 1 cycle lat.
L2 Unified Cache	256KB 4-way; 64B line; 12 cycles
Memory	64 bit-wide; 20 cycles on page hit, 40 cycles on page miss
Branch Pred.	2k gshare + 2k bimodal + 2k meta
BTB	1024 entry 4-way set assoc.
Return Addr. Stack	32 entry queue
ITLB	32 entry fully assoc.
DTLB	64 entry fully assoc.

Table 2: Processor resources.

Parameter	Units
Fetch/Issue/Commit Width	4
Integer ALU	3
Integer Mult/Div	1
FP ALU	2
FP Mult/Div/Sqrt	1
DL1 Read Ports	2
DL1 Write Ports	1
Instruction Window Entries	64
Load/Store Queue Entries	16
Fetch Queue	16
Minimum Misprediction Latency	6

Table 3: Latency and occupancy of each resource.

Resource	Latency	Occupancy
Integer ALU	1	1
Integer Mult	3	1
Integer Div	20	19
FP ALU	2	1
FP Mult	4	1
FP Div	12	12
FP Sqrt	24	24
Memory Ports	1	1

Our simulations are executed on SPECint95 benchmarks; they were compiled using a re-targeted version of the GNU *gcc* compiler, with full optimization. This compiler generates 64 bit-wide instructions, but only 32 bits are used, leaving the others for future implementations; in order to model a typical actual machine, we convert these instructions to 32 bits before executing the code. Since one of our architectures was intended by the authors for floating point applications [9], we also ran a subset of SPECfp95 in this case. Since we are executing a full model on a a very detailed simulator, the benchmarks take several hours to complete; due to time constraints we feed the simulator with a small set of inputs. However we execute all programs entirely (from 80M instructions in *compress* to 550M instructions in *go*).

2.2 Power model

Energy dissipation in CMOS technology circuits is mainly due to charging and discharging gate capacitances; on every transition we dissipate $E_t = \frac{1}{2} \cdot C_{eq} \cdot V_{dd}^2$ Watts. To obtain the values for the equivalent capacitances, C_{eq}, for the components in the memory subsystem, we follow the model given by Wilton and Jouppi [12]. Their model assumes a 0.8 μm process; if a different process is used, only the transistor capacitances need to be recomputed. To obtain the number of transitions that occur on each transistor, we refer to Kamble and Ghose [6], adapting their work to our overall architecture.

An m-way set associative cache consists of three main parts: a data array, a tag array and the necessary control logic. The data array consists of S rows containing m lines. Each line contains L bytes of data and a tag T which is used to uniquely identify the data. Upon receiving a data request, the address is divided into three parts. The first part indexes one row in the cache, the second selects the bytes or words desired, and the last is compared to the entry in the tag to detect a hit or a miss. On a hit, the processor accesses the data from the first level cache. On a miss, we use a write-back, write-allocate policy. The latency of the access is directly proportional to the capacitance driven and to the length of bit and word-lines. In order to maximize speed we kept the arrays as square as possible by splitting the data and tag array vertically and/or horizontally. Sub-arrays can also be folded. We used the tool CACTI [12] to compute sub-arraying parameters for all our caches.

According to [6] we consider the main sources of power to be the following three components:E_{bit}, E_{word}, E_{output}. We are not considering the energy dissipated in the address decoders, since we found this value to be negligible compared to the other components. Similar to Kin [7], we found that the energy consumption of the decoders is about three orders of magnitude smaller than that of the other components. The energy consumption is computed as:

$$E_{cache} = E_{bit} + E_{word} + E_{output} \quad (1)$$

A brief description of each of these components follows. Further details can be found in [1].

Energy dissipated in bit-lines

E_{bit} is the energy consumption in the bit-lines; it is due to precharging lines (including driving the precharge logic) and reading or writing data. Since we assume a sub-arrayed cache, we need to precharge and discharge only the portion directly related with the address we need to read/write.

Table 4: **Baseline results:** Number of cycles, accesses and energy consumption in the base case. Energy is given in Joules

.

Test	*Cycles*	**DL1 Cache**		**IL1 Cache**		**UL2 Cache**	
		Accesses	*Eng. (Joules)*	*Accesses*	*Eng. (Joules)*	*Accesses*	*Eng. (Joules)*
compress	70 538 278	27 736 186	0.092	66 095 704	0.207	3 134 116	0.402
go	826 111 517	169 860 620	0.612	430 587 086	1.499	60 055 380	6.819
vortex	250 942 324	93 470 624	0.284	108 663 648	0.375	15 336 216	1.796
gcc	392 296 667	103 457 950	0.317	176 609 209	0.618	22 959 880	2.652
li	134 701 535	74 445 274	0.232	140 347 169	0.446	5 856 655	0.652
ijpeg	139 376 153	73 358 576	0.214	135 804 661	0.437	3 906 431	0.451
m88ksim	659 451 084	130 319 219	0.369	241 836 872	0.882	34 953 990	3.980
perl	372 034 266	90 388 059	0.293	148 817 944	0.531	24 532 245	2.794

Table 5: **Instruction cache sizes:** Percent improvement in performance and power compared to the base case. Latency = 1 cycle.

Test	**8K-2way**			**16K-direct**			**16K-2way**		
	%Cyc	*%IL1 Eng.*	*%Tot. Eng.*	*%Cyc*	*%IL1 Eng.*	*%Tot. Eng.*	*%Cyc*	*%IL1 Eng.*	*%Tot. Eng.*
compress	0.24	-18.39	-5.26	0.34	-20.31	-5.67	0.32	-58.63	-17.05
go	13.15	-20.59	8.98	21.17	-22.61	15.90	26.51	-62.84	14.04
vortex	8.30	-22.12	6.14	21.14	-23.22	18.07	27.61	-66.16	18.61
gcc	5.08	-18.95	2.08	19.29	-20.04	15.44	25.02	-58.33	14.60
li	7.20	-18.78	1.58	2.47	-20.11	-5.06	8.69	-59.13	-11.40
ijpeg	4.76	-18.02	-2.28	4.410	-19.88	-3.29	5.43	-57.51	-18.60
m88ksim	16.10	-17.24	12.89	27.33	-15.53	24.73	50.50	-55.10	42.75
perl	11.13	-18.83	7.85	19.10	-20.27	14.98	24.71	-59.82	15.13

Note that in order to minimize the power overhead introduced by buffers, in the fully associative configuration, we perform first a tag look-up and access the data array only on a hit. If timing constraints make this approach not feasible, direct mapped buffers should be considered.

Energy dissipated in word-lines

E_{word} is the energy consumption due to assertion of word-lines; once the bit-lines are all precharged we select one row, performing the read/write to the desired data.

Energy dissipated driving external buses

E_{output} is the energy used to drive external buses; this component includes both the data sent/returned and the address sent to the lower level memory on a miss request.

3 Experimental results

3.1 Base case

In this section we will describe the base case we used; all other experiments will compare to this one.

As stated before, our base case uses 8K direct mapped on-chip first level caches (i.e. DL1 for data and IL1 for instruction), with a unified 256K 4-way off-chip second level cache (UL2). Table 4 shows the execution time measured in cycles and, for each cache, the number of accesses and the energy consumption (measured in Joules). The next sections' results will show percentage decreases over the base case; thus positive numbers will mean an improvement in power or performances and negative number will mean a worsening.

First level caches are on-chip, so their energy consumption refers to the CPU level, while the off-chip second level cache energy refers to the board level. In the following sections we show what happens to the energy in the overall cache architecture (L1 plus L2), and also clarify whether variations in power belong to the CPU or to the board. As shown in Table 4 the dominant portion of energy consumption is due to the UL2, because of its bigger size and high capacitance board buses. Furthermore we see that the instruction cache demands more power than the data cache, due to a higher number of accesses.

3.2 Traditional techniques

Traditional approaches for reducing cache misses utilize bigger cache sizes and/or increased associativity. These techniques may offer good performances but present several drawbacks; first an area increase is required and second it is more likely that a larger latency will result whenever bigger caches and/or higher associativity are used.

Table 5 presents results obtained changing instruction cache size and/or associativity. Specifically, we present 8K 2-way, 16K direct mapped and 16K 2-way configurations, all of them maintaining a 1 cycle latency. Reduction in cycles, energy in IL1 and total energy are shown. The overall energy consumption (i.e. *Total Energy*) is generally reduced, due to a reduced activity in the sec-

Table 6: **Victim cache fully associative:** Percent improvement in performance and power compared to the base case.

Test	**Swapping**						**Non Swapping**					
	Data Only		*Inst. Only*		*Data & Inst.*		*Data Only*		*Inst. Only*		*Data & Inst.*	
	%Cyc.	*%Eng.*	*%Cyc.*	*%Eng.*	*%Cyc.*	*%Eng.*	*%Cyc.*	*%Eng.*	*%Cyc.*	*%Eng.*	*%Cyc.*	*%Eng.*
compress	2.30	6.79	-0.44	-1.51	2.83	5.97	2.91	6.98	-0.44	-1.50	2.77	5.65
go	3.33	10.98	4.40	3.48	7.84	14.49	4.22	12.89	6.50	4.60	10.92	18.00
vortex	3.04	12.01	5.20	4.80	7.43	12.99	3.43	12.34	5.63	4.86	9.75	17.74
gcc	1.72	6.98	3.01	2.54	4.61	8.96	2.11	7.41	4.30	3.37	6.69	11.27
li	2.88	6.62	3.06	2.04	6.26	9.11	3.82	8.96	6.11	4.90	10.69	15.00
ijpeg	4.41	11.04	1.61	-0.72	5.92	11.44	4.86	10.58	2.66	-0.24	7.54	12.16
m88ksim	0.36	3.06	9.80	9.39	10.22	12.51	0.61	4.41	21.07	18.14	21.85	23.12
perl	0.90	4.49	5.60	5.06	6.75	9.78	1.69	5.42	7.58	5.64	8.50	11.36

ond level cache (since we decrease the IL1 miss rate). However we can also see cases in which we have an increase in power, since benchmarks like *compress*, *li* and *ijpeg* already present a high hit rate in the base case.

Even if the total energy improves, we show that the on-chip power increases significantly (up to 66%) as the cache becomes bigger. As will be shown in the coming section, we observed that using buffers associated with first level caches permit us to obtain an overall energy reduction with an on-chip increase by *at most* 3%. We also observed that if the increase in size/associativity requires having a 2 cycle latency, performance improvements are no longer so dramatic and in some cases we also saw a decrease (e.g. especially for the instruction cache).

In the following sections we will presents various architectures we have analyzed. We will start with some literature examples such as the *victim cache* [5] and *non-temporal buffer* [9] and then we will present new schemes based on the use of buffers and how to combine them.

3.3 Victim cache

We consider the idea of a *victim cache* originally presented by Jouppi in [5], with small changes. The author presented the following algorithm. On a main cache miss, the victim cache is accessed; if the address hits the victim cache, the data is returned to the CPU and at the same time it is promoted to the main cache; the replaced line in the main cache is moved to the victim cache, therefore performing a "swap". If the victim cache also misses, an L2 access is performed; the incoming data will fill the main cache, and the replaced line will be moved to the victim cache. The replaced entry in the victim cache is discarded and, if dirty, written back to the second level cache.

We first made a change in the algorithm, performing a parallel look-up in the main and victim cache, as we saw that this helps performance without significant drawbacks on power. We refer to this algorithm as *victim cache swapping*, since swapping is performed on a victim cache hit.

We found that the time required by the processor to perform the swapping, due to a victim hit, was detrimental to performance, so we also tried a variation on the algorithm that doesn't require swapping (i.e. on a victim cache hit, the line is not promoted to the main cache). We refer to it as *victim cache non-swapping.*

Table 6 shows effects using a fully associative *victim cache*. They refer to the *swapping* and *non swapping* mechanisms. We present cycles and overall energy reduction for three different schemes; they use a buffer associated with the *Data* cache, the *Instruction* cache, or both of them. We observed that the combined use of buffers for both caches offers a roughly additive improvement over the single cache case. This result generally applies to all uses of buffers we tried. As stated before, we show that the *non swapping* mechanism generally outperforms the original algorithm presented by the author. In [5], the data cache of a single issue processor was considered, where a memory access occurs approximately one out of four cycles; thus the victim cache had ample time to perform the necessary swapping. The same cannot be said of a 4-way issue processor that has, on average, one data memory access per cycle. In this case the advantages obtained by swapping are often outweighed by the extra latency introduced.

It is interesting to note that increased performance and energy reduction usually go hand-in-hand, since a reduced L2 activity helps both of them.

3.4 Non-temporal buffer

This idea has been formulated by Rivers and Davidson in [9] and refers to data buffer only. They observed that in numerical applications data accesses may be divided in two categories: *scalar* accesses that present *temporal* behavior, i.e. are accessed more than once during their lifetime in the cache, and *vector* accesses that present *non-temporal* behavior, i.e. once accessed are no longer referenced. This is true, since vectors are generally accessed sequentially and often their working-set is bigger than the cache itself, so that when they need to be referenced again, they no longer reside in the cache. The idea is to use a special buffer to contain the non-temporal data and reduce conflicts in the main cache. They presented a history based algorithm that tags each line with a temporal/non-temporal bit; this information is also saved in the second level cache, requiring additionally writebacks for replaced clean blocks.

Since this algorithm was intended for numerical applications we added a subset of SPECfp95 to this set of runs. Experimental results show this technique not to be effective for integer programs; performance generally is

Table 7: **Speculative buffer:** Percent improvement in performance and power compared to the base case.

Test	Fully Associative						Direct Mapped					
	Data Only		*Inst. Only*		*Data & Inst.*		*Data Only*		*Inst. Only*		*Data & Inst.*	
	%Cyc.	*%Eng.*	*%Cyc.*	*%Eng.*	*%Cyc.*	*%Eng.*	*%Cyc.*	*%Eng.*	*%Cyc.*	*%Eng.*	*%Cyc.*	*%Eng.*
compress	1.90	5.37	-0.20	-1.27	2.40	4.79	0.88	1.79	-0.12	-1.83	0.65	-0.010
go	3.41	10.76	6.28	4.49	9.84	15.41	2.20	8.07	5.44	3.27	7.68	11.09
vortex	2.39	7.75	5.78	5.09	8.00	12.84	1.76	6.00	5.55	4.30	7.66	10.47
gcc	1.51	5.27	4.52	3.59	6.15	9.13	1.07	4.12	4.40	2.94	5.46	7.14
li	2.44	4.99	5.94	5.66	8.75	10.84	1.88	3.48	6.32	5.46	8.28	8.94
ijpeg	3.33	8.01	1.40	-0.72	4.58	7.38	2.61	5.50	1.75	-0.71	4.26	5.12
m88ksim	0.29	2.20	22.88	21.65	23.13	24.47	0.25	1.99	22.05	20.45	22.49	22.96
perl	0.66	3.25	6.60	5.65	7.38	9.15	0.79	2.99	6.26	4.79	7.10	7.98

worse by 3% to 20%, while power increases by 7% to 58%. We also observed that only specific numeric applications benefit from this algorithm; for example *swim* improves 6.8% in performance and 46% in overall energy consumption, but others like *apsi* or *hidro2d* worsen up to 7% in power and 1% in performance.

The main reason of these negative results is due to an increased writeback activity to the second level cache required by the algorithm that saves in the L2 the temporal/non-temporal information. Given a shorter latency L2 cache (e.g. on-chip L2 design), this algorithm might present better overall results.

3.5 Speculative buffer

The idea of a speculative buffer is based on previous work by the authors [1] and based on use of confidence predictors presented in Manne *et al.* [8]. In this case we mark every cache access with a confidence level obtained by examining the processor speculation state and the current branch prediction estimate. We use the main cache to accommodate misses that are most likely on the correct path (high confidence) and the *speculative buffer* for misses that have a high probability to be from a mis-speculated path (low confidence). This idea originates from a study in which the authors found that line fills coming from mis-speculated path misses have a lower probability of being further accessed. Putting them in the buffer reduces the contention misses in the main cache by reducing cache pollution.

Table 7 presents results using either a fully associative or direct mapped *speculative buffer*. Although the *victim cache* scheme with swapping gives better results when used with the data cache only, overall the results show that the *speculative buffer* scheme is better at reducing energy consumption and improving performances.

Using these buffers not only reduces energy consumption in the overall cache memory system, but it does so without significantly increasing on-chip energy consumption. For the results listed in Table 7 we found that the on-chip energy consumption in the data portion increases on average only by 0.7%, and in the instruction portion by 2.8% (these number are not shown in the table). Even more, for some programs like *go* we reduce the on-chip data cache portion up to 8%. This is in contrast to results shown in Table 5 where on-chip power increased by as much as 66%.

We also tried a variation on this algorithm deciding randomly whether to put an incoming fill, either in the main cache or in the buffer. We tried, among other cases, to put randomly 10% or 20% of the fills in the buffer. It is interesting to note that the *random* case presents, on average, good results. This demonstrates that simply adding "associativity" to the main cache, suffices to eliminate most of the contention misses (and also may indicate our benchmarks are not ideal candidates for *speculative* sorting).

3.6 Penalty buffer

This model is applied to the instruction cache buffer only. We observed that, as opposed to data cache behavior, there is not a fixed correlation between variations in instruction cache miss-rate and performance gain. This is due to the fact that some misses are more critical than others; fetch latency often may be hidden by the Fetch/Dispatch queue as well as the Instruction Window (Resources Reservation Unit - RUU in SimpleScalar terminology) making some instruction misses non-critical. However, since misses sometime present a burst behavior, these hardware structures can remain empty and all latency is detrimental for performance.

Our idea is to divide misses on a penalty basis; we monitor the state (number of valid entries) of the Dispatch Queue and RUU upon a cache miss and we mark them as *critical* (Dispatch Queue or RUU with few valid entries) or *non-critical* (Dispatch Queue or RUU with many valid entries). We place *non-critical* misses in the *penalty buffer* and *critical* ones in the main cache. In this way we preserve the instructions contained in the main cache that are presumed to be more critical.

We tried two different schemes in deciding when to bypass the main cache and place fills instead in the penalty buffer. In the first scheme we compared the number of Dispatch Queue or RUU valid entries to a fixed threshold. In the second scheme, we used a history-based method that saves in the main cache the state of the hardware resources at the moment of the fill (e.g. how many valid instructions we have in the RUU). This number will be compared to the current one at the moment of a future replacement. If the actual miss is more critical than the past one, it will fill the main cache, otherwise it will fill the *penalty* buffer.

Table 8 presents results using either a fully associative or direct mapped *penalty buffer*. We observed some good results, but when we tried different threshold val-

Table 8: **Penalty buffer:** Percent improvement in performance and power compared to the base case.

Test	**Fully Associative**				**Direct Mapped**			
	Dispatch		*RUU*		*Dispatch*		*RUU*	
	%Cycles	*%Energy*	*%Cycles*	*%Energy*	*%Cycles*	*%Energy*	*%Cycles*	*%Energy*
compress	0.02	-1.04	0.08	-1.22	0.10	-1.49	0.10	-1.65
go	6.02	4.26	6.37	4.60	4.35	2.65	5.27	3.28
vortex	6.52	5.40	6.29	4.28	6.25	4.22	4.91	2.78
gcc	4.76	3.94	4.69	3.88	4.11	3.10	4.30	3.15
li	6.25	5.70	6.35	5.88	6.11	5.04	6.12	5.17
ijpeg	2.95	1.07	2.21	-0.94	2.51	-0.48	2.62	-0.91
m88ksim	18.40	17.17	24.55	23.14	20.62	18.97	23.83	22.07
perl	7.05	5.95	6.56	5.29	6.32	4.83	5.69	4.00

ues or history schemes, we found great variability in the behavior. This variability is due to the fact that in some cases looking separately at the Dispatch Queue or the RUU doesn't give a precise model of the state of the pipeline, thus creating interference in the *penalty buffer* mechanism. Nevertheless, this scheme produces better results than the *speculative buffer* scheme. We are currently investigating a combined use of these two schemes that may prove even better.

3.7 Combining use of buffers

So far we presented two main categories of buffer implementations; the first one is based on the *victim cache* that is used to offer a "second chance" to data that, otherwise, should be moved out from L1. In the second category (all other cases), the algorithm decides what to put in the buffer, before data has the opportunity to be written to the L1 cache. These are not mutually exclusive, so we tried to combine them, i.e. the buffer is at the same time used to store specific kinds of data and to accommodate L1 cache victims. Due to space limitations, we will not show the results, but it is important to point out that this mechanism is generally beneficial and gives improvements over the non-combined case. However most of the gain is still due to the original algorithm and not to the victim scheme.

4 Conclusions and future work

In this paper we presented tradeoffs between power and performance in cache architectures, using conventional as well as innovative techniques; we showed that adding small buffers can offer an overall energy reduction without decreasing the performance, and in most cases improving it. These small buffers suffice in reducing the overall cache energy consumption, without a notable increase in on-chip power that traditional techniques present. Furthermore use of a buffer is not mutually exclusive to increasing the cache size/associativity, offering the possibility of a combined effect. We are currently investigating further improvements in combined techniques as well as replacement policies. In the results we showed LRU policy was adopted; we are now studying policies that favor replacement of clean blocks, thereby reducing writeback activity.

Acknowledgments

We wish to thank Bobbie Manne for her invaluable help and support throughout this work. We also wish to thank Doug Burger for providing us with the framework for the SimpleScalar memory model.

References

[1] I. Bahar, G. Albera and S. Manne, "Using Confidence to Reduce Energy Consumption in High-Performance Microprocessors," to appear in *International Symposium on Low Power Electronics and Design (ISLEPD)*, 1998

[2] D. Burger, and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report TR#1342, University of Wisconsin, June 1997.

[3] D. Burger, and T. M. Austin, "SimpleScalar Tutorial ," presented at *30th International Symposium on Microarchitecture*, Research Triangle Park, NC, December, 1997.

[4] T. L. Johnson, and W. W. Hwu, "Run-time Adaptive Cache Hierarchy Management via Reference Analysis," *ISCA-97: ACM/IEEE International Symposium on Computer Architecture*, pp. 315–326, Denver, CO, June 1997.

[5] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *ISCA-17: ACM/IEEE International Symposium on Computer Architecture*, pp. 364–373, May 1990.

[6] M. B. Kamble and K. Ghose, "Analytical Energy Dissipation Models for Low Power Caches," *ACM/IEEE International Symposium on Low-Power Electronics and Design*, August, 1997.

[7] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," *MICRO-97: ACM/IEEE International Symposium on Microarchitecture*, pp. 184–193, Research Triangle Park, NC, December 1997.

[8] S. Manne, D. Grunwald, A. Klauser, "Pipeline Gating: Speculation Control for Energy Reduction," to appear in *ISCA-25: ACM/IEEE International Symposium on Computer Architecture*, June 1998.

[9] J. A. Rivers, and E. S. Davidson, "Reducing Conflicts in Direct-Mapped Caches with a Temporality-Based Design," *International Conference on Parallel Processing*, pp. 154-163, August 1996.

[10] C. Su, and A. Despain, "Cache Design Tradeoffs for Power and Performance Optimization: A Case Study," *ACM/IEEE International Symposium on Low-Power Design*, pp. 63–68, Dana Point, CA, 1995.

[11] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun, "Managing Data Caches using Selective Cache Line Replacement," *Journal of Parallel Programming*, Vol. 25, No. 3, pp. 213–242, June 1997.

[12] S. J. E. Wilton, and N. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches," Digital WRL Research Report 93/5, July 1994.

[13] Jeffrey Gee, Mark Hill, Dinoisions Pnevmatikatos, Alan J. Smith, "Cache Performance of the SPEC Benchmark Suite," IEEE Micro, Vol. 13, Number 4, pp. 17-27 (August 1993)

A new scheme for I-Cache energy reduction in High-Performance Processors *

Nikolaos Bellas, Ibrahim Hajj and Constantine Polychronopoulos
Coordinated Sciences Laboratory, and
Electrical and Computer Engineering Department
University of Illinois at Urbana-Champaign, Urbana, IL 61801

Abstract

Prompted by demands in portability and low cost packaging, the microprocessor industry has started viewing power, along with area and performance, as a decisive design factor in today's microprocessors. To that effect, a number of research efforts have been devoted to architectural modifications that target power or energy minimization. Most of these efforts, however, involve a degradation in processor performance, and are, thus, deemed applicable only for the embedded, low-end market.

In this paper, we propose the addition of an extra, small cache between the I-Cache and the CPU, that serves to reduce the effective energy dissipated per memory access. In our scheme, the compiler generates code that exploits the new memory hierarchy and reduces the likelihood of a miss in the extra cache. We show that this is an attractive solution for the high-end processor market, since the performance degradation is minimal. We describe the hardware and compiler modifications needed to efficiently implement the new memory hierarchy, and we give the performance and energy results for most of the SPEC95 benchmarks. The extra cache, dubbed L-Cache from now on, is placed between the CPU and the I-Cache. The D-Cache subsystem remains as is.

1 Introduction

In the latest generations of microprocessors, an increasing number of architecture features have been exposed to the compiler to enhance performance. The advantage of this cooperation is that the compiler can generate code that exploits the characteristics of the machine and avoids expensive stalls. We believe that such schemes can also be applied for power/energy optimization by exposing the memory hierarchy features in the compiler.

The filter cache [1] tackles the problem of large energy consumption of the L1 caches by adding a small, and, thus, more energy efficient cache between the CPU and the L1 caches. Provided that the working set of the program is relatively small, and the data reuse large, this "mini" cache can provide the data or instructions of the program and effectively shut down the L1 caches for long periods during program execution. The penalty to be paid is the increased miss rates, and, hence, longer average memory access time. Although this might be acceptable for embedded systems for multimedia or mobile applications, it is out of the question for high performance processors. The filter cache delivers an impressive energy reduction of 58% for a 256-byte, direct mapped filter cache, while reducing performance by 21% for a set of multimedia benchmarks. Our approach has a very small performance degradation with respect to the original scheme without the filter cache, and smaller, but still very large, energy gains.

We can alleviate the performance degradation by having the compiler selecting statically, i.e. during compile time, the parts of the code that are to be placed in the extra cache (the L-Cache), and restructuring the code so that it fully exploits the new hierarchy. The CPU will then access the L-Cache only when instructions from the selected part of the program are to be fetched, and it will bypass it otherwise. Naturally, we want the most frequently executed code to be selected by the compiler for placement in the L-Cache, since this is where most of the energy gains will come from.

The approach advocated in our scheme relies on the use of profile data from previous runs to select the best instructions to be cached. The unit of allocation is the basic block, i.e. an instruction is placed in the L-Cache only if it belongs to a selected basic block. After selection, the compiler lays out the target program so that the se-

*This work was supported by Intel Corp., Santa Clara, CA

lected blocks are placed contiguously before the non-placed ones. The main effort of the compiler focuses on placing the selected basic blocks in positions so that two blocks that need to be in the L-Cache at the same time, do not map in the same L-Cache location.

The organization of the paper is as follows: Section 2 briefly describes how the compiler selects instructions to be placed in the extra cache and what code modifications are necessary for this scheme to be carried out effectively. Section 3 refers to the hardware implementation details, and the energy estimation phase. Section 4 presents and discusses experimental results on SPEC95 benchmarks, and the paper is concluded with Section 5.

2 Compiler Enhancements

In this work we describe a possible static implementation, which relies on the capability of the compiler to detect "good" candidate basic blocks for caching, i.e. blocks whose instructions can be stored in the L-Cache. The main stage in our compiler enhancement is *block placement* in which the compiler selects, and then places the selected basic blocks so that the number of blocks that are placed at the same time in the extra cache is maximized. To that effect, it avoids placing two blocks that have been selected to reside at the same time in the L-Cache in the same L-Cache location.

2.1 Block Placement for Maximum L-Cache Utilization

The main step of our algorithm is to position the code in such a way as to maximize the number of basic blocks that are cached in the extra cache. In order to do that, we need to place these blocks contiguously in memory so that they do not overlap. Consider the following code:

```
do 100 i=1, n
  B1;      # basic block
  if (error) then
     error handling;
  B2;      # basic block
100 continue
```

When the code is compiled and mapped in the address space, the basic blocks B1 and B2 will be separated by the code for the if-statement in the final layout. If the L-Cache size is smaller than the sum of the sizes of B1, B2 and the if-statement, but larger than the sum of the sizes of B1 and B2, the blocks B1 and B2 will overlap when stored in the L-Cache. Therefore, we need to place B1 and B2 one after the other and leave the if-statement at the end.

This is usually the case in loops. Blocks that are executed for every iteration are intermingled with blocks that are seldom executed. We identify such cases and move the infrequently executed code away so that the normal flow of control is in a straight-line sequence. This entails the insertion of extra branch and jump instructions to retain the original semantics of the code.

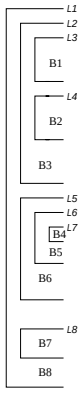


Figure 1: Nesting example

We will delineate our approach in the following paragraphs without giving many details, because of lack of space. Our compiler modification takes as input the generated object code and the profile data of the program, and outputs a new, equivalent object code with a different address mapping for some of the basic blocks.

Our strategy uses the *Control Flow Graph (CFG)* for each function of the program and then computes the nesting of each basic block [2]. For example, in Fig. 1, the nesting of basic block B_3 are the loops L_1, L_2.

The basic ideas behind the compiler modification are the following: First, the most frequently executed basic blocks of a function/procedure should be placed in the global memory space the one after the other so that their locality is enhanced. These blocks are detected during profiling. Second, the structure of the CFG is also important in minimizing the miss rate of the L-Cache. For example, B_3 should not overlap with B_1 and B_2 in the L-Cache. If that was the case, B_3 would miss in every iteration of the loop L_2 and the possible gains of its placement in the L-Cache would be nullified. The same goes for B_6 with respect to B_4 and B_5. On the other hand, B_1 and B_2 can overlap since B_2 is only executed when B_1 finishes. Our algorithm uses the CFG and the execution profile of each basic block in the program to create the new object code.

There are six reasons that hinder a basic block from being selected for placement in the L-Cache:

- It belongs to a library and not to a user function. We follow the convention that only user functions are candidates for placement, since they have the tightest loops.

- The algorithm finds that the basic block was too large to fit in the L-Cache. This can be either because the size of the block is larger than the L-Cache size, or because it cannot fit at the same time with other, more important, basic blocks.
- Its execution frequency is smaller than a threshold, and is thus deemed unimportant.
- It is not nested in a loop. There is no point in placing such a basic block in the L-Cache since it will be executed only once for each invocation of its function.
- Even if its execution is large, its *execution density* might be small. For example, a basic block that is located in a function which is invoked a lot of times might have a large execution frequency, but it might only be executed few times for every function invocation. We define the execution density of a basic block as the ratio of the number of times it is executed to the number of times that the function in which it belongs, is invoked.
- Finally, a very small basic block is not placed in the L-Cache even if it satisfies all the above requirements. The extra branch instructions that might be needed to link it to its successor basic blocks will be an important overhead in this case.

The basic blocks are laid out in the memory address space so that all the selected basic blocks are placed contiguously before the non-selected ones. This arrangement greatly simplifies the hardware of the L-Cache as we will see in the next section. Branches are placed at the end of the blocks, if needed, to sustain the functionality of the code.

The user can trade off energy savings with delay increase by adjusting the thresholds as these were discussed in this section. For example, a smaller size threshold will probably lead to larger energy savings, and larger delay as well. As an extreme, the user can also completely disable the L-Cache, by forcing the compiler (through the user given thresholds) to generate the original code, or, on the other extreme, can emulate a filter cache scheme by selecting every basic block for placement in the L-Cache. The quality of the generated code can be determined by the user during compile time. Therefore, individual applications can choose from a range of caching policies.

3 Hardware modifications and Power Estimation

In addition to the compiler enhancement, our scheme needs extra hardware for the implementation of the L-Cache scheme. The extra hardware is shown in Fig. 2.

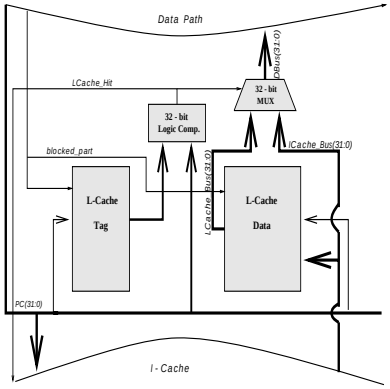


Figure 2: LCache organization

The organization of the L-Cache itself is depicted in Fig. 2. It needs the data and the tag part, an extra comparator for the tag comparison, and a 32-bit multiplexer which drives the data from the L-Cache or the I-Cache to the data path pipeline.

The functionality of the L-Cache is as follows: The PC is presented to the L-Cache tag at the beginning of the clock cycle. The L-Cache will only get activated if the *"blocked_part"* signal is on. This signal is generated by the IF unit, and its meaning is explained in the following paragraphs. In that case, the comparator checks for a match, and if it finds one, it instructs the multiplexer to drive the contents of the L-Cache in the data path. The I-Cache is disabled for this clock cycle since the signal *"blocked_part"* is on.

In case of a L-Cache miss (*"LCache_Hit"* is off), the I-Cache is activated in the next clock cycle and provides the data. The I-Cache is accessed if the L-Cache misses, whereas the L-Cache is accessed only when *"blocked_part"* = on. If *"blocked_part"* = off, the I-Cache controller activates the I-Cache without waiting for the *"LCache_Hit"* signal. The two caches are always accessed sequentially and not in parallel.

We extend the ISA, and we add an instruction called *"alloc"* which has a J-type MIPS format. It is inserted by our tool and contains the address of the first non-placed block of the function. There is one such instruction for every function of the program and the address of *"alloc"* is stored when a function is entered. During the execution of the code in the function, if the PC has a value less than that address, the *"blocked_part"* signal is set, else this signal will be set to off. This way, the machine can figure out which portion of the code executes with only an extra comparison.

We have developed our cache energy models based on [3]. This is a transistor level model which uses the run-time characteristics of the cache to estimate the energy dissipation of its main components [4]. A 0.8um technology with 3.3 Volts power supply is assumed. The cache energy is a

Experiments	*Frequency Thres.*		*Size Thres.*		*Exec. Density Thres.*	
	FP	INT	FP	INT	FP	INT
Aggressive (a)	0.01%	0.01%	5	5	5	5
Less Aggressive (b)	0.5%	0.5%	10	5	10	5
Moderate (c)	1%	1%	20	5	20	5

Table 1: User given thresholds in the L-Cache experiments

function of the cache complexity (cache size, block size, and associativity), its internal organization (banking), and the run time statistics (mumber of accesses, hits, misses, average number of bits read, input switching probabilities). The model is used for both the I-Cache and the L-Cache/filter cache.

4 Experimental Evaluation

We evaluated the effectiveness of our software and hardware enhancements by examining the energy savings on a set of SPEC95 benchmarks. The base case is a 16KB, direct-mapped I-Cache, with a block size of 32 bytes. We compared our scheme against the filter cache as well as the base case. We considered two sizes for the filter and the L-Cache: 256 and 512 bytes. The line size was always 4 bytes for the L-Cache and varied from 8 to 32 bytes for the filter cache. Both caches are direct-mapped. The L-Cache does not need to have a larger line size since its hit rate is almost always near 100%. A larger line size does not alter significantly the hit rate of the L-Cache, but affects negatively its energy profile. There is one cycle penalty in case of a miss in the L-Cache or the filter cache, and a 4 cycle penalty in case of a miss in the I-Cache since then, the data should be taken from the L2 cache. The L2 cache is off-chip and its energy dissipation is not modelled. The I-Cache is banked to optimize its cycle time [3].

We have developed an in-house, cache simulator to gather the statistics of the new memory hierarchy. We use the SpeedShop [5] set of tools from SGI to gather the profile data and generate the dynamic instruction traces that were fed into our simulator. The MIPS compiler is used for compilation and code optimization. All the necessary run time data were gathered for the computation of the energy dissipation of the cache subsystem under the different scenaria.

We experimented with three different scenaria for the user-given compiler thresholds (Table 1). The more aggressive scenario delivers larger energy gains at the expense of larger performance overhead. A frequency threshold of 0.01%, for example, will force the tool to mark for placement only basic blocks that have an execution time of at least 0.01% of the total execution time of the program. A size threshold of 10 will force the tool to mark only the basic blocks that have at least 10 instructions, and so on.

Tables 2 and 3, show the normalized energy dissipation of all the simulated cache subsystems with respect to the original I-Cache scheme. For the SPECfp95 benchmarks, the 512 byte L-Cache is almost as succesful as the filter cache in providing instructions to the data path and, thus, in saving energy by disabling the larger I-Cache. The most energy-efficient filter cache is the one with 8B or 16B line size. A filter cache with a 32B line size has larger hit rate, but larger energy dissipation per access as well. An L-Cache of 256 bytes is usually too small for any substantial energy savings.

Tables 4 and 5 show the normalized performance overhead for the various schemes, with respect to the original organization. Since we are only interested in the impact of the new memory organization on the performance overhead, we assume that there are no other stalls in the machine other than the ones created when an instruction is accessed.

The most important advantage of the L-Cache is its small performance overhead, which is vital for high performance machines. Performance overhead is due to the (very small) miss rate in the L-Cache, and the extra jump and branch instructions that are inserted by the compiler to the code. The performance of 512-byte L-Cache (under an aggressive scenario) is comparable to the performance of a 512-byte filter cache with a block size of 32 bytes. Such a filter cache, however, has worse energy gains than the L-Cache, and, in addition, entails a large 32-bytes wide bus to connect it to the I-Cache.

Most integer benchmarks do not have a large number of basic blocks that can be cached in the L-Cache. They are also insensitive to the cache size variation, which is to be expected since the basic blocks of integer programs are generally small. Most of the basic blocks of the SPECint95 benchmarks are not nested, or, they have a small execution density. Hence, they cannot be included in the L-Cache. From a performance point of view, however, a filter cache in a processor that runs an integer code, scores poorly. An L-Cache is preferable here if performance degradations cannot be tolerated.

5 Conclusions

We believe that, since performance is the most important objective of today's high-end microprocessors, no energy reduction technique will be acceptable, unless it only marginally affects the execution time, or its overhead can be hidden by other compiler/architectural techniques. If this is the case, even a moderate energy reduction will be welcome.

This paper presents a new, modified version of the filter cache in which the compiler and the extra hardware cooperate to decrease energy consumption. The compiler can select only the most important parts of the code to be placed in the extra cache, and can direct the hardware to probe the extra cache only when this code is to be fetched to the data path. The method is adaptive, since the user can aggressively pursue energy reductions to the expense of performance, or vice versa, by providing different compilation options.

References

[1] Johnson Kin, Munish Gupta and William Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," in *IEEE International Symposium on Microarchitecture*, pp. 184–193, Dec. 1997.

[2] A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques and Tools*. Addison–Wesley, 1986.

[3] S. Wilson and N. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches," DEC WRL Technical Report 93/5, July 1994.

[4] N. Bellas, *PhD Thesis in progress*. 1998.

[5] *SpeedShop User's Guide*. Silicon Graphics Inc., 1996.

Benchmark	256 bytes extra cache					
	L-Cache			Filter Cache		
	(a)	(b)	(c)	8B	16B	32B
tomcatv	0.686	0.705	0.739	0.551	0.497	0.610
swim	0.242	0.244	0.262	0.243	0.329	0.516
su2cor	0.701	0.734	0.827	0.537	0.485	0.610
hydro2d	0.418	0.438	0.493	0.397	0.413	0.566
mgrid	0.943	0.980	0.986	0.600	0.530	0.638
applu	0.736	0.755	0.840	0.541	0.490	0.610
turb3d	0.618	0.622	0.839	0.426	0.430	0.578
apsi	0.758	0.867	0.957	0.554	0.498	0.613
wave5	0.768	0.822	0.822	0.576	0.517	0.557
FP aver.	0.652	0.685	0.752	0.492	0.465	0.589
go	0.948	0.955	0.955	0.612	0.545	0.660
m88ksim	0.818	0.822	0.856	0.617	0.566	0.663
compress95	0.890	0.897	0.897	0.569	0.523	0.630
li	0.975	0.978	0.976	0.651	0.586	0.708
INT aver.	0.908	0.913	0.921	0.612	0.555	0.665

Table 2: Normalized energy relative to the base machine for 256-byte

Benchmark	512 bytes extra cache					
	L-Cache			Filter Cache		
	(a)	(b)	(c)	8B	16B	32B
tomcatv	0.692	0.692	0.745	0.306	0.369	0.539
swim	0.259	0.261	0.279	0.236	0.325	0.518
su2cor	0.295	0.308	0.367	0.272	0.343	0.527
hydro2d	0.263	0.259	0.299	0.241	0.326	0.517
mgrid	0.225	0.245	0.279	0.249	0.330	0.519
applu	0.666	0.689	0.787	0.333	0.446	0.585
turb3d	0.431	0.432	0.696	0.350	0.387	0.554
apsi	0.594	0.716	0.856	0.442	0.436	0.579
wave5	0.669	0.723	0.725	0.448	0.447	0.493
FP aver.	0.455	0.481	0.559	0.320	0.379	0.537
go	0.945	0.955	0.955	0.562	0.511	0.632
m88ksim	0.816	0.822	0.856	0.595	0.547	0.658
compress95	0.893	0.900	0.900	0.400	0.425	0.583
li	0.975	0.978	0.976	0.510	0.491	0.642
INT aver.	0.907	0.914	0.922	0.517	0.494	0.629

Table 3: Normalized energy relative to the base machine for 512-byte extra cache

Benchmark	256 bytes extra cache					
	L-Cache			Filter Cache		
	(a)	(b)	(c)	8B	16B	32B
tomcatv	1.002	1.001	1.000	1.322	1.168	1.087
swim	1.000	1.000	1.000	1.027	1.016	1.010
su2cor	1.001	1.001	1.000	1.308	1.157	1.087
hydro2d	1.038	1.032	1.023	1.174	1.091	1.051
mgrid	1.009	1.004	1.003	1.367	1.198	1.110
applu	1.080	1.052	1.032	1.314	1.165	1.089
turb3d	1.014	1.014	1.003	1.203	1.108	1.062
apsi	1.022	0.998	0.993	1.318	1.168	1.091
wave5	1.040	1.038	1.038	1.346	1.187	1.205
FP aver.	1.023	1.016	1.010	1.264	1.140	1.088
go	1.016	1.011	1.011	1.349	1.199	1.128
m88ksim	1.023	1.024	1.023	1.375	1.227	1.131
compress95	1.023	1.021	1.021	1.340	1.193	1.105
li	0.997	0.998	0.998	1.407	1.244	1.167
INT aver.	1.015	1.014	1.013	1.368	1.216	1.133

Table 4: Normalized delay relative to the base machine for 256-byte extra cache

Benchmark	512 bytes extra cache					
	L-Cache			Filter Cache		
	(a)	(b)	(c)	8B	16B	32B
tomcatv	1.000	1.000	1.000	1.083	1.049	1.025
swim	1.002	1.000	1.000	1.017	1.010	1.007
su2cor	1.010	1.003	1.001	1.050	1.026	1.015
hydro2d	1.052	1.030	1.026	1.021	1.011	1.007
mgrid	1.009	1.004	1.004	1.028	1.014	1.008
applu	1.240	1.137	1.100	1.236	1.122	1.064
turb3d	1.037	1.033	1.018	1.126	1.067	1.038
apsi	1.040	1.012	0.987	1.211	1.111	1.059
wave5	1.027	1.038	1.038	1.220	1.122	1.149
FP aver.	1.046	1.029	1.019	1.110	1.059	1.041
go	1.019	1.011	1.011	1.301	1.170	1.103
m88ksim	1.023	1.024	1.023	1.350	1.207	1.122
compress95	1.023	1.021	1.021	1.174	1.101	1.062
li	0.997	0.998	0.998	1.272	1.159	1.110
INT aver.	1.015	1.014	1.013	1.274	1.159	1.099

Table 5: Normalized delay relative to the base machine for 512-byte extra cache

Low-Power Design of Page-Based Intelligent Memory

Mark Oskin, Frederic T. Chong, Aamir Farooqui, Timothy Sherwood, and Justin Hensley
University of California at Davis

Abstract

Advances in DRAM technology have led many researchers to integrate computational logic on DRAM chips to improve performance and reduce power dissipated across chip boundaries. The density, packaging, and storage characteristics of these intelligent memory chips, however, present new challenges in power management.

We introduce *Active Pages*, a promising architecture for intelligent memory based upon pages of data and simple functions associated with that data [OCS98]. We evaluate the power consumption of three design alternatives for supporting Active-Page functions in DRAM: reconfigurable logic, a simple processing element, and a hybrid combination of reconfigurable logic and a processing element. Additionally, we discuss operating system techniques to manage power consumption by limiting the number of Active Pages computing simultaneously on a chip.

1 Introduction

As processor performance grows, both memory bandwidth and power consumption become serious issues. At the same time, commodity DRAM densities are increasing dramatically. Many researchers have proposed integrating computation with DRAM to increase memory bandwidth and decrease power consumption. The operating and packaging characteristics of DRAM chips, however, present new constraints on the power consumption of these systems.

We introduce *Active Pages*, a promising architecture for intelligent memory with substantial performance benefits for data-intensive applications [OCS98]. An Active Page consists of a page of data and a set of associated functions that operate on that data.

To use Active Pages, computation for an application must be divided, or *partitioned*, between processor and memory. For example, we use Active-Page functions to gather operands for a sparse-matrix multiply and pass those operands on to the processor for multiplication. To perform such a computation, the matrix data and gathering functions must first be loaded into a memory system that supports Active Pages. The processor then, through a series of memory-mapped writes, starts the gather functions in the memory system. As the operands are gathered, the processor reads them from user-defined output areas in each page, multiplies them, and writes the results back to the array datastructures in memory. Simulation results show up to a 1000X speedup on such applications running on systems that use Active-Page memory over conventional memory.

This paper focuses upon ongoing work which evaluates the power consumption of Active-Page implementations. First, we describe three alternative designs based upon reconfigurable logic, small processors, and a combination of the two. Then we present some power models and estimates for these designs. We also discuss software techniques for reducing power consumption. Finally, we conclude with some future directions for this work.

Acknowledgments: Thanks to Venkatesh Akella for his helpful comments. This work is supported in part by an NSF CAREER award to Fred Chong, by Altera, and by grants from the UC Davis Academic Senate. More info at http://arch.cs.ucdavis.edu/RAD

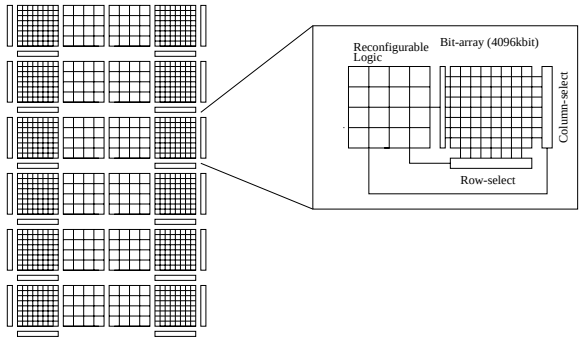


Figure 1: The RADram System

2 Design Alternatives

In this section we briefly present three architectures for Active Page memory systems. These architectures consist of RADram, or reconfigurable logic in DRAM [OCS98], an architecture consisting of only a small RISC or MISC [Jon98] like processing element near each page of data, and a hybrid architecture which provides a small processing element with reconfigurable logic processing capability.

Currently, within all of our designs we adopt a *processor-mediated* approach to inter-page communication which assumes infrequent communication. When an Active-Page function reaches a memory reference that can not be satisfied by its local page, it blocks and raises a processor interrupt. The processor satisfies the request by reading and writing to the appropriate pages. The processor-mediated approach reduces power by not requiring extensive inter-page communication networks, and global clocking.

2.1 Reconfigurable Logic

Our initial implementation of Active Pages focused upon the integration of DRAM and reconfigurable logic. For gigabit DRAMs, a reasonable sub-array size to minimize power and latency is 512Kbytes [I+97]. The RADram (Reconfigurable Architecture DRAM) system, shown in Figure 1, associates 256 LEs (Logic Elements, a standard block of logic in FPGAs which is based upon a 4-element Look Up Table or 4-LUT) to each of these sub-arrays. This allows efficient support for Active-Page sizes which are multiples of 512Kbytes.

Each LE requires approximately 1k transistors of area on a logic chip. The Semiconductor Industry Association (SIA) roadmap [Sem97] projects mass production of 1-gigabit DRAM

chips by the year 2001. If we devote half of the area of such a chip to logic, we expect the DRAM process to support approximately 32M transistors, which is enough to provide 256 LEs to each 512Kbytes sub-array of the remaining 0.5-gigabits of memory on the chip. DeHon [DeH96] gives several estimates of FPGA area, and existing prototype merged DRAM/reconfigurable logic designs demonstrate this to be a realistic design partitioning [M+97].

2.2 Processing Elements

An alternative architecture for Active Page implementations is that of a small RISC or MISC [Jon98] type processing element used to execute user level process functions in memory. Several low-power processing cores currently exist and Active Page designs can leverage the existing and future work in this area. However, a processing engine designed for Active Pages must satisfy two constraints: power and size. While high-performance low-power designs exist, such as the StrongARM [San96], their size makes them prohibitive for use in Active Page memory. For this study we assume a slightly modified Toshiba TLCS-R3900 series [NTM+95] processor. The current processor contains a 4k direct mapped instruction cache. Such a cache is not needed for the limited number of instructions an Active Page memory function contains. The calculations in this paper assume a 1k instruction cache. The remainder of the TLCS-R3900 processing engine is relatively standard, with a 1k data cache, 32 bit data-path, 32 general purpose registers and MIPS-like instruction set.

2.3 Hybrids

An alternative architecture from RADram or a small processing element, is a hybrid, which mixes a RISC-like processor with reconfigurable logic. Such an architecture is presented in [HW97]. Here we extend this notion by simplifying the processing element core, and shrinking the reconfigurable logic array. The hybrid architecture would consist of a simplified processing element described above in Section 2.2 and a smaller reconfigurable logic array than that presented for RADram in Section 2.1. The configurable logic array would be on the order of 128 LEs in size.

3 Power Consumption

In order to analyze the power consumption of the three alternative Active Page implementations described in Section 2, an analysis of each was performed based on analytical models and previously published results. In this section we present analytical models for power consumption for DRAM sub-arrays and reconfigurable logic arrays. Furthermore, we present estimations for the power consumed by an Active Page memory which utilizes off-the-shelf MIPS-like processing cores as page based functional units. Finally we estimate the power of a combined processor / reconfigurable logic array design. Throughout these analyses the target has been a 1 gigabit size DRAM array. Each design compromises a certain percentage of DRAM storage for functional logic. Each design has its own size and here we also estimate the relative percentage of logic to DRAM and thus the overall size of the resulting memory chip. While these designs are based on a 1 gigabit size DRAM, it should be kept in mind that the actual number of pages will most likely be a power of two, and thus either an increased or decreased chip size will ultimately be necessary for non-power-of-two size designs.

3.1 DRAM model for Power Analysis

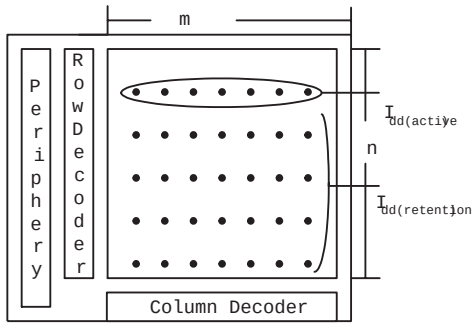


Figure 2: DRAM Model used in the analysis.

In this section we present a DRAM model for power estimation. Using this model we estimate the power consumption of a sub-array size used for gigabit size DRAMs. A DRAM block of m columns and n rows can be represented as shown in Figure 2[I+95]. This figure shows the current flowing in the DRAM block. There are two major sources of power consumption in DRAMs:

1. Active current, which flows during the read operation.
2. Data retention current, which flows during re-charging the DRAM cells.

The active current is expressed by:

$$I_{dd} = (m \cdot C_d \cdot V_w + C_{pt} \cdot V_{int}) \cdot f + I_{dcp} \qquad (1)$$

where C_d is the data line capacitance, V_w is the voltage swing at a particular data line, C_{pt} is the total capacitance of the CMOS logic and driving circuitry in the periphery, V_{int} is the internal supply voltage, f is the operating frequency, and I_{dcp} is the DC static current.

The data retention current is given by:

$$I_{dd} = (m \cdot C_d \cdot V_w + C_{pt} \cdot V_{int}) \cdot (n/t_{ref}) + I_{dcp} \qquad (2)$$

where t_{ref} is the refresh time of cells in retention mode.

C_d	0.2pf
V_w	0.5V
m	4096
n	1024
f	20Mhz
t_{ref}	20ms
V_{dd}	1.65V

Table 1: DRAM parameters for 512Kbyte block sub-arrays.

Based on Equations 1 and 2, and technological parameters summarized in Table 1, it is possible to estimate power consumption for a single 512 kilobyte page of memory. Using

this base power per-page value we will extrapolate the power consumption of various Active Page implementations.

With mixed logic and memory designs, the effects of the C_{pt} term are negligible compared to the power consumption of the attached logic, and thus the term is assumed to be zero. Furthermore, at 20Mhz operation the DC static current I_{dcp} is negligible. Thus, the active current required to access a word in memory is given by:

$$I_{active} = (4096 \cdot 0.2pf \cdot 0.5V) \cdot (20 \times 10^6) = 8.1mA \quad (3)$$

The current due to data retention is given by:

$$I_{retention} = (4096 \cdot 0.2pf \cdot 0.5V) \cdot (1024/20ms) = 20uA \quad (4)$$

Thus, for a 1.65V power supply, the total power dissipated by a 512Kbyte DRAM sub-array is:

$$P_{total} = P_{active} + P_{retention} \quad (5)$$

$$P_{total} = (I_{active} + I_{retention}) \cdot V_{dd} \quad (6)$$

$$P_{total} = (8.1mA + 20uA) \cdot 1.65V = 13.4mW \quad (7)$$

The estimated power per 512Kbyte sub-array is slightly below existing DRAM power consumption[Tos98]. This is due to the reduced supply and swing voltages used in the calculation, and is expected in future DRAM memory designs. Using this calculation we can deduce that a storage-only gigabit size DRAM chip will consume $13.4mW \cdot 256 = 3.4W$ of power. The 1 gigabit DRAM power consumption, while greater than power consumed by existing DRAM designs, is well below the maximum power ratings predicted by the SIA Technology Roadmap [Sem97] for high-performance package technology.

3.2 Power for Reconfigurable Logic

In order to estimate the power consumption of the RADram architecture, we use the power estimation guide provided by Altera [Alt96] for the FLEX architecture of devices. We model power for all devices in an operational state. It is assumed that appropriate software control by the Operating System can be used to shutdown portions of the Active Page memory device during reconfiguration. The Altera power estimation guide predicts power for operational devices as:

$$Power = P_{int} + P_{io} \quad (8)$$

$$P_{int} = (I_{CCStandby} + I_{CCActive}) \times V_{dd} \quad (9)$$

$$I_{CCStandby} = I_{CC0} \quad (10)$$

$$I_{CCActive} = (K \cdot f_{max} \cdot N \cdot toggle_{avg})uA \quad (11)$$

where P_{int} is the power consumed internally by the FPGA, P_{io} is the power consumed driving external loads, $I_{CCStandby}$ is the current drain while idle, $I_{CCActive}$ is the current drain while active, K is a technology parameter which is dependent on the FPGA design, f_{max} is the operating frequency in megahertz, N is the number of logic elements, and $toggle_{avg}$ is the average percent of cells toggling at each clock.

For estimation purposes, we assume P_{io} to be negligible, and the standby current is equal to a constant $I_{CC0} = 0.5mA$. Furthermore, current technology has K values between 32 and 95. We assume a conservative view of future technology, with a K of 30. Using the RADram configuration presented in Section 2.1, we let N equal 128 and the frequency of operation equal 100Mhz. Furthermore, we assume an average toggling percentage of 12.5 percent. Thus the active current is given by:

$$I_{CCActive} = 30 \cdot 100Mhz \cdot 256LEs \cdot 0.125 = 96mA \quad (12)$$

Therefore, power consumption of the reconfigurable logic for a single page in the RADram architecture is:

$$P = (.5mA + 96mA) \cdot 1.65V = 159mW \quad (13)$$

The RADram architecture trades DRAM memory storage for reconfigurable logic. This dedicates fifty percent of the chip space to DRAM and fifty percent to logic. Thus, a gigabit size DRAM actually stores only 0.5 gigabits of data (128 512 Kbyte pages). Using the power estimation presented in Section 3.1 we can estimate the power consumed by a 0.5 gigabit RADram memory chip to be:

$$P_{radram} = 128 \cdot (159mW + 13.4mW) = 17.1W \quad (14)$$

3.3 Power for Processing Elements

The current TLCS-R3900 series processors operate at 74Mhz and 3.3V and have a typical operating current of 110mA. We project the power consumption of the modified TLCS-R3900 series processor running at 100Mhz and 1.65V. Currently, roughly 20% of the power consumption is used by the instruction cache. By reducing the size of this cache by a factor of four, it is expected the power and area of the processing engine will be reduced. We expect the smaller instruction cache to consume one-third the power of the existing cache. Thus, the expected operating current will be 95mA. However, the faster clock will increase the current used to roughly 126mA. Finally, operating at 1.65V we expect the processor core to consume 208mW of power.

Despite this power consumption, the TLCS-R3900 series processing engine is quite large. Nearly 440k transistors are required to implement the base TLCS-R3900 core. It thus becomes impossible to continue with the 50% DRAM and 50% logic ratio when a processing element is used. In order to maintain the 512Kbyte sub-array size, a roughly 63% logic / 37% DRAM ratio is required. This means that only 92 512Kbyte pages will be available on our hypothetical Active Page memory chip. Therefore, total power consumption will be:

$$P_{pe} = 92 \cdot (208mW + 13.4mW) = 20.3W \quad (15)$$

3.4 Power for Hybrid PE/LE Design

Although the true cost of power consumption by a mixed design is not available without an actual design, we can estimate the power by mixing a TLCS-R3900 series processing element with a limited number of reconfigurable logic LEs. Here we have chosen a hybrid design with a limited number (128) LEs along side a fully functional processing element. Additional power will be required to interconnect these two processing engines, but here we assume this interconnection power to be negligible. Thus, each hybrid processing element consumes roughly one-half the power of the reconfigurable logic-only design added to the power consumed by the processing element. Combined these consume 288mW of power at 1.65V.

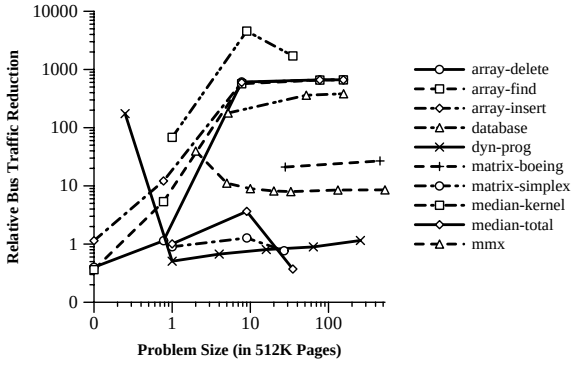


Figure 3: Experimental Results of RADram Relative Reduction in Processor / Memory Bus Traffic

The DRAM to logic ratio of this design is roughly 70% to 30%, thus yielding an Active Page device with only 78 pages of memory. Therefore total power consumption will be:

$$P_{hybrid} = 78 \cdot (288mW + 13.4mW) = 23.5W \qquad (16)$$

3.5 Clock Distribution

In high-frequency, large-area designs the power consumed by the clock distribution and generation network cannot be neglected. The Active Page memory model permits an efficient solution to this problem. Active Page designs do not include hardware support for inter-page communication do not require a globally synchronized clock. An external clock can be used and a large amount of skew can be tolerated. Hence, the internal clock generation network need only consist of regeneration buffers, without an extensive and difficult to design network of clock distribution nodes.

Furthermore, reconfigurable logic designs usually are operated at the highest possible frequency permitted by the design. This suggests that the clock of each Active Page will be distinct, or else operate at some nominal baseline. Furthermore, software techniques discussed in Section 4.1 require variable clock speeds for all Active Page designs. For this reason, we propose that all clocks be generated locally for each page, using internal oscillator techniques. A global clock may be necessary to provide a baseline reference with which to self-time the locally generated clocks.

Given these design constraints, hardware inter-page communication facilities can still be constructed, if desired, using asynchronous logic techniques. Relevant techniques can be found in asynchronous processor cores [FGT+97], caches, and reconfigurable logic array designs [BEHB] [MA98].

3.6 Reduced Memory I/O

As is pointed out by Stan and Burleson [SB97] typically 10-15% of the power used by desktop systems, and 50% of the power in low power systems comes directly from external I/O. This is due largely to the fact that the busses used for off chip communications use higher voltages and have two to three orders of magnitude more capacitance. Substantial amounts of power are consumed every time the voltage on a bus line must be changed.

The number of transitions on the bus can be reduced in two ways: by encoding the bus to minimize the number of transitions per transaction [SB97] [SB95] [WCF+94], or by reducing the total number of transactions on the bus. The use of Active Pages reduces I/O power by addressing the former.

In a traditional system, memory intensive sections of code produce poor cache hit rates which in turn results in large amounts of memory traffic. However in an Active Page system, memory intensive sections of an application are executed inside the memory chip thereby reducing the total number of external bus transactions.

Figure 3 shows relative reduction in bus-traffic for several applications executing on the RADram Active Page memory system. Each application is described in [OCS98], but here we note that a significant reduction in the total number of memory bus accesses can be achieved by the use of Active Pages. The actual reduction in bus traffic varies from application to application, however the general trend of larger problems sizes yielding more efficient bus use can be seen. The tailing off of power efficiency for very large problem sizes is due to the fact that at very large problem sizes the system becomes saturated, and the processor remains active throughout the computation.

3.7 Power Summary

Our estimates show that our initial RADram design using reconfigurable logic has the lowest power consumption of our three alternatives. A further advantage for RADram is increased chip yield due to fault-tolerant designs. The complexity and number of Active-Page functions, however, is limited by the amount of reconfigurable logic available for each page. The processing element and hybrid designs have more computational power. Furthermore, we expect to refine both of the latter designs to use less power. Refinements such as clock-gating, and asynchronous processor designs [FGT+97] will help reduce the power consumed by processor, and hybrid processor / reconfigurable logic processor element designs.

4 Software Power Management

Active Pages presents new design challenges under low-power constraints. One potential solution is to use software mechanisms which trade off performance for decreased power consumption. Most of these mechanisms can be implemented directly in the operating system and can be made to operate across all Active-Page applications, however, specific compiler techniques can be employed which reduce overall power consumption on a per-application basis. For example, Active Page memory devices could enable an operating system to decrease the clock frequency of inactive Active Pages with minimal effects on performance. Furthermore, for extremely low-power environments, an operating system can schedule fewer computations per clock cycle to keep under power budgets. Finally, for reduced power, application partitioning techniques can take into account power consumption and thus balance power consumption versus performance.

4.1 Variables Clock Frequencies

There are two situations where a slower clock frequency can be advantageous for low-power Active Page memory designs. The first is when an Active Page is allocated, but not computing. In this case the clock frequency of the Active Page processing unit can be decreased with minimal impact on application performance. The second is when an Active Page is computing but the application is limited by the processor.

Active Pages which are not computing generally remain in an idle loop checking a synchronization variable. A decrease in clock frequency of the Active Page functional unit can be viewed as an increase in the startup latency. Generally, the computation time is on the order of several thousand clock cycles. Thus, increasing the startup latency from one clock cycle to ten or even a hundred would not seriously affect performance. There would be considerable power savings by such ten or hundred fold decrease in clock frequency.

Active-Page applications often require a balance between processor and intelligent memory. If Active Pages are producing results too quickly for the processor to consume, then their clock frequency may be reduced without affecting application performance. Application characteristics vary widely, however, and can make developing general operating system policies difficult.

4.2 Replacement and Activation Control

At a coarser grain, the operating system can further decrease current draw by limiting the number of Active Pages computing. The operating system must already manage page replacement and schedule Active-Page computation under the constraints of the amount of physical memory available. If the number of Active Pages that fit in a single chip exceeds the power that can be dissipated, then the number that are allowed to compute simultaneously must be limited. Once again, if the application is processor-limited, than performance may not be affected. Furthermore, deactivation of the associated logic does not restrict ordinary memory I/O operations on the memory super-page.

4.3 Compiler Techniques

Future work will address partitioning entire applications. Existing technology exists for such system level partitioning in the form of exact solutions obtained by Integer Linear Programming (ILP), and approximate solutions obtained by genetic and simulated annealing algorithms. Such algorithms balance communication and synchronization costs against potential parallel execution in an attempt to find an efficient overall system partitioning. The same algorithms can also be employed to consider power consumption. Power thus becomes another variable in the minimization goal, and can be balanced by the programmer at compilation time against performance.

5 Future Work

Our previous work [OCS98] has shown that Active Pages are a promising architecture for intelligent memory with substantial performance benefits. Our current work focuses upon the power consumption of Active Page implementations. Estimates show that several design alternatives exist, but Active Page DRAMs will benefit from advances in low-cost packaging technologies. Future work will pursue both architectural and software approaches to reducing power consumption. This work will include detailed chip-level synthesized VHDL models. Furthermore, we will explore the software techniques discussed in this paper using cycle-by-cycle simulation of the processor and memory subsystem.

An increasing amount of research is being done in the area of asynchronous logic, due to the fact that clock skew and distribution are serious issues that a logic designer must face. Clock speed has an effect on both the area and power consumed by a design. As the clock speed increases, more complex structures are needed to distribute the clock accurately throughout the chip. The increased clock speed has two disadvantages. First, the larger clock drivers consume more power. Second, overall power dissipation increases linearly with clock frequency.

Recent developments have made it possible to use asynchronous logic in more complex logic designs. The AMULET2e processor [FGT+97] is an asynchronous ARM[Fur96] 32-bit processor that consumes 30% of the power and takes 54% of the area of the ARM810[Fur96]. An asynchronous RISC core provides a low-power means of implementing an Active Page processing element.

Another possibility is the use of an asynchronous FPGA. Commercial FPGAs are currently targeted to synchronous logic and do not lend themselves to the implementation of asynchronous logic[HBBE94][HBBE92]. Various new FPGA designs have been developed solely for the purpose of implementing self-timed logic[HBBE94][MA98]. Work done at the University of Washington has shown that it is possible to implement real circuits in an asynchronous FPGA[BEHB]. Asynchronous FPGAs may prove to be a viable architecture for Active Page memory system processing elements.

6 Conclusion

This paper evaluates three different Active Page processing-element architectures, and explores the power requirements of each. Due to the many projections required to estimate our power consumption in future technologies, we suspect up to a 30-50% error in our power models. However, these rough calculations suggest that Active Pages will be in line with the power budgets of the commodity packaging technologies of the future [Sem97]. Furthermore, software mechanisms show strong potential for reducing peak and overall power consumption.

References

[Alt96] Altera Corporation. *The Altera Data Book*, 1996.

[BEHB] G. Borriello, C. Ebeling, S. Hauck, and S. Burns. The triptych fpga architecture. Univ. of Washington, Seattle WA 98195.

[DeH96] André DeHon. *Reconfigurable Architectures for General-Purpose Computing.* PhD thesis, MIT, 1996.

[FGT+97] S.B. Furber, J.D. Garside, S. Temple, J. Liu, P. Day, and N.C. Paver. Amulet2e: An asynchronous embedded controller. In *Async '97*, 1997.

[Fur96] S.B. Furber. *ARM System Architecture.* Addison Wesley Longman, 1996.

[HBBE92] S. Hauck, G. Borriello, S. Burns, and C. Ebeling. Montage: An fpga for synchronous and asynchronous circuits. In *2nd International Workshop on Field-Programmable Gate Arrays.* FPL, August 1992.

[HBBE94] S. Hauck, S. Burns, G. Borriello, and C. Ebeling. An fpga for implementing asynchronous circuits. *IEEE Design and Test of Computers*, 11(3), Fall 1994.

[HW97] J. Hauser and J. Wawrzynek. Garp – a MIPS processor with a reconfigurable coprocessor. In *Symp on FPGAs for Custom Computing Machines*, Napa Valley, CA, April 1997.

[I+95] K. Itoh et al. Trends in low-power RAM circuit technologies. *Proceedings of the IEEE*, 83(4), April 1995.

[I+97] K. Itoh et al. Limitations and challenges of multigigabit DRAM chip design. *IEEE Journal of Solid-State Circuits*, 32(5):624–634, 1997.

[Jon98] D. Jones. The ultimate risc. *ACM Computer Architecture News*, 16(3):48–55, 1998.

[M+97] Masato Motomura et al. An embedded DRAM-FPGA chip with instantaneous logic reconfiguration. In *1997 Symposium on VLSI Circuits*, 1997.

[MA98] K. Maheswaran and V. Akella. Pga-stc: Programmable gate array for implementing self-timed circuits. *International Journal of Electronics*, 84(3), 1998.

[NTM+95] Masato Nagamatsu, H. Tago, T. Miyamori, et al. Ta 6.6: A 150mips/w cmos risc processor for pda applications. In *Proc of the International Solid-State Circuits Conference*. IEEE, 1995.

[OCS98] Mark Oskin, Frederic T. Chong, and Timothy Sherwood. Active pages: A computation model for intelligent memory. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA'98)*, Barcelona, Spain, 1998. To Appear.

[San96] S. Santhanam. Strongarm sa110: A 160mhz 32b 0.5w cmos arm processor. *HotChips*, VIII:119–130, 1996.

[SB95] M. R. Stan and W. P. Burleson. Bus-invert coding for low-power i/o. *Transactions on VLSI Systems*, 3(1), March 1995.

[SB97] M. R. Stan and W. P. Burleson. Low-power encodings for global communication in cmos vlsi. *Transactions on VLSI Systems*, 5(4), December 1997.

[Sem97] Semiconductor Industry Association. The national technology roadmap for semiconductors. http://www.sematech.org/public/roadmap/, 1997.

[Tos98] Toshiba. *TOSHIBA TC59S6416 Datasheet*, Janurary 1998.

[WCF+94] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergaele, and H. D. Man. Global communication and memory optimizing transformations for low pwer systems. In *Proc. Int. Workshop Low Power Design*, Napa Valley, California, April 1994.

Power Reduction by Low-Activity Data Path Design and SRAM Energy Models

Mir Azam, Robert Evans and Paul Franzon

Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695
email: msazam@eos.ncsu.edu, paulf@eos.ncsu.edu

Abstract

This paper is focused on different levels of circuit design for efficient power management and speedup. The material is presented in two major categories — circuit modifications to save energy in the data path and the memory, and architectural modifications in the data path for power and performance optimizations. More specifically, we present a low power adder design technique, optimal SRAM design issues, and an execution bypass technique.

1 Introduction

Low power and high performance are pursued at all levels of circuit design. The circuit and architectural levels are emphasized here. The low-activity adder reduces energy consumption by 20%–50% for different frequencies and for process corners. The measured consumption of the fabricated circuit shows power savings between 13% and 30%. The computation bypass technique shows multiplier power savings of up to 60% and the load unit savings of up to 40%. The SRAM models delineate different levels of modeling versus their accuracies. The simulation and analytical based mixed model of the SRAMs prove to be the most accurate. The global decoder and the precharge circuitry in the SRAMs consume the biggest share of energy (each with 30% of total SRAM consumption), and the ATD address data latches also consume a significant portion (20%) of the total energy budget.

2 Low-Activity Adder

Unaligned signals in the middle of combinational logic block cause unwanted transitions or glitches. These signals are aligned with self-timed latches called Fresh Start Latches (FSLs). Since glitching activity increases exponentially with logic depth, a significant portion of the glitching is eliminated by having aligned signals after a certain number of logic stages.

2.1 Signal Alignment

Figure 1 graphically demonstrates the timing management of the FSLs. The combinational logic is split into two parts — *C/L-1* and *C/L-2*. The outputs of C/L-1 are unaligned to a certain degree because they already went through several logic stages. But they are not as badly misaligned as the later stages in C/L-2. FSLs align these outputs of C/L-1 and thus provide a fresh set of aligned signals as inputs to to C/L-2 —- hence the signal alignment latches are named *Fresh Start Latches.*

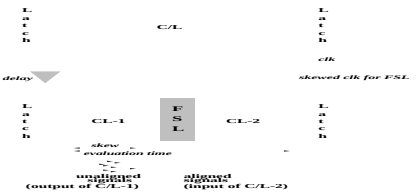


Figure 1: *Timing management and signal alignment with Fresh Start Latches.*

FSLs have to operate with a clock which is skewed enough with respect to the system clock going to the conventional latches sandwiching the combinational logic. Therefore, the constraint for the skew is such that the skew is large enough to allow C/L-1 to settle down, but not so large such that C/L-2 does not have enough time to evaluate. Other than these two constraints, there are no other major requirements for the amount of skew. The FSLs in different C/L blocks need not be aligned. Signal alignment in different logic blocks can be done with their own self timed Fresh Start Latches. Figure 2 shows that the signals in two logic blocks are aligned at different time instances. But both skews are large enough to have their respective first sub-blocks of logic to evaluate. And both skews are small enough to have their respective second sub-blocks of logic to finish evaluating before the next edge of the *system clock* so that the correct outputs can be conventionally latched. Thus the difference in skewed clocks for the different sets of FSLs do not have any effect on the correct functionality of the circuit. There is no absolute global skew required to operate the self-timed FSLs. Thus the local skews can be derived from the local portion of the cock distribution network through custom delay elements which match the required delay at that portion of the circuit.

2.2 Implementation of FSLs

The concept of Fresh Start Lathes was implemented on a 32b ripple carry adder in 0.6μ technology. A ripple carry adder was chosen because it is structured and easy to implement. An RC adder has a pattern of a chain of dependent logic through its propagated carry. This kind of

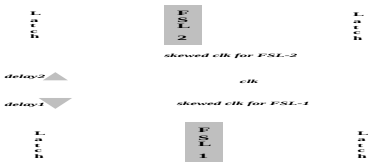


Figure 2: *Different skews for Fresh Start Latches.*

construction is easy to partition with fresh start latches. Furthermore, RC adder has the least activity power compared to any other adder architecture because it has the least amount of gates in its design.

Figure 3: *Micro-photograph of the low power chip.*

Figure 3 shows the photomicrograph of a 32b ripple carry adder with FSLs incorporated, a conventional RC adder, some random number generators and signature analyzers. The outputs of the random number generators feed to the inputs of both the adders. The outputs of the adders are fed to the inputs of the signature analyzers. The signature analyzers simply take the 32b sums and serialize their outputs through shift registers. This way, instead of analyzing a pair of 32b sums, we can now look at two signatures and verify that the adder outputs are the same.

3 Computation Bypass

Conventional data paths show sensitivity to only the control and data dependencies. After the control dependencies are predicted and data dependencies are solved, instructions are executed in the functional units. However, these executions do not consider the values processed by instructions in standard pipelined DSP and/or superscalar architectures. We proved microarchitectural features detecting the repetitiveness of the *computations* (not *instructions*) can significantly reduce processor power requirements [1] or increase performance [2].

3.1 The Execution Cache

The run-time hardware monitors we proposed are called Execution Caches (ECs) [1, 2]. An execution cache simply holds on to old computations — the two source operand values and the computed result. The ECs are distributed with the functional units, and therefore, there is no need for an

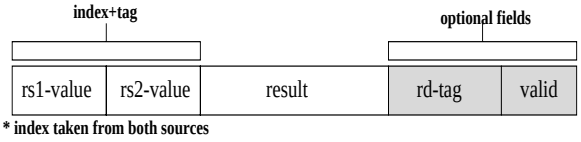


Figure 4: *Bitfields belonging to each entry in an execution cache.*

opcode except for the precision bits and subtle operational differences specified by only a portion of the opcode. The ECs are also separated within a functional unit, if the opcode produces different results for the same operand values even though they may be processed by the same functional unit. In other words, the access to the execution caches are dependent only on the operand values.

Figure 4 shows a line for the execution cache. Previously computed results are stored in the execution caches. When a new computation is attempted, the appropriate execution cache is checked for a redundant computation — a computation whose result is already available in the EC. There are three cases for computation lookup. They are discussed below.

Single Cycle Computation Lookup Simultaneous with Execution

The execution caches are single cycle lookup modules. Therefore, it does not seem beneficial to bypass single cycle instructions. One can argue that execution of the instruction can be performed instead of a lookup. But, there may not be enough execution units available to process the instruction. Execution units are expensive, and an EC provides a cheap way to provide the result if it is available. The only catch is that the EC does not guarantee a result like a functional unit can. The trade off, therefore, is the hit rate versus the extra cycle penalty and the power cost paid for having an EC in the functional unit.

Multiple Cycle Computation Lookup Simultaneous with the First-Cycle of Execution

Multi-cycle instructions can be looked up for repeated computations either in the case of a resource conflict or it can always be looked up in parallel with the first cycle of its execution. If the computation is found in the EC, the execution can be suspended. This not only provides performance enhancement by early completion of instructions, collapsing followed flow dependencies faster and resolving branches earlier, but this execution method also reduces switching activity for bypassed computations.

Multiple Cycle Computation Lookup with an Added Lookup Cycle

Multiple cycle instructions can also be looked up by adding a lookup cycle before execution of long latency instructions. In this way, switching activity is completely eliminated for redundant computations. In the case of a resource conflict, this does not even pose the extra cycle penalty for unavailable computations. But, for long latency instructions which have no resource conflicts, the extra cycle is needed when an instruction is executed instead of bypassed. This additional cycle penalty is also incurred for branch mispredictions in the stream and for flow dependency collapsing.

3.2 Integration of SRAM-ECs

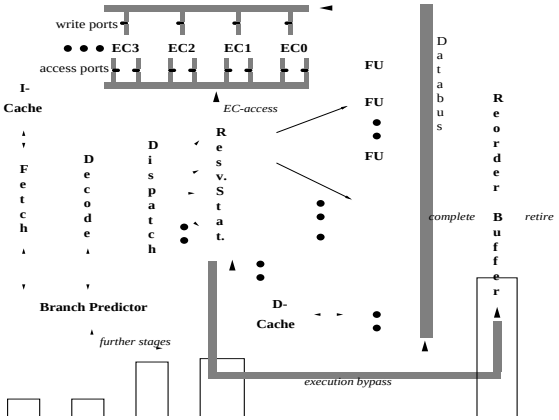


Figure 5: *Generic integration of execution caches in a superscalar data path.*

The generic integration of the ECs for a superscalar data path and the added cycle integration are shown in Figures 5 and 6. In Figure 5, the dispatched instructions are executed out of order, but EC lookups can be made for selected instructions. If enough performance advantage can be gained, then the power cost to perform the lookup can be less than the power cost to have a longer program execution time. Also, for each individual functional unit (especially for long latency units), the energy per computation can be analyzed locally. Figure 6 shows the EC integration with added lookup cycle preceding the execution.

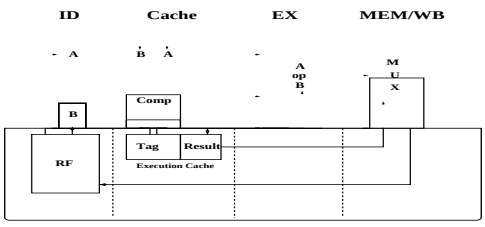


Figure 6: *Simple Bypass through operand tagged caching with added lookup cycle.*

The computation replacement in the EC can be initiated either at instruction completion or at the Reservation Stations. The simplest way to replace and later bypass instructions is to replace the entire EC line at the completion of an instruction. At a later time, when an instruction with the same computation could be bypassed, it is simply looked up in the EC, the result grabbed and the execution of the instruction is avoided. This bypass technique is shown in Figure 7. However, computation source operand values can be replaced at the reservation stations. Then the entry needs to be invalidated, and validated again when the result is available after execution. Due to register renaming and multiple tags associated with each register, the register tag of the first computation to be executed needs to

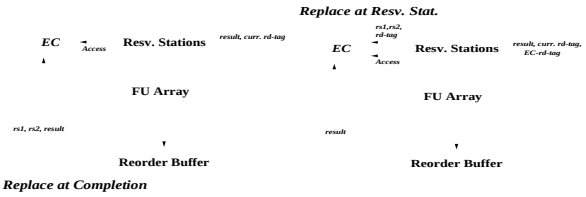


Figure 7: *(LEFT) Replace at Completion (RC) and (RIGHT) Replace at Resv. Stat. (RR).*

be stored in the EC as well. When this computation bypasses an instruction still marked *invalid*, meaning that the correct result is pending, both the tag in the EC and the new instruction destination register tag need to be passed along so both results can be associated with their respective register IDs. This replacement and bypass strategy is not recommended for its complications. In addition, we will see later that replacement of the operands at reservation stations actually has much less effect than replacement at completion.

4 Power Models for the SRAMs

In this section, we will switch gears to discuss several power models for the SRAM array architecture. The key is to know how accurate each model is, and make a decision about SRAM architectures as early in the design cycle as possible.

The Relational Sizing Model

In the relational sizing model, only the interconnect lengths are considered. The SRAM is organized in rows and columns. The word line and the bit line lengths are considered to get a relative estimate of power delay tradeoffs. This model does not produce an absolute number for the energy consumption, but provides insights to how the SRAM array should be constructed given the capacity of the memory. This is a simple model whose row enable energy can be shown as

$$E_{RowEnable} \propto 2^m.y + 2^n.x$$

where, m = # row address lines, n = # column address lines, x = horizontal size of a single memory cell, and y = vertical size of a single memory cell.

Analytical Characterization Model

The analytical based model provides information on the relative contributions of each model element. The advantage of this model is that this capacitance based model allows for energy estimates of the whole SRAM architecture, or even down to the fine grain detailed estimation of each model component, or any combination of these models. The capacitances of these elements are listed in Table 1.

Simulation Characterization Model

The simulation based characterization model uses detailed transistor descriptions to estimate the energy consumption from simulating extracted circuit. The power

Table 1: *Element descriptions in the analytical model.*

Element	Description	Typ. Val. in 2μ process fF or J
C_{gmin}	Cg (min transistor)	$2^m.2.(7.77) + 2^n.(8.17)$
C_{gOthr}	Cg (non-min transistor)	0
C_{dfMin}	Diff C (min transistor)	0
C_{dfOthr}	Diff C (non-min transistor)	2.(65.1)
C_{CdfSm}	Avg diff C	2.(65.1)
C_{intCrs}	IntConn C (m1-m2 X-over)	$2^{m-1}.3.(0.40)$
C_{intOth}	IntConn C (gate routing)	$2^m.(y).(0.08) + 2^n.(x).(0.23)$
C_{intPlG}	IntConn C (gate poly)	$2^m.(6.0)$
C_{intDcX}	IntConn C (short X-stubs)	0
C_{intSm}	Avg IntConn C	$2^m.y.(0.109) + 2^n.(x).(0.109)$
E_{thrMin}	Thru-I En (min transistor)	$6.518e^{-5}.r_1$
E_{thrOth}	Thru-I En (non-min trans)	$4.458.e^{-12} + 1.304e^{-4}.r_2$

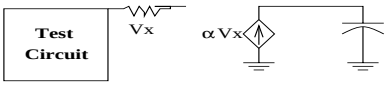


Figure 8: *Power meter for simulation estimator.*

meter technique is a popular one which operates a voltage-controlled current source to dump charge on a capacitor. The VCCS is fed off a resistive drop caused by the supply current to the simulated test chip. Figure 8 shows a diagram illustrating this technique. The SRAM circuit spec's designed and simulated in 2μ process is shown in Table 2.

Table 2: *SRAM ckt. subsections and their spec.*

Subcircuit	Model Spec.
Row Decoder	3 sizes; m=3,4,5
Row Enable	3 Row sizes; m=3,4,5 4 Word line sizes; n=4,5,6,7
Column Decoder	3 sizes; n=3,4,5
Column Enable	3 sizes; n=3,4,5
Global Decoder	1 size; p=4
Precharge	3 types; A, B, C 3 column line sizes; m=4,5,6
I/O Lines and Precharge	2 memory subarray widths; n=4,5
Global Mux TG Drivers	4 sizes; 4, 16, 32, 64 gates driven
Global Mux Thru Drivers	2 sizes; 8, 16 gates
Long Int Conn Lines (plain)	5 sizes; $L_{intConn}$=500,1000,2000,8000μm
Long Int Conn Lines (drivers)	3 sizes; $L_{intConn}$=500μm, k=8,16 $L_{intConn}$=1000μm, k=8
Sense Amp	1 size; 3 fabricated for averaging
Write Drivers	1 size; 2 fabricated for averaging
Off-Chip Drivers	1 size; 2 fabricated for averaging

The Mixed Characterization Model

The mixed characterization model is a combination of the analytical and the simulation based models. Mixed characterization model should be applied to the subsections of the circuit displaying fully digital behavior with full voltage swings (e.g. latches, decoders, buffers, off-chip drivers, etc.). The characterization equations are determined in the individual analytical and the simulation based models; therefore, there are no independent characterization equations for this technique.

The Measurement Based Model

The measurement based technique is the most accurate model for CMOS circuits. ICs are fabricated, the power rails are isolated for different subcomponent, and then simply measure the power drawn by each of the components at different frequencies. This is the most accurate model. However, this is the most difficult model to build, since there is a significant amount of design effort involved, the process corners considered, etc.

The Transistor Count Model

Table 3: *Transistor count chart for 32b data values.*

Logic Family	Transistor Count		
	MPYer	SRAM-array Access	Ratio
Static	49934	4628	10.8
Dynamic	42671	4116	10.4
CPL	38805	3860	10.1

The transistor count involved in each SRAM-array access and data drive-out was done with information from the circuit of an SRAM-array-access. Likewise, the transistor count involved for a multiplier was done with information from the circuit of the Wallace Tree multiplier construction. The major components considered to determine the energy rating of an SRAM-array were the tag comparators and the bit line and the word line drivers. The driver sizes were determined from different SRAM-array sizes since the drive strength changes with varying cache sizes. For the multiplier, for example, an 8b-MPYer is composed of two 4×8 partial product generators, 27 4×2compressor MUXes and a 16b-ADDer. With these approaches, the transistor count for the components involved are as listed in Table 3 for direct mapped caches and Wallace Tree multipliers.

5 Results

Figure 9 shows the percentage of average power (not RMS power) saved with the low power adder. The savings were between 13% to 30% for the fabricated circuit. Table 4 lists the RMS power consumption for all process parameter combinations, and the RMS savings ranged between 20% and 50%. Figure 10 illustrates the measured versus simulated average consumptions.

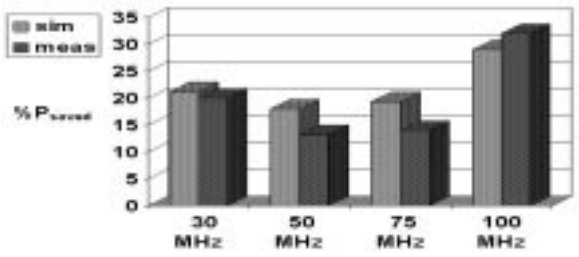


Figure 9: *Percentage of power saved with the low power adder compared to the conventional adder (simulation and measured).*

Figure 11 shows up to 60% energy savings per MPY operation for small and simple execution caches. Figure 12

Table 4: *Power consumption of the low power adder compared to a conventional adder.*

Power-Delay		Power Consumption (in mW) due to Process Variation								
MHz	↓Pwr	fP,fN	nP,nN	sP,sN	fP,nN	fP,sN	nP,fN	nP,sN	sP,fN	sP,nN
30	P_{conv}	9.85	8.75	8.25	9.08	8.58	9.38	8.42	8.85	8.52
	P_{LP}	6.75	6.08	6.57	6.38	5.97	6.92	5.98	6.50	6.36
	$\%P_{saved}$	**31%**	**31%**	**21%**	30%	30%	26%	29%	27%	25%
40	P_{conv}	11.82	10.20	9.23	10.81	10.25	10.51	9.65	10.17	9.73
	P_{LP}	6.90	6.65	6.85	6.68	6.42	7.06	6.45	6.72	6.76
	$\%P_{saved}$	**42%**	35%	**26%**	38%	37%	33%	33%	34%	30%
50	P_{conv}	13.97	12.91	11.99	13.42	12.82	13.35	12.94	12.88	12.64
	P_{LP}	8.21	7.38	7.68	8.01	7.93	7.88	7.37	7.58	7.59
	$\%P_{saved}$	41%	**43%**	**36%**	40%	38%	42%	**43%**	41%	40%
75	P_{conv}	16.56	15.20	14.19	16.04	14.01	14.45	14.47	15.17	15.03
	P_{LP}	8.63	8.07	8.20	8.30	8.28	8.34	8.01	8.17	8.30
	$\%P_{saved}$	**48%**	47%	42%	**48%**	**41%**	42%	46%	46%	44%
100	P_{conv}	18.95	17.46	16.19	18.32	17.51	18.03	16.83	17.40	17.01
	P_{LP}	9.88	9.67	9.32	9.43	9.65	9.37	9.33	9.38	9.3
	$\%P_{saved}$	48%	45%	**42%**	**49%**	45%	48%	44%	46%	45%

Table 5: *Absolute energy predictions by different energy models for a 4 Mb SRAM array.*

SRAM Subcircuit	Consumed Energy (J)							
	Analytical		Simulation		Measurement		Mixed	
	Rise	Fall	Rise	Fall	Rise	Fall	Rise	Fall
ADL Latch, Drive	1.61e-9	2.42e-11	1.60e-9	2.42e-11	N/A	N/A	1.61e-9	2.42e-11
Row Decoder	5.94e-10		5.45e-10		8.96e-10		5.94e-10	
Column Decoder	3.50e-11		3.26e-11		6.85e-11		3.50e-11	
Row Enable	1.03e-11	6.79e-10	1.01e-10	4.82e-10	3.52e-10	6.71e-10	1.03e-11	6.79e-10
Column Enable	5.57e-12	2.70e-11	8.91e-12	2.35e-11	3.24e-11	1.20e-10	5.57e-12	2.70e-11
Global Decoder	8.55e-11		7.94e-11		2.01e-10		8.55e-11	
Subarray Enable	1.89e-11	4.91e-10	N/A	N/A	N/A	N/A	1.89e-11	4.91e-10
Subglobal row Dec/En	2.06e-11	2.05e-9	2.03e-11	2.04e-9	N/A	N/A	2.06e-11	2.05e-9
Precharge En	8.92e-12	1.56e-10	0.00	1.70e-10	N/A	N/A	0.00	1.70e-10
Prchg Col Read	1.81e-9		9.87e-9		1.53e-8		9.87e-9	
Prchg Thru-I Energy	4.56e-12		2.26e-12		2.32e-12		2.26e-12	
Sense Amp	2.51e-12		3.71e-12		3.94e-12		2.51e-12	
Off-Chip Driver	5.25e-10		6.35e-10		7.32e-10		5.25e-10	
Total Energy (Read)	8.45e-11		1.90e-8		2. 10e-8		N/A	

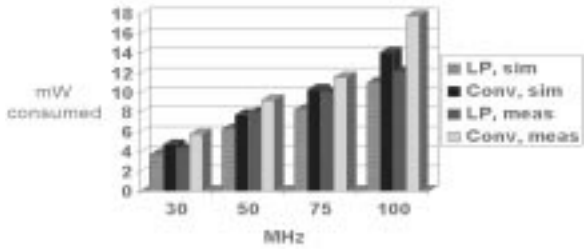


Figure 10: *Total power consumption of the low power and the conventional adders (simulation and measured).*

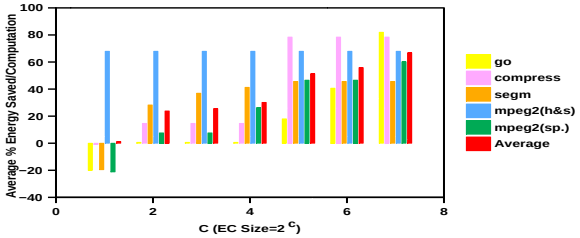


Figure 11: *Energy savings per MPY.*

plots the LOAD bypass savings. The load savings were up to 40% per load. The load bypassed a 32KB 2wsa data cache and the traffic from the main memory. The performance advantages are shown in Figure 13. SPECint95 improvement in IPC was up to 15%, SPECfp95 enhancement as much as 30% and multimedia algorithms showed as high as 45% improvement in IPC. The SRAM energy predictions are illustrated in Tables 5, 6 and 7. Table 7 shows that most of the energy consumption is in the global decoders, precharge circuitry and the address data latches. The row enable and bit-line precharge energy consumptions are plotted in Figures 14 and 15 which show the measurement model to be the most pessimistic and the analytical model to be the most optimistic.

6 Conclusion

Circuit and microarchitecture can play an important role in low energy microprocessor design. A few examples are given here. Low power components in the data path can be designed with retimed of signals and also by trading off result lookup with computation activity. In addition, energy consumption for memory circuits can be estimated early in the design cycle with reliable models. Signal aligned adder design can improve power consumption by as much as 50%. The fabricated circuit showed up to 30% improvement.

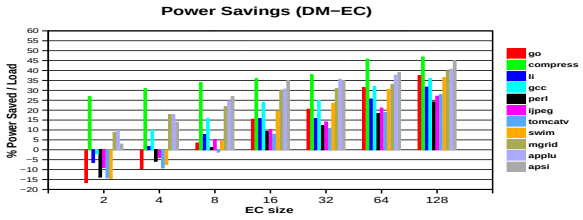


Figure 12: *Energy savings for LOAD.*

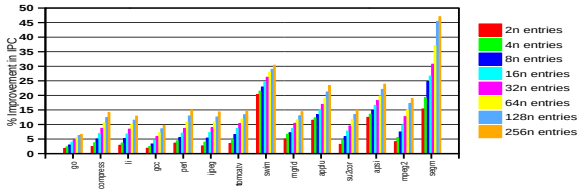


Figure 13: *Performance improvements in memory driven data paths.*

Computation bypass can cut down power consumption by half with small execution caches. Performance of processors can be elevated with such mechanisms as well. The memory consumption estimations are verified with fabricated circuits which show that the simulation based estimation is within 10% of the measured consumption. Memory energy is mostly dissipated in the decoders (31%), precharge circuits (30%) and data latches (20%). Such multi-level design approach is helpful to optimize power consumption to the fullest.

References

[1] M. Azam, P. Franzon, and W. Liu, "Low Power Data Processing by Elimination of Redundant Computations," in International Symposium on Low Power Electronics and Design, pp. 259–265, August 1997.

[2] M. Azam and P. Franzon, "Memorized computations for speedup and power conservation," Submitted to the Transactions on Computers, December 1998.

[3] R. Evans, "Energy consumption modeling and optimization for srams." Ph.D. Thesis, June 1993.

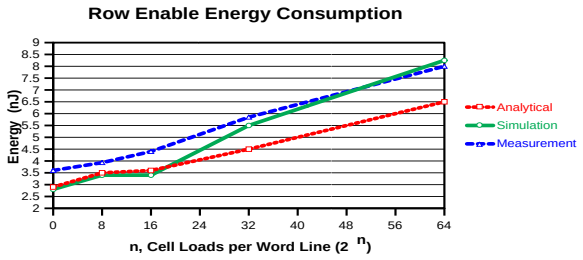


Figure 14: *Row enable energy comparison predicted/measured by different models.*

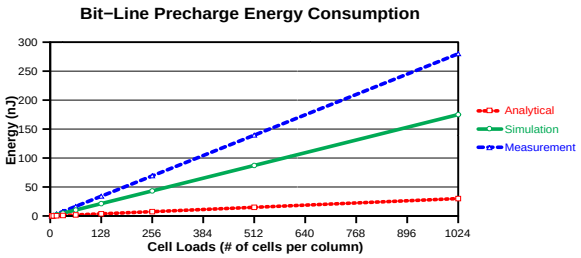


Figure 15: *Bit line energy comparison predicted/measured by different models.*

Table 6: *Comparison among the energy models.*

SRAM Arch.	Opt. Arr. mnp(p0p1)	Energy Consumption (J) Sim	Analyt.	Mixed
SnglBlk(4kb)	7-5	3.55e-9	1.35e-9	3.45e-9
SnglBlk(4Mb)	12-10	1.16e-6	2.46e-7	1.16e-6
BusSbArr	9-8-5	7.23e-8	3.00e-8	7.27e-8
MuxSbArr	9-4-9	1.09e-8	3.28e-9	7.48e-9
DWL	11-5-6	2.80e-8	1.25e-8	1.89e-8
HWD	10-5-7(3-4)	2.35e-8	1.33e-8	2.023e-8
DWL R&C	10-5-7	1.90e-8	8.45e-9	1.92e-8

[4] R. J. Evans and P. D. Franzon, "Energy consumption modeling and optimization for srams," IEEE Journal of Solid State Circuits, vol. 30, pp. 571–579, May 1995.

[5] A. Chandrakasan and R. Brodersen, Low Power CMOS Digital Design. Kluwer Academic Publishers, 1995.

[6] S. Wuytack, F. Catthoor, L. Nachtergaele, and H. D. Man, "Power Exploration for Data Dominated Video Applications," in International Symposium on Low Power Electronics and Design, pp. 359–364, August 1996.

[7] P. Landman, "High-Level Power Estimation," in International Symposium on Low Power Electronics and Design, pp. 29–35, August 1996.

[8] D. Dobberpuhl, "The Design of High Performance Low Power Microprocessor," in International Symposium on Low Power Electronics and Design, pp. 11–16, August 1996.

[9] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocesosrs," IEEE Journal of Solid State Circuits, vol. 31, pp. 1277–1284, September 1996.

[10] Shyh-Jye Jou et al., "A pipelined multiplier accumulator using a high-speed, low-power static and dynamic full adder design," IEEE Journal of Solid State Circuits, vol. 32, pp. 114–118, January 1997.

[11] T. Xanthopoulos, Y. Yaoi, and A. Chandrakasan, "Architectural Exploration Using Verilog-Based Power Estimation: A Case Study of the DCT," in International Design Automation Conference, March 1997.

[12] S. E. Richardson, "Caching function results: faster arithmetic by avoiding unnecessary computation," tech. rep., Sun Microsystems Laboratories, 1992.

[13] A. S. Hung and J. P. Shen, "A limit study of local memory requirements using value reuse profiles," in Proceedings of the 28th Annual International Symposium on Microarchitecture, vol. 28, pp. 71–81, December 1995.

[14] A. Sodani and G. Sohi, "Dynamic instruction reuse," in International Symposium on Computer Architecture, pp. 194–205, June 1997.

Table 7: *Distribution of energy in active SRAM.*

SRAM Subcircuit	% of Total Energy
ATD Address Data Latch	21%
Row Decoder	7%
Word Line	9%
Column Decoder	1%
Global Decoder	31%
Precharge	30%
Sense Amp	–
Output Driver	1%

Cache-In-Memory: A Lower Power Alternative?

Jason T. Zawodny, Eric W. Johnson†, Jay B. Brockman, Peter M. Kogge

University of Notre Dame
†Valparaiso University
{jzawodny,ejohnson,jbb,kogge}@cse.nd.edu

Abstract

The concept of Processing-In-Memory (PIM) achieves many benefits by placing memory and processing logic on the same chip, utilizing more of the available bandwidth and reducing both the power and time associated with memory accesses. Typical architectures, even current single-chip microprocessors with on-chip memory, rely on traditional cache hierarchies. This paper serves to explore the possible performance and power improvement of "migrating" the cache into the memory itself. This idea, which we call Cache-In-Memory (CIM) utilizes the sense-amplifiers inside the memory array to perform this function. Early results indicate that the most trivial implementations of this technique provide comparable power efficiency with reasonable performance. However, the increase in resident DRAM has yet to be explored. Ongoing work strives to further solidify the tools needed for energy and performance simulations as well as explore the potential capacity benefits as well.

1 Introduction

As part of an attempt to explore alternative memory configurations that provide equal performance while using less energy than traditional cache hierarchies, CIM moves the "cache" into the memory structure itself. The key to this arrangement is the utilization of latching sense-amplifiers in the memory structures to act as "free" cache-lines. With the appropriate supplementary memory arrays, equivalent to the tag arrays found in typical caches, the architecture itself can remember what was last accessed.

Others have explored the advantages of putting a processor core on a chip with large amounts of DRAM [1][3][4]. That work considered the memory as a "black box," accepting current DRAM structures without modification. Newer studies have investigated the effects of retaining entire lines of DRAM within the memory array using the sense-amplifiers, those projects looked at performance [7]. This study approaches the idea from a power efficiency standpoint.

In this paper, we discuss the ongoing simulation of the CIM architecture and the results derived from preliminary simulations. By closely coupling memory energy and time simulations and microprocessor pipeline emulators, it is possible to characterize both the time and energy performance of various configurations. The data presented in this study comes from a comparison of 16KB of row buffer in the CIM architecture against split 8KB data and instruction caches in the traditional cache arrangement. The latest results in this continuing work indicate that CIM is capable of providing comparable memory energy consumption at reasonable performance levels. Further work will help to validate our current results as well as explore the advantages that CIM provides in DRAM capacity given constant die size.

2 Memory Structure

Traditional memory hierarchies employ caches to hide the latency associated with an access to off-chip memory devices. These caches are stand-alone entities, often clearly distinguishable on photo-micrographs of processors. While caches have been mandatory in the past, decreases in feature size and increases in die size have spawned the capability to put conventional microprocessor cores and large amounts of DRAM on a single chip [8]. The first generation of these devices have simply integrated the traditional hierarchy on the single device [11][12]. Studies have shown show how much performance and energy gain can be obtained from creating systems on a single chip while leaving the memory arrays virtually unchanged [3][4]. And there have been some performance estimates of more interesting ways to utilize the proximity [7].

Memory devices are constructed of smaller arrays. The reasons involve the ability to drive and sense signal changes across limited distances due to capacitance in the wiring, as well as providing convenient redundancy. If the whole system is on the single die, could there be an arrangement of memory, processor, and cache functions which will enable equal performance with significant reduction in power compared to traditional, separate caches?

Figure 1 shows two obvious options for systems on single chips. The most basic approach would be to throw away the caches: simply access the existing memory arrays. While this would provide the highest DRAM capacity on the chip, the enormous time and energy penalties for fully decoding and sensing each memory reference would be inefficient. The traditional memory hierarchy implements unified or split first-level SRAM caches,

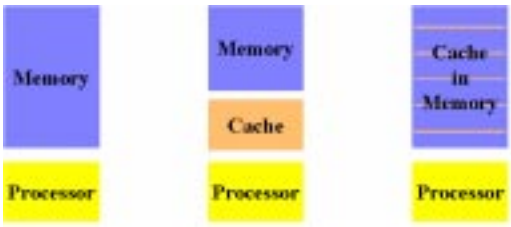


Figure 1:
Single Chip Memory Options

which take up vast amounts of die area that could be used for more DRAM. This yields good overall performance, but at the expense of DRAM capacity. Figure 1 also begins to explain how CIM differs from the previous two ideas. In the CIM depiction, we see that the traditional cache has vanished, and the area not covered by the processor is allocated for DRAM occupancy. However this DRAM has some special characteristics allow it to take the role of cache structures.

A more detailed look at the DRAM can be found in Figure 2. The primary features of this array are the latching sense amplifiers distributed throughout the array. Memory is normally broken down into subarrays, because of capacitive effects, etc. The subarray is the first of three primary entities that are dealt with in the CIM structure. The subarray is the block of memory where the rows reside, as in regular memory devices. An entire row is then sensed and latched into the sense amplifiers. After the row has been latched, it is multiplexed down to usable words. The result of this multiplexing, some subset of the columns that were read into the row, is called the subrow.

Each of these subarrays has its own row of sense amplifiers, and its own subarray control for decode and timing. This control interfaces with a tag-array that keeps track of the row that was last accessed for each of the subarrays. Based on whether the requested address matches the current subarray and subrow, and whether the row is already cached in the appropriate subarray, certain events are required to satisfy the read or write. Different decoding is required based upon how much of the address matched, and sensing directly from the memory cells need not repeat each access. Section 3 will discuss what causes these different events.

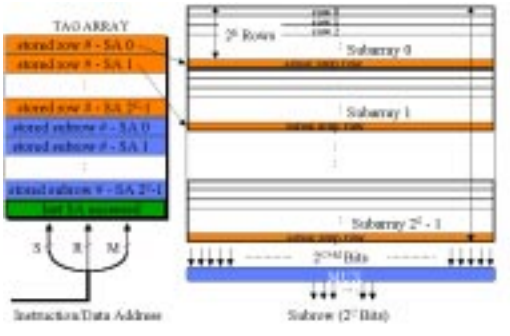


Figure 2:
CIM Memory Architecture

3 CIM Hit/Miss Taxonomy

To measure the effectiveness of the CIM structure, a taxonomy was developed that describes what occurs when an access to the memory array is made. The taxonomy is similar to one outlined for a typical memory hierarchy with various "hit" and "miss" definitions. These definitions are used to determine the performance of the CIM structure both in terms of time and energy consumption.

In CIM, the characteristic that has the most impact on performance and energy is whether the requested data is available in one of the sense amp rows. If the data is available in the sense amplifiers of the appropriate subarray, a hit is said to have occurred. If it is not available, then a miss has occurred, and another access is required to load the sense amps with the new row of data. Since a row is stored in the sense amps, it is not required to re-write the data into the row of DRAM cells after each read. Because the data stored, it can be written back only when displaced without adverse effects.

To more effectively measure the performance characteristics of the memory structure, both the hit and miss definitions can be further expanded. There are two other factors that can impact memory performance: whether current subrow multiplexing and active subarray match that which has been requested. Our taxonomy was expanded to include these conditions. The three factors, each with two options, leads to eight possible outcomes. These factors are shown in binary tree format in Figure 3.

Figure 3:
CIM Access Taxonomy

The names associated with each leaf-node in the tree are largely arbitrary, but are picked to indicate to what degree a given address matches cached data and current active elements. The subarray is decoded from the most significant bits of the address. The row is then decoded from the middle bits, followed by the subrow. The word and byte bits are, of course, the least significant bits, but these do not matter to the memory array.

4 Description of the Model

Some initial experiments with CIM were conducted using the Shade [2] tool from Sun Microsystems, Inc. Shade is a library of functions that allows the programmer to trace and analyze the instructions that are executed in most any program.

It returns the information for each instruction, and allows the user to extract and utilize any data that he/she sees fit.

In our models, we utilize most of the instruction information at one point or another. Basic simulations look only at the memory accesses that were represented in the instructions. The addresses accessed were utilized as though they came from a single-cycle architecture: no interleaving of addresses was performed to account for pipeline behavior. The resolved addresses are then fed into a representation of the tag arrays required to implement the CIM idea to determine hit and miss rates, as well as estimations of power consumed and time required for each application. These addresses can also be utilized in a similar representation of traditional cache hierarchies to compare the two.

More advanced simulations utilize the details of the instructions themselves. In this case, we chose to simulate the effects and relationships between the IBM PowerPC 401 microprocessor core and the memory. The 401 core was chosen because it is the processor that has been chosen for one or more of the projects which sponsored this study. It is a relatively simple, 32 bit RISC processor available as a synthesized core. The 401 has a wide variety of cache options, including omission of the caches. To translate between the Sun architecture and that of the PowerPC, the largest hurdle was converting the branch behavior. While the Sparc v9 architecture uses delayed and annulling branches, the 401 does not support these features.

Coupling the pipeline simulation with the CIM representation discussed above, we will achieve a complete-chip functional simulation. Again, a traditional memory hierarchy can be integrated in place of the CIM model to compare the two architectures. This simulator is aimed at comparing the time and energy performance of traditional structures to CIM structures. We are integrating energy and timing numbers from separate software to achieve a reasonable degree of realism in these simulations [5].

5 Test Suite

A reasonable effort was expended in collecting typical yet diverse applications to analyze using the Shade-based simulators. These applications are listed and described in in Table 1. The applications include pointer-chasing applications and tight unrolled loops, which should explore the gamut of most cache-performance envelopes.

6 CIM Characterization

Preliminary experiments were performed to look at hit and miss rates for various configurations of the CIM system. These arrangements varied between 128 and 512 rows per subarray and 128 to 4096 columns per subarray using the simple memory access simulator discussed in previous sections. The number of subarrays was not fixed, enabling the simulator to cover the entire address space. With virtual memory, page mapping, etc. the overall application size must be limited. The results of these simulations showed that for several diverse programs and most configurations, CIM produced instruction hit rates in excess of 90% and data hit rates in excess of 60%. While these hit rates are not stellar, they show promise for the performance and energy efficiency of CIM.

Program	Description
ShellSort	A C++ application that implements the shell sort algorithm implemented using the LEDA data structure library on 10,000 floating point numbers.
cc	The Sun performance 'C' language compiler performing source to executable compilation of a simple program (find the maximum of 5 numbers)
FpMatrMult	Multiplies 256x512 and 512x256 element floating point matrices in very tight hand optimized loop.
JpegTran	Translates a 550KB JPEG image into a GIF format image

Table 1: CIM Test Suite

7 Energy Estimation Parameters

As mentioned before, energy values in this study are derived from separate software which computes SRAM size, time, and energy [5]. The basis for this simulator was the IBM 7LD .25μ DRAM and logic process [10]. Using these parameters and requesting moderate energy and time constraints on the SRAM simulator, we were able to generate the tag and data array numbers directly. By specifying a large SRAM array with one sense amplifier per bit line it is possible to get an approximation to DRAM power figures. However, one must modify three things to obtain a good estimate. First, the cell energy must be updated, followed by appropriate scaling of the power used on the bit and word lines. Finally, one must account for another set of sense amplifiers: the primaries lie in each subarray, while the secondary sense amplifiers reside at the base of the stack of subarrays. These factors have been included in these power values. The DRAM numbers are approximations, but we believe them to be adequate for these early studies.

Catagory	Joules (W-sec)
Level 1 Tag Access	2.52e-9
Level 1 Data Access	2.52e-9
Main Memory Access	2.36e-8

Table 2: Traditional Energy Specification

All the basic power numbers that were used in these simulations can be found in Tables 2 and 3. If we look at how these numbers combine to produce per-access energy specifications, we can begin to see more interesting details. Tables 4 and 5 quantify the amount of energy expended per access in each of the different catagories.

Catagory	Joules (W-sec)
Tag Access	7.65e-10
Subarray Decode	7.0e-9
Row Decode	4.13e-9
Wordline	2.78e-11
Bitline	7.0e-9
Mux Energy	2.29e-9
Output Driver	2.35e-9
Sense Amps	7.65e-10
2ndry Sense Amps	5.0e-11

Table 3: CIM Energy Specification

Access Result	Joules (W-sec)
Cache Hit	5.04e-9
Cache Miss	3.116e-8
Writeback Penalty	2.612e-8

Table 4: Traditional Energy Per Access

8 Time Estimation Parameters

Tables 6 and 7 show the time required for each of the access results in the given architectures. These cycle times are based on a 150MHz clock, which is one possible clock frequency for the 401 to be used in the projects funding this study. These time values are ambitious, but reasonable. The processor must be able to obtain a nominal rate of two memory accesses per clock cycle: an instruction fetch and potentially a memory operation. Because the sense amplifiers themselves latch data, the reads and write to CIM should be as fast as SRAM. The 33ns access time for the cache configuration is also reasonable because all the memory references and CIM misses in the simulations are assumed to be on the chip. This means two things. First, there are no high-capacitance busses to drive off the chip, which reduces the time dramatically. Secondly, the DRAM interface to the standard cache can be full width, eliminating the need to do multiple word transfers to satisfy a 32 byte request.

Taking a closer look at of Tables 6 and 7 show the number of cycles required for each access result. Although CIM enjoyed a comfortable advantage in the per access power cost for most hit and miss instances in the energy domain, in the time domain, it has lower performance than the standard cache hit for two of the four CIM hit

Access Result	Joules (W-sec)
Full Hit	8.15e-10
Subarray Hit	5.46e-9
Row Hit	8.69e-9
Subrow Hit	6.40e-9
Full Miss	2.22e-8
Subarray Miss	1.99e-8
Row Miss	1.67e-8
Subrow Miss	1.90e-8
Writeback Penalty	8.63e-9

Table 5: CIM Energy Per Access

Access Result	# Cycles
Cache Hit	.5
Cache Miss	5
Writeback Penalty	5

Table 6: Traditional Cycles Per Access

catagories. Even if CIM managed to match hit rates, it would fall behind in raw cycle comparisons.

Access Result	# Cycles
Full Hit	.4
Subarray Hit	.5
Subrow Hit	2
Row Hit	2
Full Miss	5
Subarray Miss	5
Subrow Miss	5
Row Miss	5
Writeback Penalty	5

Table 7: CIM Cycles Per Access

9 Analysis and Simulation Results

Two approaches are used to examine the behavior of CIM against traditional cache structures. First, we look at the power model and timing model to find inherent characteristics that should lead to recognizable trends in the numerical data. Second, those models are used in the shade simulation to generate data over the test suite applications.

At a very fundamental level, Equations 1 and 2 on Page 5 relate energy to the frequency and energy associated with the hits and misses that were presented above. These equations are simply adapted versions of standard energy calculations for any system. In this instance, however, the adaptation of these equations, coupled with the power and timing data outlined above reveals several trends.

The tag array is accessed at least once on every memory request. This frequency of 1 means that substantial savings on tag access could prove to be an important factor in overall energy savings. For CIM, the tag array contains the same number of entries as there are subarrays in the memory device: in our initial testing, this value is 128. The number of bits in the CIM tag array is 10, nine to store the row that the subarray has latched in its sense amplifiers and a single valid bit. For the traditional cache structure, we have separate 8KB instruction and data caches. Each of the tag arrays for the caches have 2048 entries, each with 24 bits of tag and valid bits. The energy used to access the CIM tag array should be substantially less than what the normal cache architecture. This is indeed the case, as seen in Tables 2 and 3, CIM tag access consume less than one third that of the standard caches.

If we look at the energy consumed on cache hits, we notice that CIM is actually less efficient

$$E_{CIM} = E_{Tag\,Access} + \sum (E_{CIM\,Hits} \cdot F_{CIM\,Hits}) + \sum (E_{CIM\,Misses} \cdot F_{CIM\,Misses}) + E_{Write\,Back} \cdot F_{Write\,Back} \quad (1)$$

$$E_{Trad} = E_{Tag\,Access} + E_{Cache\,Access} \cdot F_{Cache\,Hit} + E_{Memory\,Access} \cdot F_{Cache\,Miss} + E_{Write\,Back} \cdot F_{Write\,Back} \quad (2)$$

in all but one rare case. In Table 4 and 5, one finds that the cache hit is slightly less than all but the CIM full hit. While the full hit consumes half the power of the cache hit, we expect that these are extremely rare. A full hit indicates that the exact same word has been requested as the last access, meaning that no multiplexing, decoding, etc. is required. Obviously this has little bearing on the comparison. These numbers show that the normal cache is more efficient for hits. We have presented approximate hit rates for the CIM architecture of 90% instruction and 60% data. If we look at typical hit rates for split 8KB direct mapped caches, we find rates of approximately 99% instruction and 90% data [6]. This illustrates that the normal cache architecture should be more efficient in terms of power.

Next we examine the energy associated with cache misses. Again looking in Tables 4 and 5 we find that CIM holds an edge in per access energy for misses. Without considering the the frequency weighted average of the different CIM misses, the simple average has only 63% the energy cost of the normal cache miss. Let us also examine the difference between miss penalties: CIM write-backs are one third the energy of their traditional cache counterparts.

Based on these values, we believe that there is a point where CIM becomes more power efficient than traditional arrangements. We can write the mathematical equation that describes where the cross-over occurs, however it requires simplifying Equations 1 and 2 and setting them equal. If we insist on keeping all four types of hits and all four types of misses in the CIM architecture in this equation, the problem becomes multi-dimensional. We can, however, perform some simplifications, or fix some values in order to obtain a reasonably solvable problem. The series of equations below shows the transformation to a simple solution. From this point, you can simply plug in values for the hit rate for either architecture to obtain the hit rate that should produce equal energy in the other. For instance, a hit rate of 85% in CIM would be the energy equivalent of 89% in the traditional cache model. This equation will serve as a useful check for simulator results.

$E_{CIM} \leq E_{Trad}$

$E_{CIM} = 7.65 \cdot 10^{-10} + 6.0 \cdot 10^{-9} \cdot F_{CIM\,Hit} + 2.05 \cdot 10^{-8} \cdot (1 - F_{CIM\,Hit}) + .05 \cdot 8.63 \cdot 10^{-9}$

$E_{Trad} = 2.52 \cdot 10^{-9} + 2.52 \cdot 10^{-9} \cdot F_{Trad\,Hit} + 3.12 \cdot 10^{-8} \cdot (1 - F_{Trad\,Hit}) + .05 \cdot 2.61 \cdot 10^{-8}$

$F_{CIM\,Hit} \geq 1.97 \cdot F_{Trad\,Hit} - .916$

Now that we have examined how we expect the trends to appear in the simulation results, we can look at the end-to-end Shade simulations. Although this simulator has been in development for quite a while, recent attempts to put larger programs through it have revealed precision problems. Using the same energy figures as the simulator has at its disposal, hand calculated values for the energy per access have been substituted into Table 9. The modified Table entries are indicated with an asterik. We will discuss the results with these corrections.

Table 8 shows the hit rates that the simulator achieved using each processor and memory combination. For the most part, the results for the traditional cache structure falls close to the expected figure. Except for the FpMatrMult data result, which we expect to thrash the cache [6], all other programs approach or exceed the 99% instruction and 90% data rates expected. The results for the CIM simulations also show good correlation between previous study and these simulations, meeting or exceeding the 90% instruction and 60% data hit rates.

Program	**SC+ Trad Hit%**	**401+ Trad Hit%**	**SC+ CIM Hit%**	**401+ CIM Hit%**
ShellSort	1.0/.92	1.0/.92	.94/.64	.94/.69
cc	.97/.95	.98/.95	.92/.61	.92/.60
FpMatrMult	1.0/.83	1.0/.83	.98/.57	.98/.57
JpegTran	1.0/.94	1.0/.94	.90/.71	.92/.72

Table 8: Instruction/Data Hit Rates

We reduced the complex and multi-dimensional equations above to derive a equal power rule between the two memory architectures. A quick look at Table 9 indicates that the CIM hit rate never gets quite high enough to break past the traditional cache configuration. A look at the detailed simulator output shows that the highest combined hit rate achieved by CIM is 87.5%. This is equivalent to a 91% cache hit rate. Unfortunately, none of the combined results show an overall hit rate this low.

Program	**SC+ Trad nJ/A**	**401+ Trad nJ/A**	**SC+ CIM nJ/A**	**401+ CIM nJ/A**
ShellSort	6.55	6.48	8.94	8.39
cc	6.11	5.89	7.10	6.38
FpMatrMult	7.14*	7.11*	7.83	7.77
JpegTran	5.50*	5.40*	6.46	5.67

Table 9: Energy Per Access Comparison

As expected, the CIM architecture shows lower performance that the standard cache architecture. The difference seen in Table 10 is between .75 and 2 cycles per instruction. This is a relatively large difference up to 50% increase in execution time.

Note that the FpMatrMult is very floating point intensive. The 401 uses 2 clock cycles to process a floating point operation, while single cycle simulation assumes 1 clock per operation. This

Program	SC+ Trad	401+ Trad	SC+ CIM	401+ CIM
ShellSort	1.76	1.66	3.64	3.56
cc	1.35	1.32	2.35	2.25
FpMatrMult	1.93	2.21	3.24	3.21
JpegTran	1.23	1.16	2.08	2.11

Table 10: Cycles Per Instruction Comparison

is why the values for FpMatrMult using the single cycle simulation with both memory structures in Table 10 are close or below the value using the 401 and perfect memory.

10 Conclusion and Future Work

This paper starts with the supposition that combining main memory, CPU, and cache on the same die is a clear win from a power and performance perspective. The objective is then to explore how much cache is appropriate and how we might take advantage of the structure internal to the memory to reduce it. In the process of performing this study, we have characterized a broader range of events between a classical hit and a classical miss, each with a different power and performance penalty. Equations have been built that relate this to overall energy per access, and an early simulator has been constructed to explore the energy and performance ramifications.

For the single design point explored here, the results are interesting, but somewhat inconclusive in terms of power. CIM shows comparable energy efficiency to traditional cache structures, especially for applications that make poor use of those caches, such as matrix multiply. The time performance is around 50% slower, however, in looking at these results we should remember that this was the first design point simulated. A range of row and column sizes need to be explored. Looking at the effect of increases in associativity [7] and small L1 caches are possible next steps.

We will continue to look at how to choose bits from the address to build each of field for the subarray and row mapping. The bits that are picked out of the address for each field is largely arbitrary: the data does not care where it actually resides. However, mapping these bits appropriately and adapting compilers to make use of the CIM structure, should positively affect hit rates for CIM-based systems.

Further, we will continue to refine the DRAM power and performance values, as well as add size figures. While there should not be a huge change in the energy results obtained so far, developing a more robust DRAM energy model is essential in order to claim conclusive findings. Additionally, we would like to generate a DRAM model that will allow us to accurately compute area to look at trade-offs that may exist between more DRAM capacity filling the size difference between CIM and normal cache architecture.

We believe that the simulator used for this study can be easily modified to provide some confirmation of other studies [3][7]. By simply changing parameter, and adding provision for higher associativity sense-amplifiers, it should be possible to confirm or rebut findings that have been published in past papers. Duplicating those results would help to validate that our simulations are indeed accurate, and lend more credibility to those previous studies that have largely stood without confirmation.

11 Acknowledgements

This work was performed for the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored in part by the JPL REE program, the Defense Advanced Research Projects Agency (DARPA) and the National Security Agency (NSA) through an agreement with the National Aeronautics and Space Administration.

12 References

References

[1] Brockman, Jay B., and Peter M. Kogge, *The Case for PIM*, Notre Dame CSE TR-9707, Jan. 10, 1997.

[2] Cmelik, B. *The SHADE simulator*, Sun-Labs Technical Report 1993

[3] Fromm, Richard, ET AL., *The Energy Efficiency of IRAM Architectures*, In Conference Proceedings, 1997 International Symposium on Computer Architecture (Denver, Colorado, June 1997), pp. 327-337.

[4] Kogge, Peter M., *EXECUBE- A New Architecture for Scalable MPPs*, 1994 International Conference on Parallel Processing.

[5] Nanda, Kartik, *SRAM Energy Modeleing* , Notre Dame CSE Technical Report, June 1998.

[6] Hennessey, John, and David Patterson, *Computer Architecture: A Quantitative Approach, Second Edition*, Morgan Kaufmann Publishers, 1996

[7] Saulsbury, Ashley, ET AL., *Missing the Memory Wall: The Case for Processor/Memory Integration*, In Proceedings of the 23rd Annual International Symposium on Computer Architecture (Philadelphia, Pennsylvania, May, 1996), pp. 90-101

[8] *http://notes.sematech.org/97melec.htm*

[9] *http://www.chips.ibm.com/products/embedded/chips/401core.html*

[10] *http:www.chips.ibm.com/services/foundry/technology/cmos7ld.html*

[11] *http:www.mitsubishichips.com/products/mcu/m32rd/m32rd.htm*

[12] *http:www.usa.samsungsemi.com/new/dramasic.htm*

Innovative VLIS Techniques for Power Reduction

Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System

Trevor Pering, Tom Burd, and Robert Brodersen
University of California Berkeley, Electronics Research Laboratory
{pering, burd, rb}@eecs.berkeley.edu
http://infopad.eecs.berkeley.edu/~pering/lpsw

Abstract

This paper describes the design of a low-power microprocessor system that can run between 8Mhz at 1.1V and 100MHz at 3.3V. The ramifications of Dynamic Voltage Scaling, which allows the processor to dynamically alter its operating voltage at run-time, will be presented along with a description of the system design and an approach to benchmarking. In addition, a more in-depth discussion of the cache memory system will be given.

1. Introduction

Our design goal is the implementation of a low-power microprocessor for embedded systems. It is estimated that the processor will consume 1.8mW at 1.1V/8MHz and 220mW at 3.3V/100MHz using a 0.6 µm CMOS process. This paper discusses the system design, cache optimization, and the processor's Dynamic Voltage Scaling (DVS) ability.

In CMOS design, the energy-per-operation is given by the equation

$$E_{op} \propto CV^2$$

where C is the switched capacitance and V is the operating voltage [2]. To minimize E_{op}, we use aggressive low-power design techniques to reduce C and DVS to optimize V.

Our system design, which addresses the complete microprocessor system and not just the processor core, is presented in Section 2. Our benchmark suite, which is designed for a DVS embedded system, is presented in Section 3. Section 4 discusses the issues involved with the implementation of DVS, while Section 5 presents an in-depth discussion of our cache design.

The basic goal of DVS is to quickly (~10µs) adjust the processor's operating voltage at run-time to the minimum level of performance required by the application. By continually adapting to the varying performance demands of the application energy efficiency is maximized.

The main difference between our design and that of the StrongARM is the power/performance target: our system targets ultra-low power consumption with moderate performance while the StrongARM targets moderate power consumption with high performance. Our processor core is based on the ARM8 architecture [1], which is virtually identical to that of the StrongARM. The similarities and differences between the two designs are highlighted throughout this paper.

2. System Overview

To effectively optimize system energy, it is necessary to consider all of the critical components: there is little benefit in optimizing the microprocessor core if other required elements dominate the energy consumption. For this reason, we have included the microprocessor core, data cache, processor bus, and external SRAM in our design, as seen in Figure 1. The energy consumed by the I/O system (not shown) is completely application and device dependent and is therefore beyond the scope of our work. The expected power distribution of our system is given in Figure 2.

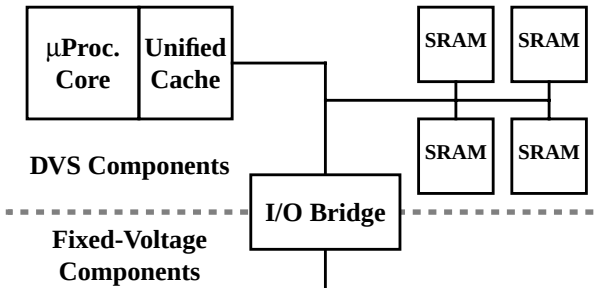


Figure 1: System Block Diagram

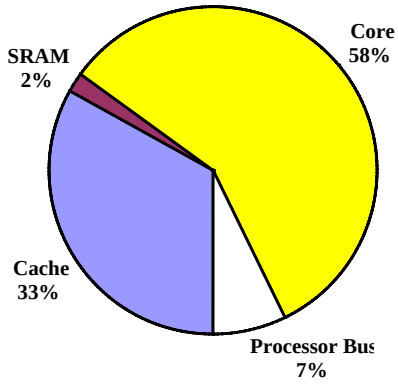


Figure 2: System Energy Breakdown

To reduce the energy consumption of the memory system, we use a highly optimized SRAM design [3] which is 32 data-bits wide, requiring only one device be

activated for each access. Schemes that use multiple narrow-width SRAMs require multiple devices to be activated for each access, resulting in a significant increases in the energy consumption. To alleviate the high pin count problem of 32-bit memory devices, we multiplex the data address onto the same bit-lines as the data words.

We use a custom designed high-efficiency non-linear switching voltage regulator [14] to generate dynamic supply voltages between 1.1V and 3.3V. An efficient regulator is crucial to an efficient system because all energy consumed is channeled through the regulator. When switching from 3.3V to 1.1V, a linear regulator would only realize a 3x energy savings, instead of the 12x reduction afforded by our design.

The threshold voltage (V_t) significantly effects the energy and performance of a CMOS circuit. Our design uses a V_t of 0.8V to achieve a balance between performance and energy consumption. The StrongARM [13], for comparison, uses a V_t of 0.35V, which increases performance at the expense of increased static power consumption. When idle, the StrongARM is reported to consume 20mW, which is the predicted power consumption of our processor *when running at 20MHz*. When idle, we estimate our processor will consume 200µw, an order of magnitude improvement.

3. Benchmarks

Our benchmark suite targets PDAs and embedded applications. Benchmark suites such as SPEC95 are not appropriate for our uses because they are batch-oriented and target high-performance workstations. DVS evaluation requires the benchmarking of workload idle characteristics, which is not possible with batch-oriented benchmarks. Additionally, our target device has on the order of 1MB of memory and lacks much of the system support required by heavy-weight benchmarks; running SPEC95 on our target device would simply be impractical.

We feel the following six benchmarks are needed to adequately represent the range of workloads found in embedded systems:

- AUDIO Decryption
- MPEG Decoding
- User Interfaces
- Java Interpreter
- Web Browser
- Graphics Primitive Rendering

As of this writing, we have implemented the first three of these and their characteristics are summarized in Table 3. “Idle Time” represents the portion of system idle time, used by DVS algorithms. The “Bus Activity” column reports the fraction of active cycles on the external processor bus, an important metric when optimizing the cache system. The cache architecture used to generate Table 3 is discussed in Section 5.

As an example, Figure 4 shows an *event impulse graph* [13], which is used to help characterize programs for DVS analysis. Each impulse represents one MPEG frame and indicates the amount of work necessary to process that frame. For this example, there is a fixed frame-rate which can be used to calculate the optimal processor speed for each frame, assuming only one outstanding frame at any given time.

Benchmark	Miss Rate	Idle Time	Bus Activity
AUDIO	0.23%	67%	0.35%
MPEG	1.7%	22%	14%
UI	0.62%	95%	0.52%

Table 3: Benchmark Characterization

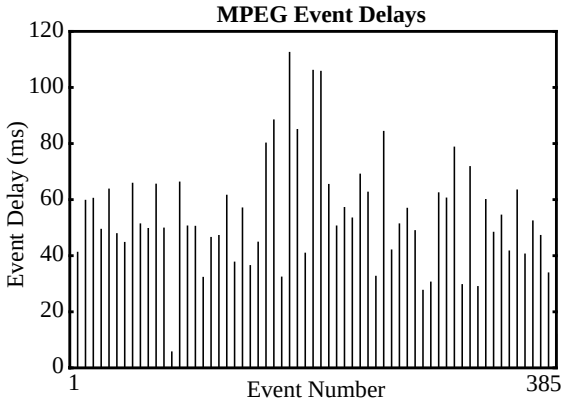


Figure 4: MPEG Event Impulse Graph

4. Dynamic Voltage Scaling

Our processor has the ability, termed Dynamic Voltage Scaling (DVS), to alter it’s execution voltage while in operation. This ability allows the processor to operate at the optimal energy/efficiency point and realize significant energy savings, which can be as much as 80% for some applications [13]. This section discusses DVS design considerations and explains how it affects architectural performance evaluations.

DVS combines two equations of sub-micron CMOS design [2]:

$$E_{op} \propto V^2 \text{ and } f_{max} \propto \frac{V - V_t}{V}$$

where E_{op} is the energy-per-operation, f_{max} is the maximum clock frequency, and V is the operating voltage. To minimize the energy consumed by a given task, we can reduce V, affecting a reduction in E_{op}. A reduction in V, as shown in the second equation, results in a corresponding decrease in f_{max}. A simple example of these effects is given below.

Reducing f_{clk}, the actual processor clock used, without reducing V does not reduce the energy consumed by a processor for a given task. The

StrongARM 1100, for example, allows f_{clk} to be dynamically altered during operation [16], affecting a linear reduction in the power consumed. However, the change in f_{clk} also causes a linear increase in task runtime, causing the energy-per-task to remain constant. Our system always runs with $f_{clk} = f_{max}$, which minimizes the energy consumed by a task.

From a software perspective, we have abstracted away the voltage parameter and specify the operating point in terms of f_{max}. The actual voltage used is determined by a feedback loop driven by a simple ring oscillator. The primary reason for this design was ease of the hardware implementation; fortunately, it also presents the most useful software interface.

Our system applies one dynamic voltage to the entire system to realize savings from all components. It would be possible, however, to use multiple independent supply voltages to independently meet subsystem performance requirements. This was not attempted in our design. To interface with DVS-incompatible external components we use custom designed level-converting circuits.

The implementation of DVS requires the application of *voltage scheduling* algorithms. These algorithms, discussed in Section 4.2, monitor the current and expected state of the system to determine the optimal operating voltage (frequency).

4.1 Energy/Performance Evaluation Under DVS

DVS can affect the way we analyze architectural trade-offs. As an example, we explore the interaction between DVS and the ARM Thumb [4] instruction set. We apply Thumb to the MPEG benchmark from Section 3 and analyze the energy consumed. This example assumes a 32-bit memory system, which is a valid assumption for high-performance systems but not necessarily for all embedded designs.

The MPEG benchmark is 22% idle when running at 100 MHz using the 32-bit ARM instruction set. DVS allows us to minimize the operating voltage to fill unnecessary idle-time. Using a first-order approximation, this would reduce the energy consumed by 40% and slow down the processor clock to the point at which idle time is zero. From this starting point, we consider the application of the Thumb instruction set to this benchmark.

For typical programs, the 16-bit Thumb instruction-set is 30% more dense than it's 32-bit counterpart, reducing the energy consumed in the cache and memory hierarchy. However, due to reduced functionality, the number of instructions executed increases by roughly 18%, increasing the energy dissipated in the processor core as well as the task execution time.

This example will teach two important lessons. First, an increase in task delay directly relates to an increase in energy: DVS exposes the trade-off between energy and performance. Second, an increase in delay affects the *entire* system (core and cache), not just one fragment: it is vital that the associated increase in the energy-per-operation is applied to the entire system.

Figure 5 presents six metrics crossed with three configurations running the MPEG benchmark. The three configurations are:

- **Base:** 78 MHz using 32-bit instructions.
- **Thumb:** 78 MHz using Thumb instructions.
- **Adjusted:** 92 MHz using Thumb instructions.

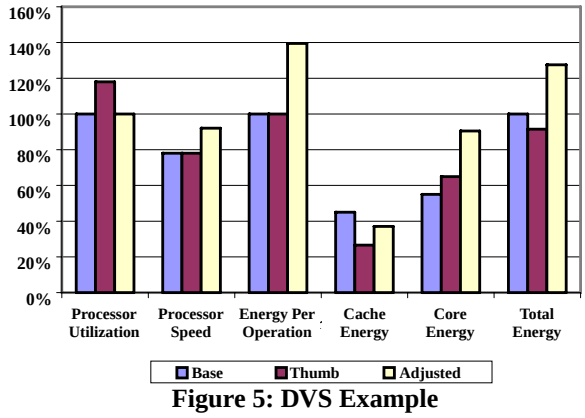


Figure 5: DVS Example

The 'Base' configuration represents the MPEG benchmark running as 32-bit code, as discussed above. 'Thumb' illustrates the intermediate effects of the 16-bit Thumb architecture without increasing the clock speed. The energy consumed in the cache (see Figure 5) decreases due to the decreased memory bandwidth caused by the smaller code size. The energy of the core, however, rises slightly due to the increased number of instructions processed. Overall, the energy decreases by approximately 10%.

The delay increase caused by the expanded instruction stream pushes the processor utilization over 100%. Because of this, the MPEG application will not be able to process its video frames fast enough. The 'Adjusted' configuration represents the increase in processor speed required to maintain performance. This change in clock frequency necessitates an increase in voltage which raises the energy-per-operation of the entire system. As can be seen from the 'Total Energy' columns, the energy savings are no longer realized: the 16-bit architecture increases overall energy consumption.

Although not energy-efficient in all situations, the Thumb instruction set may be efficient for some tasks due to the non-linearity of voltage-scaling. If the base system were initially running at a very low voltage, for example, the increase in processor speed necessary would not dramatically increase the energy-per-operation. The savings due to the reduced code-size, therefore, would affect an overall *decrease* in system energy.

4.2 Voltage Scheduling

To effectively control DVS, a *voltage scheduler* is used to dynamically adjust the processor speed and volt-

age at run-time. Voltage scheduling significantly complicates the scheduling task since it allows optimization of the processor clock rate. Voltage schedulers analyze the current and past state of the system in order to predict the future workload of the processor.

Interval-based voltage schedulers are simple techniques that periodically analyze system utilization at a global level: no direct knowledge of individual threads or programs is needed. If the preceding time interval was greater than 50% active, for example, the algorithm might increase the processors speed and voltage for the next time interval. [5][13][17] analyze the effectiveness of this scheduling technique across a variety of workloads. Interval-based scheduling has the advantage of being easy to implement, but it often has the difficulty of incorrectly predicting future workloads.

More recently, investigation has begun into thread-based voltage schedulers, which require knowledge of individual thread deadlines and computation required [7][12]. Given such information, thread-based schedulers can calculate the optimal speed and voltage setting, resulting in minimized energy consumption. A sample deadline-based *voltage scheduling graph* is given in Figure 6; S_x and D_x represent task start-time and deadline, respectively, while the graph area, C_x, represents computational resources required.

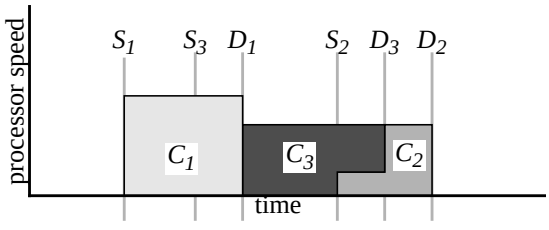


Figure 6: The Voltage Scheduling Graph

4.3 Circuit Level Considerations

At the circuit level, there are two types of components in our design adversely affected by DVS: complex logic gates and memory sense-amps. Complex logic gates, such as 8-input NAND gates, are implemented by a CMOS transistor chain which will have a different relative delay if the voltage is varied. Additionally, memory sense-amps are sensitive to voltage variations because of their analog nature, which is necessary to detect the small voltage fluctuations of the memory cells.

To the largest extent possible, these voltage sensitive circuits are avoided; however, in some situations, such as in the cache CAM design described below, it is better to redesign the required components with increased tolerance. Redesigns of these components will often be less efficient or slower than the original version when running at a fixed voltage. We estimate an increase in the average energy/instruction of the microprocessor on the order of 10%, which is justified by the overall savings afforded by DVS.

5. Cache Design

This section describes the design of our cache system, which is a 16kB unified 32-way set-associative read-allocate write-back cache with a 32-byte line size. The cache is an important component to optimize since it consumes roughly 33% of the system power and is central to system performance. Our primary design goal was to optimize for low-power while maintaining performance; our cache analysis is based on layout capacitance estimates and aggregated benchmark statistics.

Our 16kB cache is divided into 16 individual 1kB blocks. The 1kB block-size was chosen to achieve a balance between block access energy and global routing energy. Increasing the block-size would decrease the capacitance of the global routing but it would also increase the energy-per-access of the individual blocks.

Our cache geometry is very similar to that of the StrongARM, which has a split 16kB/16kB instruction/data cache. Other features, namely the 32-way associative CAM array, are similar. In the StrongARM design, the caches consume approximately 47% of the system power [13].

5.1 Basic Cache Structure

We have discovered that a CAM based cache design (our implementation is given in Figure 7) is more efficient than a traditional set-associative organization (Figure 8) in terms of both power and performance. The fundamental drawback with the traditional design is that the energy per access scales linearly with the associativity: multiple tags and data must be fetched simultaneously to maintain cycle time. A direct-mapped cache, therefore, would be extremely energy efficient; its performance, however, would be unacceptable. We estimate that the energy of our 32-way set-associative design is comparable to that of a 2-way set-associative traditional design.

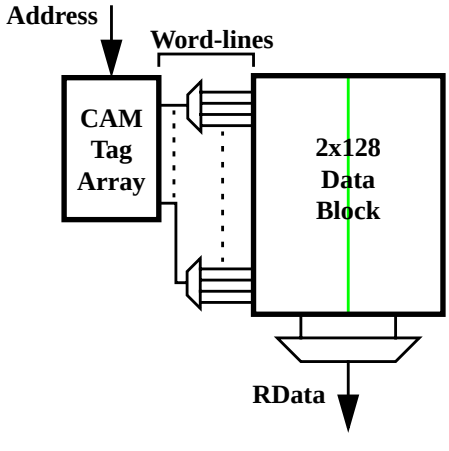


Figure 7: Implemented CAM Cache Design

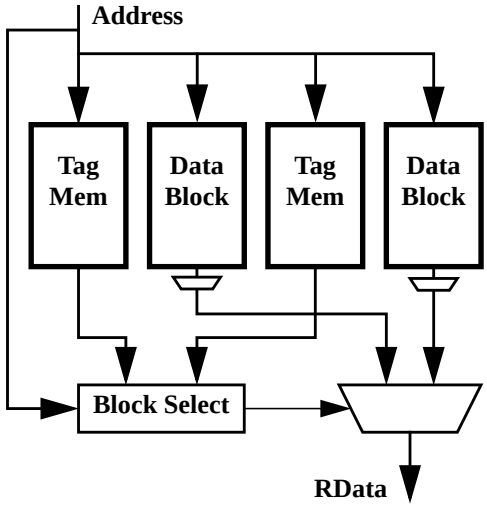


Figure 8: Traditional Set-Associative Cache Design

Our design has been modified from a vanilla CAM in two major ways:

- **Narrow memory bank:** The fundamental SRAM data block for our design is organized as a 2-word x 128-row block, instead of a 8-word x 32-row block.
- **Inhibited tag checks:** Back-to-back accesses mapping to the same cache line do not trigger multiple tag checks.

The 2-word by 128-row block organization for our cache data was chosen primarily because a large block width would increase the energy-per-access to the data block. A block width of 8 words, for example, would effectively entail fetching 8 words per access, which is wasteful since only one or two of these words would be used. The narrow block width unfortunately causes an irregular physical layout, increasing total cache area; however, we chose this design as energy was our primary concern.

There are two natural lower-bounds on the block width. First, the physical implementation of the SRAM block has an inherent minimum width of 2-words [3]. Second, the ARM8 architecture has the capability for double-bandwidth instruction fetches and data reads [1], which lends itself to a 2-word per access implementation.

Unnecessary tag checks, which would waste energy, are inhibited for temporally sequential accesses that map to the same cache line. Using the sequential-access signal provided by the processor core and a small number of access bits, this condition can be detected without a full address comparison. Our simulations indicate that about 46% of the tag checks are avoided with a 8-word cache line size, aggregated across both instruction and data accesses. For the individual instruction and data streams, 61% and 8% of tag checks are prevented, respectively.

5.2 Cache Policies and Geometry

Cache energy has a roughly logarithmic relationship with respect to its overall size, due to selective block enabling: a 16kB cache consumes little more energy than an 8kB cache. Our fundamental cache size constraint was die cost, which is determined primarily by cache area. Benchmark simulations indicate that a 16kB unified cache is sufficient; we felt the increased cost of a 32kB cache was not justified. We chose a unified cache because it is most compatible with the ARM8 architecture.

The cache line size has a wide-ranging impact on energy efficiency; our analysis (Figure 9) indicates that an 8-word line size is optimal for our workload. Given the 1kB block size, our associativity is inversely proportional to the line size: an 8-word line yields 32-way associativity (1kB / 8-words = 32-way). The energy of a CAM tag access is roughly linear with associativity. Also, smaller cache line sizes generate less external bus traffic, consuming less energy. The energy of the data memory is practically constant, although there are slight variations caused by updates due to cache misses.

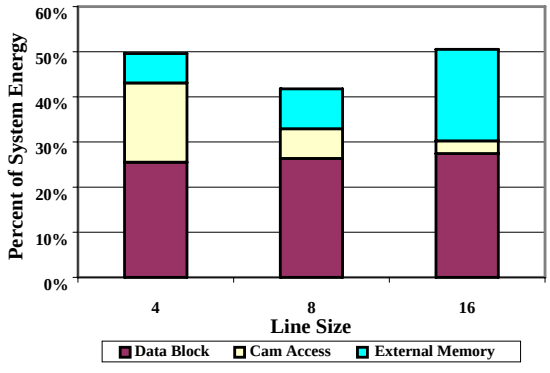


Figure 9: Line-Size Energy Breakdown

We implement a write-back cache to minimize external bus traffic. Our simulations indicate that a write-through cache would increase the external bus traffic by approximately 4x, increasing the energy of the entire system by 27%. We found no observable performance difference between the two policies.

Our simulations find no significant evidence either for or against read-allocate in terms of energy or performance; we implement read-allocate to simplify the internal implementation. Similarly, we find that round-robin replacement performance is comparable to that of both LRU and random replacement, due to the large associativity.

5.3 Related Work

Most low-power cache literature [6][9][15][8] suggests improvements to the standard set-associative cache model of Figure 8. The architectural improvements proposed center around the concepts of sub-banking and row-buffering. Sub-banking retrieves only the required portion of a cache line, saving energy by not extraneously fetching data. Row-buffering fetches and saves an entire cache line to avoid future unnecessary

tag comparisons.

Our CAM-based cache design indirectly implements the concepts of sub-banking and row-buffering. The 2-word block size of our memory bank is similar to 2-word sub-banking. Tag checks inhibition is similar to row-buffering: only one tag-check is required for each cache-line access.

[8] presents a technique for reducing the energy of CAM-based TLBs by restricting the effective associativity of the parallel tag compare and modifying the internal CAM block. Due to time constraints, these modifications were not considered for our design.

6. Conclusion

This paper describes the implementation of a low-power Dynamic Voltage Scaling (DVS) microprocessor. Our analysis encompasses the entire microprocessor system, including the memory hierarchy and processor core. We use a custom benchmark suite appropriate for our target application: a portable embedded system.

Dynamic Voltage Scaling allows our processor to operate at maximal efficiency without limiting peak performance. Understanding the fluid relationship between energy and performance is crucial when making architectural design decisions. A new class of algorithms, termed voltage schedulers, are required to effectively control DVS.

A description of our cache design was given which presents the architectural and circuit trade-offs with energy and performance for our application domain. For minimized energy consumption, we found that a CAM-based cache design is more energy efficient than a traditional set-associative configuration.

Acknowledgments

This work was funded by DARPA and made possible by cooperation with Advanced RISC Machines Ltd (ARM). The authors would like to thank Eric Anderson, Kim Keeton, and Tom Truman for their proofreading help.

7. References

[1] *ARM 8 Data-Sheet*, Document Number ARM DDI0080C, Advanced RISC Machines Ltd, July 1996.

[2] T. Burd and R. Brodersen, “Energy Efficient CMOS Microprocessor Design,” *Proc. 28th Hawaii Int’l Conf. on System Sciences*, 1995.

[3] A. Chandrakasan, A. Burstein, R. Brodersen, “A Low-Power Chipset for a Portable Multimedia I/O Terminal,” *IEEE Journal of Solid-State Circuits, Vol. 29, No. 12, December 1994.*

[4] L. Goudge, S. Segars, “Thumb: reducing the cost of 32-bit RISC performance in portable and consumer applications,” *Forty-First IEEE Computer Society International Conference, 1996.*

[5] K. Govil, E. Chan, H. Wasserman, “Comparing algorithms for dynamic speed-setting of a low-power CPU,” *Proc. of the First Annual Int’l Conf. on Mobile Computing and Networking, 1995*.

[6] P. Hicks, M. Walnock, and R. Owens, “Analysis of Power Consumption in Memory Hierarchies,” *Proc. 1997 Int’l Symp. on Low Power Electronics and Design.*

[7] T. Ishihara, H. Yasuura, “Voltage Scheduling Problem for Dynamically Variable Voltage Processors,” *Proc. 1998 Int’l Symp. on Low Power Electronics and Design.*

[8] T. Juan, T. Lang, J. Navarro, “Reducing TLB Power Requirements,” *Proc. 1997 Int’l Symp. on Low Power Electronics and Design*.

[9] M. Kamble and K. Ghose, “Analytical Energy Dissipation Models For Low Power Caches,” *Proc. 1997 Int’l Symp. on Low Power Electronics and Design.*

[10] T. Kuroda, et. al., “Variable Supply-Voltage Scheme for Low-Power High-Speed CMOS Digital Design,” *IEEE Journal of Solid-State Circuits, Vol. 33, No. 3, March 1998.*

[11] J. Montanaro, et. al., “A 160Mhz 32b 0.5W CMOS RISC Microprocessor,” *1996 IEEE Int’l Solid-State Circuits Conf.*

[12] T. Pering and R. Brodersen, “Energy Efficient Voltage Scheduling for Real-Time Operating Systems,” *4th IEEE Real-Time Technology and Applications Symposium, 1998, Works In Progress Session*.

[13] T. Pering, T. Burd, and R. W. Brodersen, “The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms,” *Proc. 1998 Int’l Symp. on Low Power Electronics Design*.

[14] A. Stratakos, S. Sanders, R. Brodersen, “A low-voltage CMOS DC-DC converter for a portable battery-operated system,” *25th Annual IEEE Power Electronics Specialists Conference, 1994.*

[15] C. Su and A. Despain, “Cache Designs for Energy Efficiency,” *Proc. 28th Hawaii Int’l Conf. on System Sciences*, 1995.

[16] M. Viredaz, “Itsy: An Open Platform for Pocket Computing,” presentation from Digital Equipment Corporation’s Western Research Laboratory.

[17] M. Weiser, “Some computer science issues in ubiquitous computing,” *Communications of the ACM*, Vol. 36, July 1993.

Transmission Line Clock Driver

Matthew E. Becker
Artificial Intelligence Laboratory
Massachusetts Institute Of Technology
Cambridge, MA, USA
mbecker@mit.edu

Thomas F. Knight, Jr.
Artificial Intelligence Laboratory
Massachusetts Institute Of Technology
Cambridge, MA, USA
tk@ai.mit.edu

Abstract *Clock distribution is one of the major issues in digital design. Engineers want to distribute a square-wave with low skew and fast transition times across very wide chips. And they want to do so, wasting as little power as possible. Common micro-processor chips, like the Alpha, throw 40% of their power into distributing a square-wave across a 1.5 cm die. This paper describes a new clock distribution technique utilizing resonant transmission lines that not only reduces clock skew and transition times, but also reduces power consumption by up to an order of magnitude over standard clock drivers.*

1. Motivation

Over the years a number of trends have increased the amount of power consumption of clock drivers. First, the clock capacitance has steadily increased as the die size and the gate capacitances have become larger. Second, the clock frequency is also increasing. Clock drivers on current micro-processor chips must drive a 4 nF load at 500 MHz. This burns almost 20 W or 40% of the overall chip power.[i]

Researchers have explored many ways of reducing power consumption. Arguably the simplest way has involved reducing the power supply voltage. This provides a square reduction in power, but adversely affects circuit speeds and increases sub-threshold leakage. Other methods temporarily stop driving unused sections of the chip. Unfortunately, this complicates system design and increases the amount of clock skew between chip sections.

On a different topic, a relatively new approach to coordinate different chips at a system level involves using a resonating cavity. Vernon Chi at the University of North Carolina resonates a uniform transmission line with a single sinusoid.[ii] Because the entire transmission line crosses zero simultaneously, this structure can synchronize individual chips throughout the system. Unfortunately, a sinusoidal waveform cannot be used to drive individual clock loads directly. Its transition times are just too long and the large clock capacitance would interfere with transmission line operation.

This work takes the resonant approach the next step. I will describe a resonating transmission line which can drive the large clock capacitance directly. This requires resonating a square-wave instead of a single sinusoid. Unfortunately, a uniform transmission line driving a large capacitive load will not support the odd harmonics required to produce a square-wave. Instead the resonant line must be tuned to support each of the harmonics.

There are two major benefits of using a tuned transmission line to drive the clock load directly. First, the half crossing still occurs simultaneously across the central section of the transmission line. This translates to virtually no skew across a single die. Second, the power to drive the on-chip capacitance, comes from the transmission line instead of the power supplies. Therefore, the power consumption falls to the amount burnt in the parasitic resistance of the transmission line. This results in a 10x reduction in power consumption.

The next section describes a standard clock driver used in the micro-processor industry. Section 3 delves into our proposed driver technique using resonant transmission lines. Then, the paper reviews some experimental work verifying this technique's viability.

2. Standard Clock Driver

A standard clock distribution structure appears in Figure 1. It is relatively simple. It includes a clock generator, a buffer and a distribution network. We have drawn the clock lines as transmission lines. Since the series resistance dominates over the inductance, the clock loads and lines must be modeled as distributed RC lines.

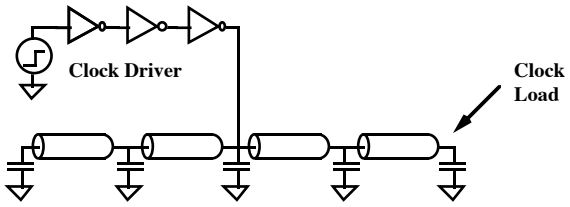


Figure 1. Standard Clock System For Large Chips

Power to fill up the clock capacitance and pre-driver capacitance comes from the power supply. This power is dumped into the gnd supply when discharging the capacitances explaining why the clock uses 40% of chip power.

3. Transmission Line Clock Driver

Our technique shown in Figure 2, simply adds an external transmission line. We recover power by charging and discharging the final clock load not through

the clock driver but with the transmission line. This also means that the clock buffers can be smaller resulting in less pre-driver power.

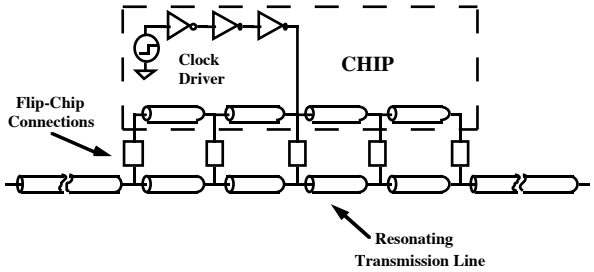


Figure 2. New Transmission Line Clock Driver

Let me walk through a brief description of how the technique works. The central driver forces a square wave into the chip and the external transmission line. Because the driver is smaller, it cannot initially drive the system to vdd. However, a reduced height pulse flows down the length of the transmission line. The pulse travels to the open termination of the transmission line and reflects back towards the chip. The line length is such that the pulse in the transmission line will reach the driver exactly when it drives again. The result is an increase in the pulse height. Eventually the transmission line will be resonating a full clock pulse at a given clock frequency.

Taps using flip-chip technology strap the internal clock line to the external transmission line. A pin structure like flip-chip is useful because of its low inductance, short pin lengths and the even distribution of pins across the chip.

3.1 Tuning a Line Using Traps

Unfortunately, the simple structure presented in the previous section works only with an ideal transmission line. When an actual square-wave travels along a lossy transmission line, the different components of the waveform travel at different velocities based on their frequency. This results from the parasitic resistance which varies linearly with the frequency of the waveform.

So in the actual transmission line one must introduce impedance variations along its length to reinforce the desired waveform. For example, one could include a version of the traps that are used in RF design.[iii]

When a waveform reaches a trap, the low frequency components pass through, while the high frequency part reflects. The length of the trap affects what will pass and what won't. By changing the location of the traps our transmission line can be tuned for each of the desired frequencies.

Figure 3 shows a simple model of a transmission line with a single trap. High frequencies are trapped on the left. By varying L_{HIGH} one can change the resonant frequency of the left section. The lower frequency component of the signal can pass through the trap. Therefore, the combined length of L_{HIGH} and L_{LOW} set the lower resonant frequency.

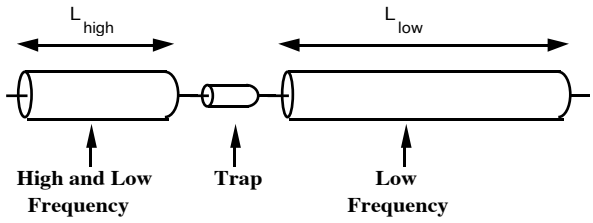


Figure 3. Traps Used to Tune Resonant Frequencies

3.2 Clock Skew and Rise Times

One of the very exciting prospects of this design regards clock skew. Ideally the **entire** resonating transmission line crosses through 1/2 Vdd at the **same** time. In real cases, the clock load, interconnect delay and dispersion within the transmission line introduce a minimal amount of skew, less than 10 ps. This performance figure is almost an order of magnitude better than what is achieved in the Alpha!

Unfortunately, though the center crossing is relatively well controlled, the differences in rise and fall times across the chip can be serious. Consider the situation in the frequency domain as shown in Figure 4. In the frequency domain, our clock waveform decomposes into a sum of sinusoids. The primary components are the 1st, 3rd and 5th harmonics. As we move farther away from the center of the chip, the amplitudes of the third and the fifth harmonic fall off quickly. This degrades the rise and fall times at the edge of the chip.

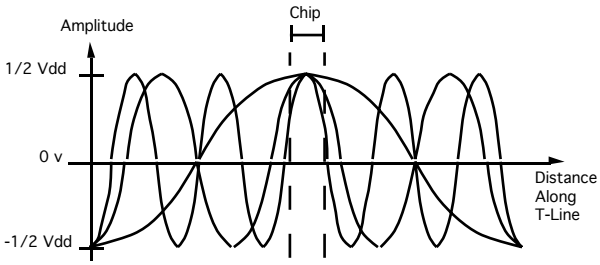


Figure 4. Frequency Domain View of Rise Time Problem

For a 2 cm chip this technique provides reasonable rise times up to 1 GHz. Beyond this frequency, the reduced amplitude of the higher harmonics does not provide adequate rise times at the edge of the chip. One correction would be to shift the tap points, flip-chip connections, towards the center of the chip. Unfortunately, this introduces extra skew from the new tap points to the edge of the chip. Another correction would be to drive each tap point with independent transmission lines, but this would involve a more complicated structure.

4. Large Scale Mockup

In order to test this driver technique quickly and cheaply, we have built a low frequency version. This allows standard off the shelf parts to be used while still testing all the basic principles.

The large scale mockup has proven the viability of the transmission line clock driver technique. At 20 Mhz the driver with uniform impedance reduces power consumption by a factor of 5.7 relative to a standard clock driver. Using a more complex transmission line saves a factor of 10 over the standard driver. Further, the quality of waveforms is significantly better for the complex transmission line.

The next two sections describes the overall structure of our mockup and the testing method. Then the paper presents the results from each test case: the standard clock driver, the driver with a uniform transmission line and the driver with a complex transmission line.

4.1 Structure

Figure 5 shows the basic schematic of the large scale mockup. It can be broken into two distinct parts: the external transmission line and on-chip clock structure.

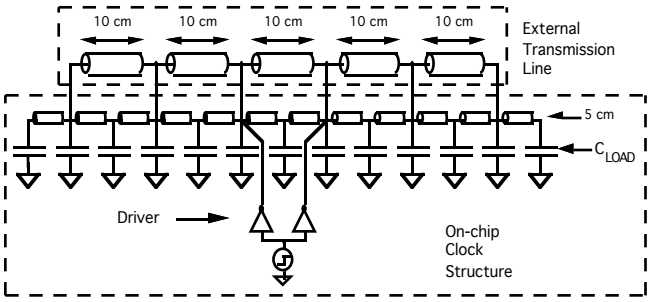


Figure 5. Standard Clock Driver

All driver configurations share the **on-chip clock** structure. As shown at the bottom, it contains a driver involving a pulse generator with variable frequency driving a pair of inverters. In the transmission line drivers, the frequency is set to the resonant frequency. For the standard driver, the impedance of each inverter is approximately 2 Ω. In order to test a configuration with a different inverter width, W, series resistance is added between the inverters and the load. Lumped elements, C_{LOAD}, model the on-chip clock capacitance. The total load is 2.2 nF. The small, 5 cm wires represent the on-chip clock wires.

All configurations including the standard driver share the central section of the **external transmission line**. There are two reasons for this. First, the standard driver performed better with the external transmission line and so made a harder standard to beat. Second, for a real chip the clock impedance would probably be lower than the 5 cm segments by themselves, but definitely not lower than the impedance with the external transmission lines. The central section of the external transmission line includes the five 10 cm segments that match the length of the entire chip, 50 cm. This length is derived from the frequency scale up factor of 500 MHz : 20 MHz, and the width of a standard chip, 2 cm. The sides of the external transmission line change extensively between configurations. As seen in Figure 5, the standard driver has no sides at all. The next configuration in Figure 7 uses a uniform transmission line that is 4.6 m long. Figure 9 shows the case with a complex transmission line.

4.2 Testing Method

This section lists the steps involved in testing and evaluating each transmission line driver. It then describes how to normalize power relative to frequency, self-loading power and pre-driver power.

The first three steps are performed on the standard driver:

1) Measure the rise time, T_{STD}, for a given inverter width, W_{STD}
2) Measure the power, P_{STD}, at W_{STD}
3) Normalize P_{STD}

Then the following steps are performed on each transmission line driver:

4) Measure the rise time, T, for a given inverter width, W
5) Change W until $T = T_{STD}$
6) Measure the power, P, at W
7) Normalize P
8) Compare the normalized powers

The first three steps involve the standard clock driver, and provide a base line against which to compare the transmission line drivers. In the final 5 steps, the driver width acts as the independent variable. As the width increases the signal rise times improve, while the power increases. Please recall that changing the width is simulated by adding extra series resistance. Step 5 sets the driver width such that the transmissions line driver provides the same quality signal, based on rise time, as the standard driver. At this size one can compare the normalized powers.

Power must be normalized for three different effects. First, in order to compare powers obtained at different frequencies, the power is normalized relative to a frequency of 20 MHz. As seen in Equation 1, the three frequency terms, (20 MHz/F), (20 MHz/F_{SELF}) and (20 MHz/F_{STD}) normalize each power. F, F_{SELF} and F_{STD} represents the frequency for the current configuration, for the self-loading test case and for the standard driver, respectively.

Second, in order to compare configurations, one needs to include the self-loading power. Self-loading power is the power required to fill the source and drain capacitance of the driver. Because all the tests use the same inverter part, they all consume the same amount of self-loading

power, even in the cases with a smaller, simulated inverter. Therefore, one must normalize the measured power to correct for the reduced self-loading when simulating a smaller inverter. This correction appears in the second term of Equation 1. $(1-W/W_{STD})$ represents the ratio of the extra width to the total width. Multiplying this by the total self-loading, leaves that power related to just the unused width. This can then be subtracted from the total power.

Finally, one needs to include pre-driver power. Pre-driver power represents the amount of power consumed in driving the input to the final inverter stage. Please note that in all of the configurations, this power comes from the pulse generator not the power supply. Therefore, the pre-driver power is estimated by dividing the measured power in the standard case by an inverter scale up factor of 4. The third term of Equation 1provides this correction. Again, in order to compare pre-driver powers with different inverter widths, the pre-driver power has to be normalized relative to the inverter width. Therefore, we multiply the standard pre-driver power by the by the new width, W/W_{STD}.

$$P_{NORM} = P_{TEST}\left(\frac{20\,MHz}{F_{TEST}}\right) - P_{SELF}\left(1-\frac{W}{W_{STD}}\right)\left(\frac{20\,MHz}{F_{SELF}}\right) + \frac{1}{4}P_{STD}\left(\frac{W}{W_{STD}}\right)\left(\frac{20\,MHz}{F_{STD}}\right)$$

Equation 1

4.3 Standard Clock Driver

This section describes a few test cases used to established a base line for evaluating later designs. These measured results are substituted into Equation 1. Table 1 shows the power consumption for a few important cases as wells as the rise times for the edge and center of the chip. The self-loading test simply involves disconnecting the clock load before power is measured.

	Center T	Edge T	P	P_{NORM}
Self @16.4 MHz			0.12 W	
Std @2 MHz	15.6 ns	11.6 ns	0.19 W	2.4 W
Std @16.4 MHz	13.3 ns	10 ns	1.5 W	2.3 W
Std @14 MHz	14.7 ns	11.4 ns	1.48 W	2.6 W

Table 1. Mock-up of Standard Clock Driver

The first test provides the self-loading power and frequency: P_{SELF}=0.12 W and F_{SELF}=16.4 Mhz. The second test provides the pre-driver power and frequency: P_{SELF}=0.19 W and F_{SELF}=2 Mhz. Once these values are substituted into Equation 1, things can be simplified down to Equation 2. As a check note that for two different frequencies, namely the third and fourth tests shown in Table 1, the normalized powers calculated by Equation 2 are reasonably close.

$$P_{NORM} = P_{TEST}\left(\frac{20\,MHz}{F_{TEST}}\right) - 0.15\,W + 0.62\,W\left(\frac{W}{W_{STD}}\right)$$

Equation 2

Figure 6 shows a plot of the last test case of the standard driver. This was run at 14 Mhz, a period of 71.5 ns. Signal C1 corresponds to the center of the chip and signal C2 corresponds to the edge. Despite the seemingly reasonable rise times shown in Table 1 and appearing on the right of Figure 6, note the poor quality of the waveform for this given width. The next section will show how the transmission line drivers have cleaned up the quality appreciably.

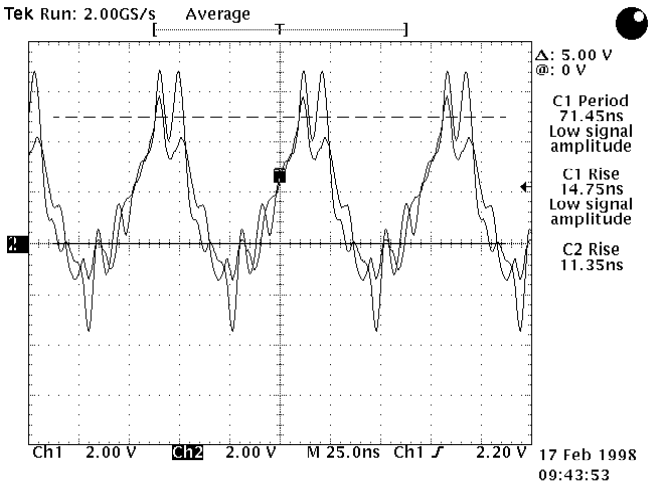


Figure 6. Clock Signals of the Standard Driver

4.4 Driver with Uniform Transmission Line

This section describes the transmission line clock driver with uniform impedance. Figure 7 shows its associated external transmission line. The uniform line resonated a 14 Mhz square wave. The impedance of the entire transmission line was fixed at approximately 10 Ω.

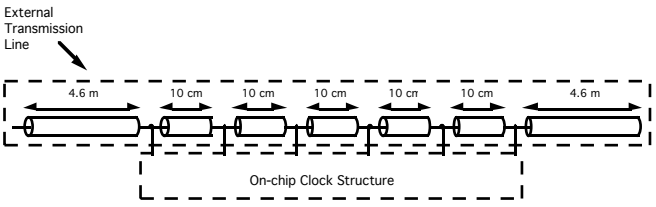


Figure 7. External Transmission Line with Uniform Impedance

Table 2 shows the power for many different driver widths, W. In order to calculate the normalized power, W/W_{STD} and P were substituted into Equation 2. Based on the rise time and overall quality of the waveform, the point where the transmission line driver matches the

standard clock driver appears when $W/W_{STD} = 0.07$. At this width, the transmission line clock driver consumes 0.42 W saving a factor of **5.7** over the standard clock driver.

W/W_{STD}	Center T	Edge T	P	P_{NORM}
0.04	14.1 ns	20.5 ns	0.32 W	0.33 W
0.06	14 ns	19 ns	0.35 W	0.39 W
0.07	10 ns	19 ns	0.37 W	0.42 W
0.15	6.6 ns	18.8 ns	0.38 W	0.49 W
0.33	4.6 ns	19.3 ns	0.40 W	0.62 W

Table 2. Mock-up of a Driver with a Uniform Transmission Line

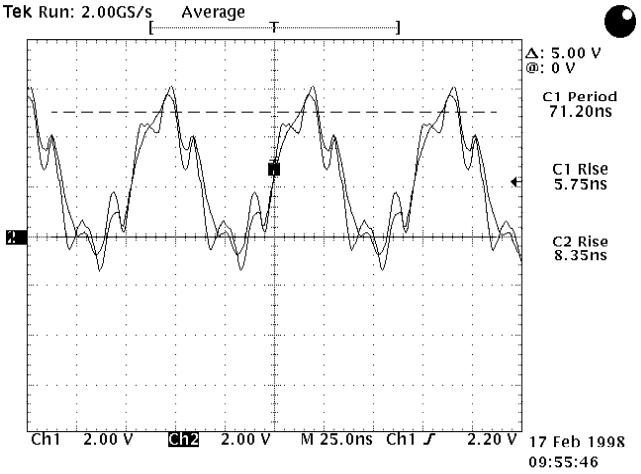


Figure 8. Clock Signal of a Driver with a Uniform Transmission Line

Figure 8 shows a plot of the uniform transmission line driver at 14 Mhz, a period of 71.5 ns. The width of the driver, W, is 0.15 times the width of the standard driver, W_{STD}. Signal C1 corresponds to the center of the chip and signal C2 corresponds to the edge. The quality of this waveform is quite superior to that found with the standard driver, even though the driver width and power consumed is significantly smaller.

4.5 Driver with Complex Transmission Line

This section describes the transmission line driver with a complex impedance. Figure 9 shows the external transmission line. The complex line reinforces the first and third harmonics of a 16.4 Mhz square wave. In order to tune for the third harmonic, a pair of 20 cm sections are added to the central section. Beyond this a 40 cm trap of higher impedance, 35 Ω, blocks the third harmonic. The outside sections, 1.4 m, tune the first harmonic.

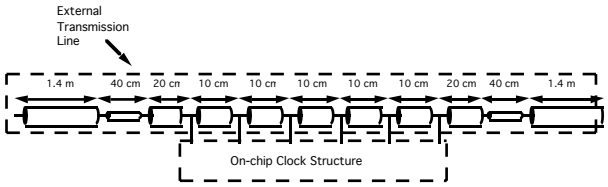


Figure 9. External Transmission Line with Complex Impedance

Table 3 shows the power consumption and rise times for many different driver widths. Based on the rise time and overall quality of the waveform, the point where the complex transmission line driver matches the standard clock driver appears when $W/W_{STD} = 0.06$. At this width the transmission line clock driver consumes 0.24 W saving a factor of **10.0** over the standard driver.

W/W_{STD}	Center T_R	Edge T_R	P_{MEAS}	P_{NORM}
0.04	19.9 ns	21.0 ns	0.27 W	0.20 W
0.06	5.6 ns	14.5 ns	0.29 W	0.24 W
0.07	5.0 ns	14.2 ns	0.3 W	0.26 W
0.15	3.5 ns	12.1 ns	0.33 W	0.35 W
0.33	3.2 ns	12.1 ns	0.33 W	0.46 W

Table 3. Mock-up of a Driver with a Complex Transmission Line

Figure 10 shows a plot of the complex transmission line driver at 16.4 Mhz, a period of 61.1 ns. The width of the driver, W, is set to 0.15 times the width of the standard driver, W_{STD}. Signal C1 corresponds to the center of the chip and signal C2 corresponds to the edge. The quality of this waveform is quite superior to that found with the previous drivers, even though the driver width and power consumed is significantly smaller.

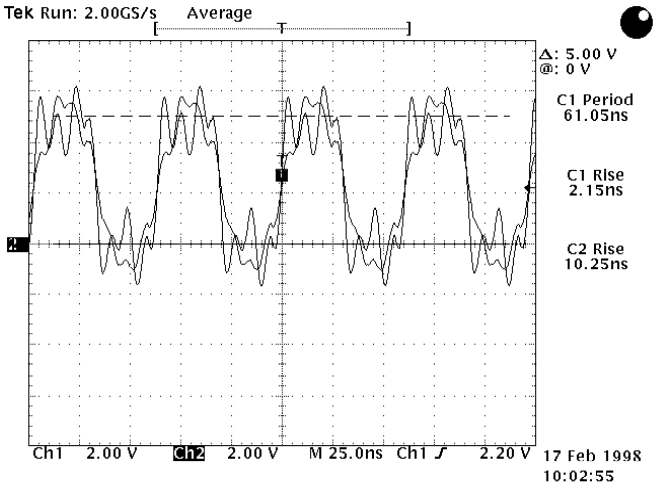


Figure 10. Clock Signal of a Driver with a Complex Transmission Line

5. Conclusions

Clock distribution represents an important aspect of digital system design. Current techniques have stalled in their effort to reduce clock skew, transition times and power consumption. A new clock distribution technique utilizing resonant transmission lines with a complex impedance solves this problem. It can match the waveform quality of a standard clock driver for a tenth of the driver width and a tenth of the power.

---

[i] Bowhill and Gronowski. *Practical Implementation Methods and Circuit Examples used on the ALPHA 21164*. VLSI Circuits Workshop. Digital Semiconductor

[ii] Chi, Vernon. *Salphasic Distribution of Timing Signals for the Synchronization of Physically Separated Entities*. US Patent #5,387,885. University of North Carolina. Chapel Hill, NC 1993.

[iii] Hall, G.L. *Trap Antennas*. Technical Correspondence, QST, Nov. 1981, pp. 49-50.

Architectural Power Reduction and Power Analysis

An Architectural Level Power Estimator

Rita Yu Chen, Mary Jane Irwin, Raminder S. Bajwa*
Department of Computer Science and Engineering
The Pennsylvania State University
*Semiconductor Research Laboratory, Hitachi America Ltd.

June 1998

Abstract

Architecture trade-off experiments require the availability of an accurate, efficient, high level power estimation tool. We have developed such a tool that provides cycle-by-cycle power consumption statistics based on the instruction/data flow stream. While the approach only assumes a high level definition of the architecture, it does require that the energy consumption of the functional units has been previously characterized. The accuracy of our estimation approach has been validated by comparing the power values our tool produces against measurements made by a gate level power simulator of a commercial processor for the same benchmark set. Our estimation approach has been shown to provide very efficient, accurate power analysis. Using this same power estimation technique, several architecture level trade-off experiments for various architectures have been performed.

1.0 Introduction

Power dissipation has become a critical issue in processor design. When designing high performance, low power processors, designers need to experiment with architectural level trade-offs and evaluate various power optimization techniques. Thus, a tool for doing efficient and accurate architectural level power estimation becomes indispensable. Most of the research in this area falls in the category of empirical methods which "measure" the power consumption of existing implementations and produce models based on those measurements. This macromodeling technique can be subdivided into three sub-categories. The first approach [1] is a fixed-activity macromodeling strategy called the Power Factor Approximation (PFA) method. The energy models are parameterized in terms of complexity parameters and a PFA proportionality constant. Thus, the intrinsic internal activity is captured through this PFA constant. This approach implicitly assumes that the inputs do not affect the switching activity of the hardware block.

This work is supported in part by a grant from the National Science Foundation (MIP-9705128) and by Hitachi America Ltd. The authors can be contacted at yuchen@cse.psu.edu, mji@cse.psu.edu, or rbajwa@hmsi.com

To remedy this weakness of the fixed-activity approach, activity-sensitive empirical energy models have been developed. They are based on predictable input signal statistics, such as used in the SPA method [2][3][4]. Although the individual models built in this way are relatively accurate (the error rate is 10%-15%), overall accuracy may be sacrificed for the reasons of unavailable correct input statistics or an inability to model the interactions correctly.

The third empirical approach, transition-sensitive energy models, is based on input transitions rather than input statistics. The method presented in [5] assumes an energy model is provided for each functional unit - a table containing the power consumed for each input transition. The authors give a scheme for collapsing closely related input transition vectors and energy patterns into clusters, thereby reducing the size of the tables. A significant reduction in the number of clusters, and also the size of the tables and the effort necessary to generate them, can be obtained while keeping the maximum error within 30% and the root mean square error within 10%-15%. After the energy models are built, it is not necessary to use any knowledge of the unit's functionality or to have prior knowledge of any input statistics during the analysis.

Our work follows the third approach in the development of a power estimator for a commercial processor. The estimator imitates the behavior of the processor in each clock cycle as it executes a set of benchmark programs. It also collects power consumption data on the functional units (ALU, MAC, etc.) exercised by the instruction and its data. The results of our power estimator are compared with the power consumption data provided by the manufacturer [6]. This comparison confirms the accuracy of our power estimation technique [7]. To show the application of the estimator, several architecture trade-off experiments have been performed.

The rest of this paper consists of four sections. Section 2 presents the power estimation approach. Section 3 overviews the processor architecture and discusses the validation results. Section 4 shows several architecture trade-off experiments. Finally, Section 5 draws the conclusions.

2.0 Power Estimation

An overview of the power estimator is shown in Figure 1. The inputs of the estimator are a benchmark program and energy models for each functional unit. The architectural simulator consists of several parts: the

assembler, the controller, and the datapath implemented as many functional units (the small blocks labeled as **m**).

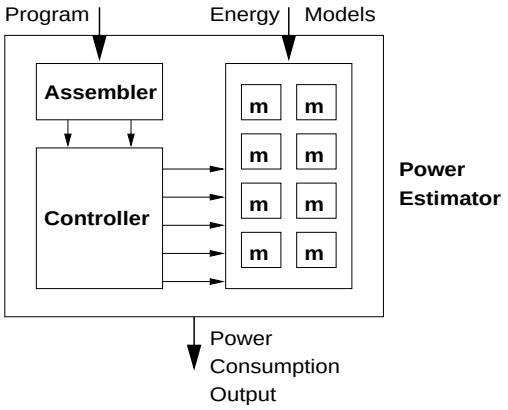


FIGURE 1. Power estimator overview

The controller generates control signals for each functional unit as required for the instructions. For example, when a load instruction puts data from the memory to the IDB bus, the "MEMtoIDB " control bit will be set. As the control signals are set, the appropriate functional units are activated. For each unit, there is a function routine which gathers all event activities. Figure 2 shows an example. In this example, when a control signal is set, the IDB bus will receive data from one of its sources. This input data to the functional unit will be used to determine its power consumption.

```
toIDB()
{
    if ctr_bit[MEMtoIDB] is set
        read data from memory;
        put data on IDB;
    if ctr_bit[MWBtoIDB] is set
        MWB buffer puts data on IDB;
    if ctr_bit[BIFtoIDB] is set
        BIF (Bus InterFace) puts data on IDB;
```

FIGURE 2. Event activities in the IDB

The energy consumed by each functional unit per access can be computed as follows where C_m is the

$$E_m = \frac{1}{2} C_m Vdd^2$$

switch capacitance per access of the functional unit and Vdd is the supply voltage. For each active functional unit, C_m is calculated from the energy model of the unit based on the previous and present input vectors. The functional units can be grouped into two classes: *bit-dependent* and *bit-independent* functional units. Here, the terms energy and power are used interchangeably.

In the bit-dependent functional units, the switching of one bit affects other bit slice' s operations. Typical bit-dependent functional units are adders, multipliers, decoders, multiplexers, etc. Their energy characterization is based on a lookup table consisting of a full energy transition matrix where the row address is the previous input vector, the column address is the present input vector, and the matrix value is the switch capacitance. All combinations of previous and present input vectors are contained in one table. A major problem is that the size of this table grows exponentially in the size of the inputs. A clustering algorithm [11] solves this problem and the sub-problems associated with it by compressing similar energy patterns. Table 1 shows an example of uncompressed/compressed energy table for a 2:1 multiplexer. The capacitance data in the energy characterization table is obtained from a switch level/circuit level simulation of a circuit level/layout level implementation of the functional unit.

TABLE 1. Energy table for a 2:1 multiplexer

Uncompressed		Compressed	
000 000	0.00	000 0xx	0.00
000 001	0.00	000 100	0.04
000 010	0.00	000 101	0.05
000 011	0.00	000 110	0.04
000 100	0.04	000 111	0.05
000 101	0.05	...	
000 110	0.04		
000 111	0.05		
000 111	0.05		
001 000	0.00		
...			

In the bit-independent functional units, the switching of one bit does not affect other bit slice' s operations, for example, registers, logic operations in the ALU, memories, etc. The total energy consumption of the functional unit can be calculated by summing the energy dissipation of the individual bits. To model the energy dissipation of a bus, the load capacitance of each bit is used as an independent bit characterization. Furthermore, each bit is assumed to have the same load capacitance. The product of the load capacitance and the number of bits is the C_m in the energy equation.

Some energy models are not built from the complete functional unit but from smaller subcells of these units. For example, because a 4:1 multiplexer can be made from three identical 2:1 multiplexers, its energy dissipation can be calculated by using the energy model of a 2:1 multiplexer. This reduces 2^{6**2} = 4096 table entries to 2^{3**2} = 64 entries before compression. After building the subcell energy model, energy routines will implement those operations needed to calculate the functional unit power dissipation from subcells.

3.0 Validation

To validate this power estimation technique, we have implemented it for a commercial processor which integrates a 32-bit RISC processor and a 16-bit DSP on a single chip. Figure 3 shows its main block diagram. The processor core includes the X-memory, the Y-memory, the buses, the CPU engine and DSP engine. The CPU

engine includes the instruction fetch/decode unit, a 32-bit ALU, a 16-bit Pointer Arithmetic Unit (PAU), a 32-bit Addition Unit (AU) for PC increment, and 16 general purpose 32-bit registers. As the ALU calculates the X-memory address, the PAU can calculate the Y-memory address. Additional registers in the CPU support hardware looping and modulo addressing. The DSP engine contains a MAC, an ALU, a shifter, and 8 registers (six 32 bits wide and two 40 bits wide).

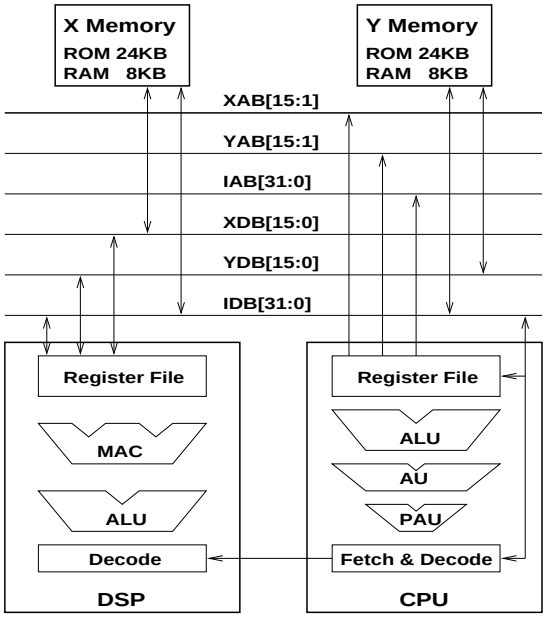


FIGURE 3. Block diagram of the commercial processor

A set of simple synthetic programs listed in Table 2 are used as the benchmarks to validate our power analyzer. These are the same benchmarks used to test the processor in [6]. Power consumption of the instruction loop buffer (ILB), memories, buses, ALU, the multiplier and other functional units are collected as the benchmarks are run. Data values used were those which maximized the switching in the datapath. In Table 2, “padd pmuls movx+ movy” contains four separate operations executed simultaneously. They are an addition, a multiplication, two loads and two address increments. “Padd pmuls movx movy” is the same as the previous instruction except it contains no address increments.

TABLE 2. The Power Benchmarks

Program	Use ILB	Main Loop Operation
pow032	no	padd pmuls movx+ movy+
pow033	yes	padd pmuls movx+ movy+
pow035a	yes	padd pmuls movx movy
pow034	no	padd pmuls
pow035	yes	padd pmuls
pow035b	yes	padd
pow035d	no	nop
pow035c	yes	nop

Figure 4 compares the results generated by our power estimator with the data presented in [6]. The power consumption data of each program is normalized to that of the pow034 benchmark. As you can see, for most of the benchmarks our simulator produces results very close to those reported in [6]. However, in both the case of the pow035d benchmark and the pow035c benchmark, the power consumption is underestimated by our simulator. Since we did not have access to the design of the control unit of the processor, average data is used to characterize its power consumption. Also, a statistical power consumption value was used in estimating clock power, capturing the average value of the clock drive circuitry and clock distribution tree. Power consumed by the gated clock logic is not captured. These two benchmarks use the least power overall, so that clock and control unit power account for a higher percentage of the total power consumption. Overall, the average error rate of power analysis by our simulator is 8.98%.

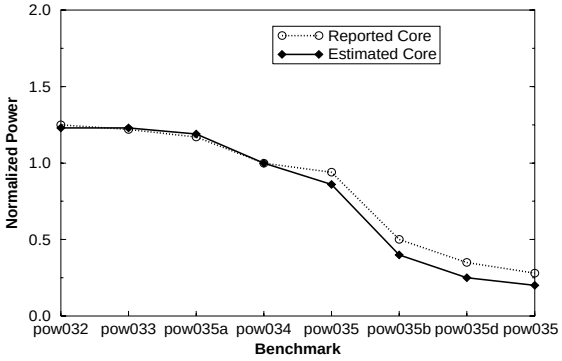


FIGURE 4. Core (DSP+CPU) estimated and reported power

4.0 Architectural Level Trade-offs

Using our power estimator, we have experimented with various architectural level trade-off. For example, the DSP multiplication operation can consume 67.8% power of the DSP engine and 34.4% of the total power as shown in Figure 5 for the pow035 benchmark. Thus, not surprisingly, reducing the power consumption of the MAC unit would lead to a significant power reduction for programs with a number of multiplication operations.

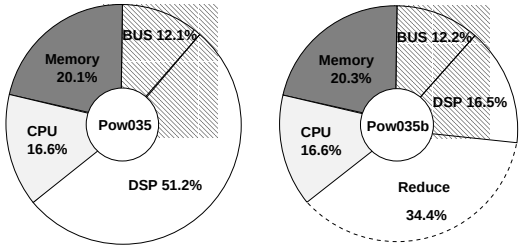


FIGURE 5. Power comparison for MAC unit

More architectural level experiments [9] have been performed using a power estimator based on a 5-stage pipelined DLX processor [10]. The power estimator

identifies the most power hungry modules: instruction and data caches, pipeline registers and the register file. To reduce register file accesses, an architecture optimization approach - early bypass detection - places the register forwarding logic in the ID (Instruction Decode) stage. As soon as an instruction is decoded, bypass detection will be performed before the register access. If the machine is going to forward a source register from a pipeline register, then the register file access will not occur eliminating unnecessary register reads. Another modified register forwarding approach - source to source forwarding - avoids fetching a source register if it is "alive" in the pipeline register. However, for both approaches, while register file power is reduced, the extra forwarding control hardware will consume energy. Figure 6 shows the reduction of register file accesses when using early bypass detection, source to source forwarding or both. The total power consumed in register file, pipeline latches and forwarding hardware is compared in Figure 7.

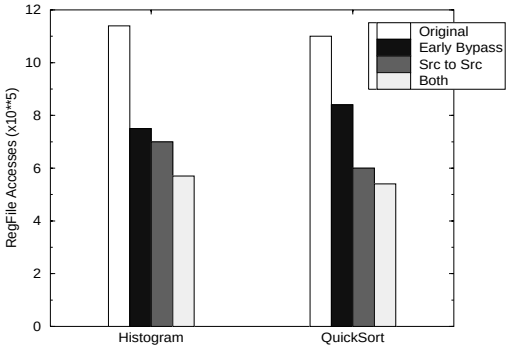


FIGURE 6. Decrease in register file accesses

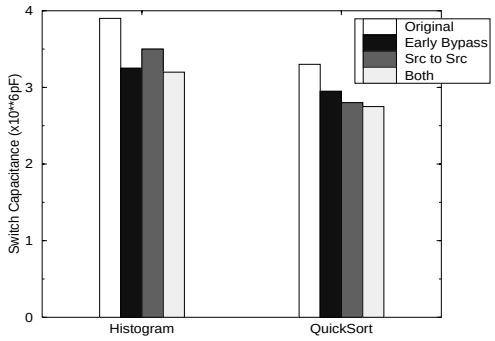


FIGURE 7. Decrease in switch capacitance

In other architectural level experiments, we have modified the normal load/store architecture to support one operand from memory by adding a memory addressing mode for several ALU instructions. The approach reduces not only the number of the register file accesses but also the instruction count. As a result, it lowers register file switching activity and instruction fetch power. However, the register memory addressing mode may increase clock cycle time or the number of clocks per instruction (CPI). This scheme also requires more complex control unit which may consume more power. Two implementation options have been considered: a 5-stage pipeline and a 6-stage pipeline. In the 5-stage implementation the MEM stage precedes the EX stage and must include address calculation hardware. This may adversely affect clock cycle time. In the 6-stage implementation the MEM stage and the EX stage are again swapped and a separate address calculation stage precedes the MEM stage. This will affect the CPI. The energy saving comparison in Figure 8 shows that the 5-stage pipeline reduces switch capacitance from 5% to 13%. But, the 6-stage pipeline consumes as much or more energy than the normal architecture due to the additional pipeline register.

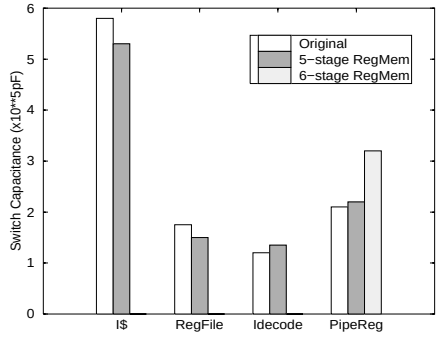


FIGURE 8. Energy saving of register memory addressing mode

Different from above approaches which change the architecture, a register relabeling technique [11] modifies only the compiler. This approach encodes the register labels such that the switching costs of all the register label transitions are minimized. It can reduce the energy of the pipeline registers, register file and the instruction bus. To decrease the energy consumption of the switching from 0 to 0 or from 1 to 1, flipflops which can guard clock signals are used to implement registers. However, because of the additional logic, the 1 to 0 or 0 to 1 transitions take more switch capacitance than unguarded registers. Overall, Figure 9 shows that the energy reduction is possible by using the register relabeling technique.

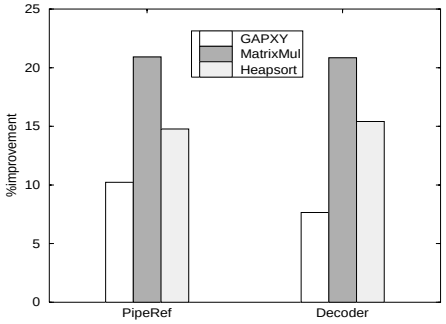


FIGURE 9. Energy improvement of register relabeling

5.0 Conclusions

In this paper, we described an accurate, high level power estimation technique. Our technique needs to do energy characterization for each functional unit (ALU, MAC, etc.) only once. When the operations of the functional units are specified by the instruction stream, the power consumed in each functional unit is calculated from its energy model. The simulation results for a commercial processor clearly verified the correctness of our power analysis methodology. Without loss of accuracy, the running time of power estimation for each synthetic benchmark program was less than 90 CPU seconds. This time saving feature of our power estimator is beneficial for reducing the design cycle time. The simulator is also applicable for power optimization research at the architecture, system software, and application software levels.

Research continues on several fronts. We have promising results to improve the estimation of control power using a hierarchical technique [12]. Since clock power consumption is typically the largest portion of the total chip power, it needs to be accurately modeled. To characterize clock power, a statistical energy model of the ungated clock drive logic/distribution tree will be combined with transition-sensitive models of different clock gating schemes. Finding a more accurate high level interconnect power model is also important. More complex memory structures (e.g., caches) can be modeled using methods like the one presented in [13].

Bibliography

1 S.R. Powell and P.M. Chau, "Estimating power dissipation of VLSI signal processing chips: The PFA technique," in *VLSI Signal Processing IV,* 1990.

2 P. Landman and J. Rabaey, "Power estimation for high level synthesis," in *Proc. of EDAC-EUROASI*C, 1993.

3 P. Landman and J. Rabaey, "Black-box capacitance models for architectural power analysis," in *Proc. of Inter. Symp. on Low Power Electronics and Design (SLPED)*, April 1994.

4 P. Landman and J. Rabaey, "Activity-sensitive architectural power analysis for the control path," in *Proc. of SLPED*, April 1995.

5 H. Mehta, R.M. Owens, and M.J. Irwin, "Energy characterization using clustering," in *Proc. of Design Automation Conf.*, June 1996.

6 R. Bajwa, N. Schumann, and H. Kojima, "Power analysis of a 32-bit RISC microcontroller integrated with a 16-bit DSP," in *Proc. of SLPED*, Aug. 1997.

7 R.Y. Chen, R.M. Owens, M.J. Irwin, and R.S. Bajwa, "Validation of an architectural level power analysis technique," to appear in *Proc. of Design Automation Conf.*, June 1998.

8 H. Mehta, R.M. Owens, and M.J. Irwin, "Instruction level power profiling,'"in *Proc. of Inter. Conf. on Acoustics, Speech and Signal Processing*, Aug. 1996.

9 Atul Kalambur, *Micro-architectural techniques for low power processors*, M.S. Thesis, Penn State University, 1997.

10 J.L. Hennessy, and D.A. Patterson, *Computer Architecture - A Quantitative Approach*, Morgan Kaufmann, 1996.

11 Huzefa Mehta, *System Level Power Analy*sis, Ph.D. Thesis, Penn State University, 1996.

12 R.Y. Chen, M.J. Irwin and R.S. Bajwa, "Architectural level hierarchical power estimation of control units," submitted to ASIC'98.

13 J. Kin, M. Gupta and W. Mangione-Smith, "The filter cache: An energy efficient memory structure," in *Proc. of MICRO'97*, Nov. 1997.

Multivariate Power/Performance Analysis For High Performance Mobile Microprocessor Design

George Z.N. Cai[+] Kingsum Chow[++] Tosaku Nakanishi[+] Jonathan Hall[+++] Micah Barany[+]

[+] Intel Corp. MS RA2-401
MHPG Low Power Design Lab
Hillsboro, Oregon 97124

[++] Intel Corp. MS EY2-09
MicroComputer Research Labs
Hillsboro, Oregon 97124

[+++] Intel Corp. MS RA2-401
MD6 Division
Hillsboro, Oregon 97124

1 Abstract

The power consumption becomes one of the important architectural and design issues besides the primary goal of performance on high performance microprocessors. With increasing high clock frequency and design complexity on high performance mobile microprocessors, the power consumption has directly impacts on future generation of mobile microprocessor quality and reliability. Traditional performance and power analysis can not provide enough analysis and information to support decisions on power/performance tradeoffs on architecture and design for high performance mobile microprocessors. The multivariate power/performance analysis provides a systematic method to identify and analyze the power consumption and the corresponding performance impact. It can be useful method to support power/performance tradeoffs in microprocessor architecture and design.

2 Challenge to power/performance analysis

With dramatically increasing microprocessor frequency, architectural and design complexity, microprocessor power analysis and reduction become one of the most important issues in high performance microprocessor design [1,3,4,6,7]. The power limitation of high performance mobile microprocessors is already critical to design. The high power consumption of a high frequency microprocessor not only increases the cost of mobile computing system integration and it also reduces the battery life time and generates additional heat to compact mobile computing systems. In other words, a mobile computing system quality and reliability could be affected by the high power manufacturing, consumption of a high frequency microprocessor. To avoid the microprocessor from overheating, especially on the application programs with high power consumption characteristics, we must explore the new architecture and techniques, which are beyond the traditional clock gating method to further reduce microprocessor power consumption. To achieve the minimal power consumption in a high performance mobile microprocessor, we have used multivariate analysis between power consumption and performance in the microprocessor architecture and design. In this paper, we present a method, which employs multivariate analysis to aid the design of microprocessor in having less power consumption.

There are two fundamental questions in a high performance low power microprocessor design: (1) How do we identify targets for power reduction within microprocessor architecture and design. It basically asks where the power is heavily consumed and why. It challenges the entire microprocessor architecture and micro-architecture from the point of view of power consumption. (2) How do we reduce power consumption at the identified architecture and design points with either minimal or no performance impact. This essentially asks for a power/performance optimization in a complex mobile microprocessor. It involves determining new tradeoffs in the design of micro-architecture between the performance and power

This work was supported by Intel Corporation. For information on obtaining reprints of this article and commercially use this method please contact Tosaku Nakanishi at tnakanis@ichips.intel.com.

consumption. The traditional microprocessor architecture tradeoff decisions are based on cost/performance analysis. These decisions are good for a microprocessor performance. They may or may not be good for power saving. In high performance mobile microprocessors, the power consumption requirement is critical. The architecture and design tradeoff criteria and priority of the mobile microprocessors are very different comparing to the desktop computer and servers in this sense. The conventional microprocessor architecture and design analysis can not provide enough information and analysis data to make optimal design decisions for high performance mobile microprocessors.

3 Multivariate power/performance analysis

Multivariate power/performance analysis is a multivariate statistical analysis method. It is concerned with the collection and interpretation of simulation data from multiple variables of microprocessor power consumption, architectural and design simultaneously [2,8]. Multivariate power/performance analysis helps microprocessor architects and designers to identify the possible power reduction targets within complex microprocessor architecture and micro architecture. It also provides an easy and quick way to predict a potential power consumption and performance impact due to the architecture and design changes. This power consumption and performance prediction and estimation is used to keep track of the architectural modifications on the correct direction before complex detailed microprocessor power/performance simulations carry out.

There are 8 steps for setting up and running multivariate power/performance analysis. They are: create power/performance model, qualify power/performance model, select representative variables, select target software and workload, collect the simulation results and build the analysis database, identify power reduction target and logic, modify micro-architecture, logic, and circuit implementation, and verify power reduction and performance impacts.

One method of multivariate power/performance analysis is the multivariate correlation analysis. In a traditional correlation analysis, the correlation between two variables is studied. In a multivariate correlation analysis, the correlations of all pairs of variables are studied. Multivariate

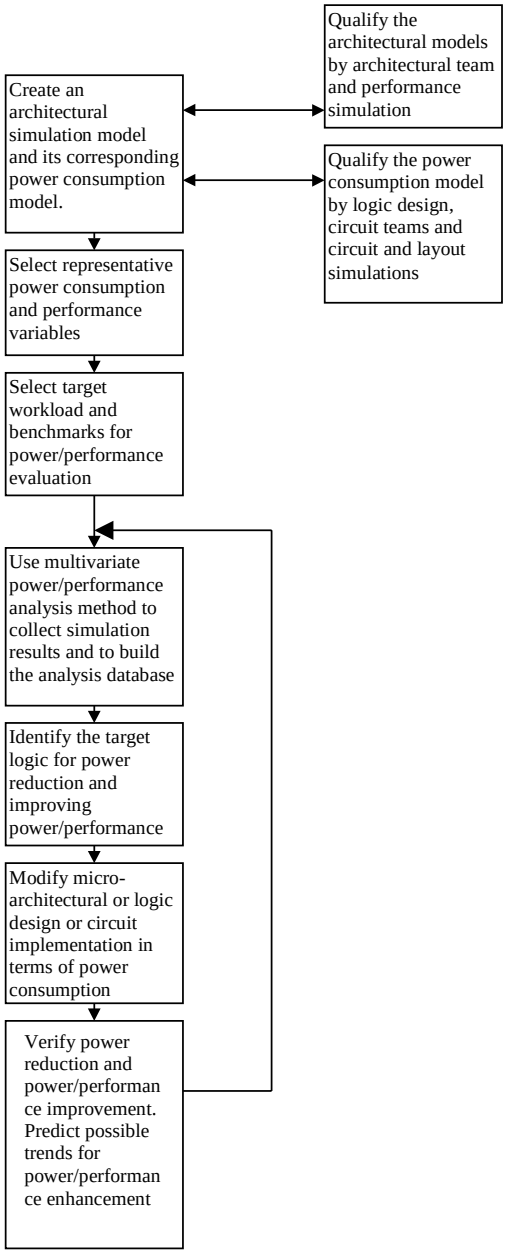


Figure 1 The flow chart of implementing multivariate power/performance analysis for high performance microprocessor design

power/performance correlation analysis is a specialization of multivariate correlation analysis. In this specialization, the correlations between architectural parameters, different levels of power consumption and performance measurements are studied.

When we measure a group of microprocessor architectural parameters and their corresponding power consumption, not only are we interested in their central tendency and variation of each individual architecture parameters, but also in an assessment of the association among the performance and different levels of power consumption, such as a full chip power consumption and a logic cluster power consumption. Multivariate power/performance correlation analysis focuses on the relationships between variations of performance and power consumption for each architectural parameter. Mathematically, the relationships are measured by correlation coefficients, which can have values between –1 and 1, inclusively. A strong relationship is indicated by a large positive or negative correlation coefficients while a weak relationship is indicated by a correlation coefficient that is close to 0. The correlation coefficient by itself cannot determine if an architectural parameter has a high power consumption or performance. Instead it only tends to indicate that an architectural parameter is a tight grip on the power consumption or performance. The correlation coefficients of those parameters are not systematically controlled by architects or designers but by complex factors that need not be precisely determined. In other words, the multivariate coefficient values represent the fact of power consumption and performance from compounded multiple effects on complex microprocessor architectural and design factors. This is one of the reasons that multivariate power/performance analysis is different from other power and performance analysis methods.

The multivariate power/performance analysis does not assume that any particular architectural parameter has a causal logic relation with power consumption at full chip level and logic cluster level. By observing how the architectural parameters, power consumption, and performance vary with each other in a set of experiments, our analysis tend to have the effect of summarizing complex relationships among variables without deriving all possible complex equations. However, it does not establish a causal relation between an architectural parameter and power consumption or performance. In the multivariate power/performance analysis, we do not make any causal logic relation assumption during the correlation analysis. This assumption reflexes that the power consumption and performance of a high performance mobile CPU is related to many architectural and design issues. The simple causal logic relation may not always fit in a real microprocessor design. The correlation studies could be the common, inexpensive, and valuable power/performance studies in general. It, however, will give us an indication the effect of altering an architecture and design on power consumption and performance.

Table 1 Classification of architectural parameters based on their correlations with microprocessor performance and power consumption

Classes of architectural parameters	Correlation with microprocessor performance	Correlation with microprocessor power consumption
Class 1	Strong	Strong
Class 2	Strong	Weak
Class 3	Weak	Strong
Class 4	Weak	Weak

Table 1 summarizes the classification of architectural parameters based on their correlations with performance and power consumption.

Class 1 correlation for a group of architectural parameters means that the architectural parameters are strongly correlated to both performance and power consumption. If the correlation coefficients are both positive or both negative, reducing power consumption is likely to reduce performance. On the other hand, if the correlation coefficients are of the opposite signs, reducing power consumption and increasing

performance are possible. Not surprisingly, the latter case is rare.

One example for class 1 correlation is that the bus unit power/performance correlation coefficients (coefficient with the full chip power consumption and coefficient with overall performance) may have the opposite signs. This means that the higher power consumption on the bus unit and the lower over all microprocessor performance will be achieved. One of reasons could be too many inaccurate speculative instruction fetches and memory data accesses to deliver correct instructions and data to internal core for execution. Therefore, even the bus unit is extremely busy; the internal core is still idle. When the incorrect speculations are reduced, the bus unit power consumption is reduced and the instructions and data can be delivered to the execution units effectively.

Class 2 correlations for a group of architectural parameters mean that the architectural parameters are strongly correlated to performance but weakly correlated to power consumption. While such architectural parameters are good candidates for performance, they are not good candidates for power reduction targets.

Class 3 correlations for a group of architectural parameters mean that the architecture parameters are strongly correlated to power consumption but weakly correlated to performance. While such architectural parameters may have minimal impact on performance, they are excellent candidates for power reduction without severe effect on performance degradation. We may use a new micro-architecture and circuit with aggressive power reduction to replace the exist micro-architecture or circuit that being represented by these group of architectural parameters. The class 3 correlation provides opportunities for architects and designers to further reduce the power consumption on a very complex microprocessor. The best part of the class 3 correlation is that it tries to isolate the power consumption problems from complex performance impacts.

Class 4 correlation for an architectural parameter means that the architectural parameter is weakly correlated to both performance and power consumption. Obviously, such architectural parameters should not be selected for performance optimization or as power reduction targets.

These four classes of power/performance correlations represent the overall effects, such as micro-architecture, clock gating, etc. We have to mention that the power characteristics of selected benchmarks and target software may affect the correlation in some cases. Therefore, the correlation class variations of a logic block/unit between different benchmarks provide the additional information of the power/performance behaviors under different software executions. The average power/performance correlation of a logic block provides the power/performance behaviors in general.

In a microprocessor design, the correlation analysis must reach further details to achieve additional power reduction possibilities. For example, we may use the following table to oversee the correlation among all related micro architectures of Load buff and Store buff.

LoadBufFull and StoreBufFull are two architectural parameters that have varying correlations with microprocessor performance and power consumption. The resident logic blocks and units are displayed along with their

Table 2 Two examples of architectural parameters and their correlations with performance and power consumption

Architectural Parameter	Resident Logic Block	Resident Logic Unit	Microprocessor Performance		Block Power Consumption		Unit Power Consumption		Microprocessor Power Consumption	
LoadBufFull	Load buffer	Load/Store	ISPEC95 number	Corr. Value	mW	Corr. Value	mW	Corr. Value	mW	Corr. Value
StoreBufFull	Store buffer	Load/Store	ISPEC95 number	Corr. Value	mW	Corr. Value	mW	Corr. Value	mW	Corr. Value

performance and power correlations Microprocessor architects may use such a table to scan for class 3 architectural parameters and their corresponding resident logic blocks and units for power reduction opportunities. In our analysis, we have found that principal component analysis [2] is a good data reduction technique for multivariate power/performance analysis. Principal component analysis maximizes the variance of information in the first few major components. By examining the principal components, instead of all the data, architects and designers can easily glance at the simulation results respected with the performance and power consumption in different depths. Principal component analysis also enables us to build a power/performance prediction model. This prediction model captures the complex correlation among multiple design variables and power/performance within a high performance microprocessor. The prediction model also provides a quick estimation of the power consumption and performance changes due to multiple architectural variable modifications. This prediction model is useful when there are micro-architecture changes experimentally. The prediction is able to indicate if the power/performance is on the right track while the micro-architecture is being modified. By using such a prediction model, we can reduce the number of expensive and time consuming simulations. In other words, the multivariate power/performance prediction model is a complementary method to the microprocessor simulations.

The collection of qualified multivariate data is extremely important. One of the limitations of this analysis is that unqualified data can mislead the multivariate power/performance analysis into a wrong direction. Unqualified data is a complex issue which can occur at either the original architectural simulator, or the corresponding power modeling, or a data collection method, or incorrectly using the multivariate power/performance method. This limitation leads us to another important topic: the multivariate power/performance analysis qualification and verification. It is very useful to qualify the multivariate power/performance analysis results with corresponding architecture performance and the circuit simulation results. Theoretically, all the correlation in the multivariate power/performance analysis should be preserved in different design modeling stages, such as a high level architecture model, RTL model, and layout model. Because all measurements may contain errors, this correlation and prediction qualification and verification is a very complex issue.

From our experience, multivariate power/performance analysis helps us to identify the power reduction targets in a very complex high performance microprocessor design. For example, when multivariate power/performance analysis is applied to evaluate how a speculative branch prediction impacts on microprocessor power and performance, useful information can be obtained to make a correct design decision.

4 Acknowledgment

We would like to thank many people involved in the multivariate power/performance analysis at Intel, especially Doug Carmean, Steve Gunther for their comments and helps on research directions. We would also like to thank Wayne Scott, Pierre Brasseur, and Konrad Lai for their support and Shih-Lien Lu for useful comments for the earlier drafts of this paper.

5 References

[1] Abdellatif Bellaouar and Mohamed Elmasry, “Low-Power Digital VLSI Design Circuits and Systems”, Kluwer Academic Publishers, 1997.

[2] Kingsum Chow and Jason Ding, “Multivariate Analysis of Pentium® Pro Processor”, Intel Software Developers Conference, Portland, Oregon, October 27-29, 1997, pp. 84-91.

[3] R. Mehra and J. Rabaey, “Behavioral Level Power Estimation and Exploration”, Proceedings of the International Workshop on Low Power Design, Napa, California, pp. 197-202, April 1994.

[4] Gary Yeap, “Practical Low Power Digital VLSI Design”, Kluwer Academic Publishers, 1998

[5] C. Huang, B. Zhang, A Deng, and B. Swirski, “The Design and Implementation of PowerMill”, Proceedings of International Symposium on Low Power Design, pp. 105-109, 1995.

[6] P. Lanman and J. Rabaey, “Activity-sensitive Architectural Power Analysis”,

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol 15, no. 6, pp.571-587, June 1996

[7] Jan M. Rabaey, “Digital Integrated Circuits”, Prentice Hall Electronics and VLSI Series, pp. 234-255, 1996.

[8] Sam K. Kachigan, “Multivariate Statistical Analysis”, 2nd Edition, Radius Press, New York, 1991.

[9] J. Winer, “Statistical Principles in Experimental Design”, 2nd Edition, McGraw-Hill, New York, 1971.

Power-Aware Architecture Studies: Ongoing Work at Princeton

Christina Leung, David Brooks, Margaret Martonosi, and Douglas W. Clark*
Depts. of Electrical Engineering and Computer Science*
Princeton University
Contact: martonosi@princeton.edu

Abstract

The effects of power consumption increasingly shape processor design choices not only in battery-operated devices, but also in high-performance processors. Unfortunately when compared to the huge body of performance-oriented architecture research, much less research attention has been placed either on modeling power consumption at the architectural level, or on devising organizational mechanisms for reducing power consumption in high-end processors.

Our work seeks to address these issues in two main ways. First, we are developing a suite of high-level modeling tools that allow architects to manipulate designs and compare power tradeoffs in a manner similar to current performance-oriented architecture studies. In this paper we discuss the status of the work and our observations on how the needs of power modelling affect the simulator speed and structure. Second, we are harnessing parts of this infrastructure to evaluate a technique for aggressively clock-gating arithmetic units in response to observed operand values. We find that many operand values in current programs are much smaller than the actual word width of the machines. As a result, dynamic techniques for adjusting functional unit width to match operand width are promising for reducing power consumption in these units.

1 Introduction

Current-generation high-end microprocessors can consume 30 watts or more of power in normal operation. System designs become increasingly challenging as chips and systems get smaller and power consumption increases. While power consumption is clearly a significant worry in battery-operated portable devices, power is also a pressing concern in desktop systems as well. Most notably, processors in the 30W power range require elaborate cooling mechanisms to keep them from overheating. With each new generation of chips, their implementation is increasingly shaped by attempts to keep power consumption and thermal load to manageable levels. High power is also a system cost issue: fancy heat sinks can be expensive, and increased power consumption also raises the costs of the power supply and capacitors required.

For all these reasons, the power consumption of high-end processors has been a major industry concern for several chip generations. Initial attempts to limit power have focused on relatively straightforward applications of voltage scaling and clock gating. With each new generation, however, more strenuous efforts must be made, and these efforts are beginning to include *organizational* attempts to limit power consumption. The academic architecture community, however, has been slow to respond to this situation. While the RISC revolution spurred an increased focus on quantitative evaluation of architectural tradeoffs, the quantities most often studied have been performance-oriented. The time has come for academics to consider both power and performance as important architecture metrics to be optimized together.

The work described here consists of two main thrusts. First, we are developing architectural-level tools to model and evaluate power consumption in high-performance processors. We are beginning to use these tools to evaluate power-aware organizational tradeoffs in processor design. Second, we are also researching mechanisms for dynamic operand analysis that allow processors to use arithmetic functional units more efficiently.

2 Modeling Power Tradeoffs in High-Performance Processors

The traditional academic approach to processor organization tradeoffs has been to consider their effect on performance. In cache studies, for example, one might evaluate a fast direct-mapped cache against a slower set-associative cache with a higher hit ratio. The design with superior performance would be the superior design, modulo worries about hardware complexity and area.

Academics have rarely considered power in these analyses, although they have used very informal estimates of chip area as a proxy for power. Thus, in the cache analysis above, caches requiring equal amounts of SRAM bits might be regarded as equivalent from the power standpoint. To first order, such estimates are useful, but as power issues become more important, more careful area analyses and resulting power tradeoffs are warranted.

With this in mind, we are building modeling infrastructure that supports power analyses at the *architectural* level. Using the SimpleScalar toolset as a starting point [2, 1], we have built expanded modules that include power models and rudimentary power accounting.

2.1 Cache Power Models

We have begun this effort by modifying SimpleScalar's cache model to account for power dissipation. In particular, we started by incorporating a cache power model first described in the literature by Su and Despain [5]. This model considers the cache in terms of three primary modules: the address decoding path, the cell array in which the data is stored, and the data I/O path by which data enters or leaves the cache.

Our simulator tracks current and previous data and address values on the lines entering and exiting each module. Since CMOS power dissipation is related to the frequency at which bits switch from 0 to 1 or vice versa, tracking current and previous values allows us to track data on the frequency of such bit changes.

While power modeling at this level is very implementation-dependent, our primary goal in building these models is not to provide absolute wattage estimates, but rather to provide higher-level intuition on which organizations use more or less power. Such relative comparisons can be gotten more easily than precise absolutes.

2.2 Implications of Power Modelling on Simulation Speed, Structure

Adding power modeling to the simulator has significant implications on the speed and structure of the simulator. Tracking bit changes within the decoding path requires that we store the last address referenced so we can compute bit switches between address decoding bits based on the current and previous cache address. More significantly, tracking bit switches in the cell array and the I/O path requires storing the cached data itself. SimpleScalar is usually used in a mode in which the functionality and timing of the cache is simulated without actually storing the data in the cache. The memory overhead related to data storage and bit-change calculations varies depending on the cache organizations and sizes, but can result in 25% or more increases in SimpleScalar process size. Similarly, runtime overheads can also be quite significant. Our unoptimized prototype version of the cache power simulator currently runs about 50% more slowly than the original sim-cache shipped with SimpleScalar.

2.3 Summary

Our work on power modeling has focused thus far on the caches, but we intend to broaden this modeling infrastructure considerably. As we do, we note that relative power analysis and tradeoffs between like structures (e.g. instruction cache and data cache) is fairly straightforward, but tradeoff analysis among different structures needs more information. What is required is either an absolutely accurate power model of each structure in the tradeoff, or a kind of accurate "exchange rate" that correctly estimates how two structures (or more) can be compared. For most academics, the latter is more easily obtained. An example might be the claim that some number of kilobytes of instruction cache was equivalent in power to some other number of entries in the reorder buffer. Our goal is not simply to maximize performance for a fixed power consumption, but also to consider regions of higher or lower power efficiency, that is, to evaluate tradeoffs from the perspective of the energy-delay product metric.

Another possible approach in developing architectural-level power models is to follow Liu and Svensson [4], who refined a coarse complexity-based model (power follows the number of gate-equivalents) into a set of specialized and more accurate models. They built special parameterized models for logic, memory, interconnect, and clock distribution, and then validated these against published information on the Digital Alpha 21064 and the Intel 80386. We will be evaluating this technique within SimpleScalar in upcoming work. Overall, our goal is focused more on incorporating power abstractions into high-level architectural simulators, with lesser emphasis on developing extensive new models.

3 Operand Value Analysis for Aggressive Clock Gating

In parallel with our efforts to develop power simulation and modelling techniques for architectural decisions, we are also investigating particular power reduction ideas whose quantitative evaluation will benefit from the modeling infrastructure we are developing. In particular, this section focuses on dynamic techniques that work to save power by recognizing and exploiting particular attributes when present in operand values.

3.1 Background

Current processors exploit a variety of clock gating techniques to reduce power dissipation in CMOS circuits. That is, information learned about the instructions during the decode phase is used to determine which CPU units will not be needed for this instruction. Such information enables or disables the clock signal at particular chip modules. By disabling the clock, one prevents power dissipation at unused units.

Clock gating has traditionally been based on static instruction attributes embodied in the opcode. Our approach extends on this to exploit dynamic properties of the operands as they appear at runtime. Our work begins from the observation that in many codes, the dynamic values stored and manipulated in registers are often much smaller than the full word width provided by the machine.

3.2 Our Approach

In order to exploit the difference between average operand value width and machine word width, we are exploring the potential of gating off the top end of functional units to save power in instances when small operands are being manipulated. Figure 1 illustrates the basic structure we propose.

In our approach, the last step of any arithmetic computation is for the functional unit to perform a zero-detect on the high-order 48 bits of the result. (In some CPUs, some sort of zero-detect might be performed at this stage anyway, in order to update condition codes.) If the result of the zero detect is that none of the high-order bits were on, then the result is said to be a low-bit value. If any of the high-order 48 bits are on, then the result is a high-bit value. This "high-bit" flag value (0 or 1) is associated with values as they travel back to the register file, to reservation stations, and to the bypassing paths.

When operands arrive at an arithmetic functional unit, their flag value is used to gate the clock that controls the input latch. In all cases, the low-order 16 bits are clock into the latch. In small-operand cases, the latch for the high-order bits is never clocked. This prevents bit changes from occurring inside the functional unit, thus diminishing the power requirements.

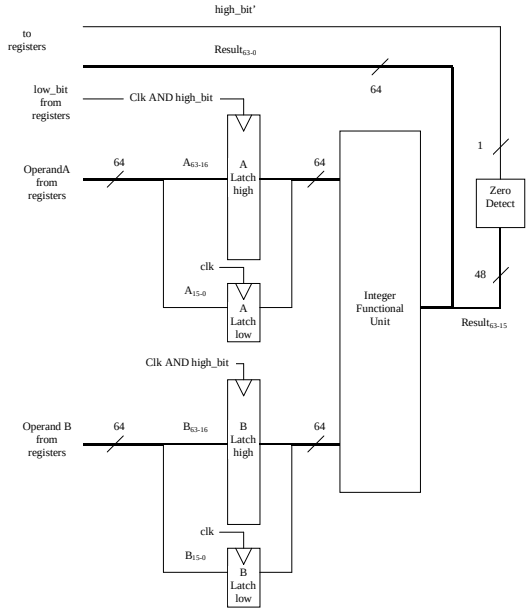


Figure 1: Block diagram of value-based clock gating at ALU input.

3.3 Experimental Results

Whether or not this approach is beneficial depends on: (1) the frequency of small operands in the code, (2) the power dissipation saved by gating off most of the arithmetic unit, and (3) the power dissipation increased by computing and maintaining the high-bit indicator flags. Our initial results focus on the first of these three factors. We are working now to generate arithmetic unit power models that help us quantify the tradeoffs between the second and third of the factors listed.

The potential of this approach is illustrated in Figure 2. In this graph, the total height of each bar corresponds to the observed frequency small-operand operations performed in eight of the SPECint95 benchmarks, as a fraction of the total operations that were eligible for consideration. The operations eligible for this optimization include arithmetic, logical, and some shift operations. In addition, there is a fourth category that includes other miscellaneous operations, such as some comparison instructions.

Overall, the results show that a large number of SPECint95 instructions operate on small operands and are therefore eligible for the optimization. Fractions range from 56% for go, up to over 70% for m88ksim and compress. We note, however, that in many cases, the small operands are for logic instructions. The benefits of clock gating for logic instructions are implementation dependent. If the implementation would be performing a zero-detect anyway, for condition codes, then power savings may result from storing this result. On the other hand, if the zero-detect and clock gating logic is all new, then the power savings resulting from saving part of the logic operation (much simpler than an add or multiply) is less clear.

Although we present results here only for integer codes, we are currently exploring mechanisms for applying similar techniques on floating point operands as well. For floating-point operands, the approaches used will depend on the type of operation being performed. For example, in a floating-point addition, determining which bits are needed and relevant cannot be performed until the two operands have been normalized (i.e., their exponents adjusted). In contrast, floating-point multiplications can have mantissa checks that are more similar to their integer counterparts. Finally, we are also exploring options for using compile-time analysis and profiling to determine narrow bit-width operations without the overhead of dynamic on-line checks. We are also considering approaches for speculatively assuming narrow bit widths in some operations, and then cleaning up computation via overflow detection in cases when the full bit-width indeed should have been used.

4 Conclusions

Overall, the philosophy embodied by this work is that appropriate monitoring infrastructure is imperative to gaining insights on power tradeoffs at design time. In addition, detailed operand analysis techniques, applied at compile-time or run-time, are a promising method for using operand values to drive clock gating techniques, and offer sizable

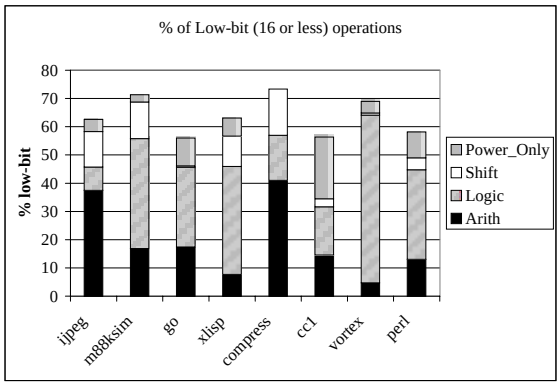


Figure 2: Operand width distribution in the SPECint benchmarks.

improvements over purely opcode-based clock gating techniques.

References

[1] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. *Computer Architecture News*, pages 13–25, June 1997.

[2] D. Burger, T. M. Austin, and S. Bennett. Evaluating Future Microprocessors: the SimpleScalar Tool Set. Technical Report TR-1308, University of Wisconsin-Madison Computer Sciences Department, July 1996.

[3] P. Landman. High-level power estimation. In *International Symposium on Low-Power Electronics and Design*, pages 29–35, August 1996.

[4] D. Liu and C. Svensson. Power consumption estimation in CMOS VLSI chips. *IEEE Journal of Solid-State Circuits*, pages 663–670, June 1994.

[5] Ching-Long Su and Alvin M. Despain. Cache design trade-offs for power and performance optimization: A case study. In *International Symposium on Low-Power Design*, pages 63–68, April 1995.

[6] V. Tiwari, S. Malik, and A. Wolfe. Compilation techniques for low energy: An overview. In *Proc. Symp. on Low Power Electronics*, October 1994.

[7] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on VLSI Systems*, 2(4):437–445, December 1994.

[8] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on VLSI Systems*, 2(4):437–445, December 1994.

Architectural Tradeoffs for Low Power

Vojin G. Oklobdzija*

Integration
Berkeley, CA 94708
http://www.integr.com

*Electrical Engineering Department
University of California, Davis, CA 95616
http://www.ece.ucdavis.edu/acsel

Abstract
A study of several RISC, DSP and embedded processors was conducted. It has been shown that the transistor utilization drops substantially as the complexity of the architecture increases. Simple architecture that are enabling full utilization of technology are favorable as far as energy efficient design styles are concerned. The results favor simple architectures that leverage performance improvements through technology improvements.

1. Introduction

Demand for reducing power in digital systems has not limited to systems which are required to operate under conditions where battery life is an issue. The growth of high-performance microprocessors has also been constrained by the power-dissipation capabilities of the package using inexpensive air-cooling techniques. That limit is currently in the neighborhood of fifty watts. However, the increasing demand for performance (which has been roughly doubling every two years) is leaving an imbalance in the power dissipation increase, which is growing approximately at 10 Watts per year.

This growth is threatening to slow the performance growth of future microprocessors. The "*CMOS ULSIs are facing a power dissipation crisis*" in the words of Kuroda and Sakurai [1]. The increase in power consumption for three generations of Digital Equipment Corporation, "Alpha" architecture high-performance processors is given in Fig. 1.

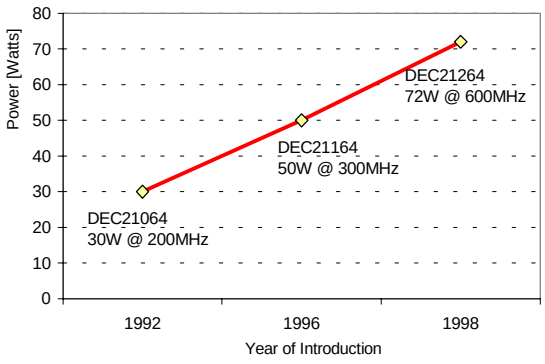


Fig. 1. *Power increase for three generations of DEC "Alpha" processor*

2. Comparative Analysis

Most of the improvement on power savings is gained by technology. Scaling of the device and process features, lowering of the threshold and supply voltage result in an order of magnitude savings in power. Indeed, this resulting power reduction has been a salient achievement during the entire course of processor and digital systems development. Had this not been the case then the increase in power from one generation to another would have been much larger limiting the performance growth of microprocessors much earlier.

The technology amounts for approximately 30% improvement in gate delay per generation. The resulting switching energy CV^2 has been improving at the rate of 0.5 times per generation. Given that the frequency of operation has been doubling for each new generation the power factor $P = CV^2f$ remained constant (0.5 X 2 = 1.0). It is the increase in complexity of the VLSI circuits that goes largely uncompensated as far as power is concerned. However, it is estimated that the number of transistor has been tripling for every generation. Therefore, the expected processor performance increase is 6 times per generation (two times due to the doubling of processor frequency multiplied by the three times increase in the number of transistors).

The fact that the performance has been increasing four times per generation instead of six is a strong indication that the transistors are not efficiently used. What that means is that the added architectural features are at the point of diminishing returns.

This diminishing trend is illustrated in Table 1 which compares a transition from a dual-issue machine to a 4-way-super-scalar for the IBM PowerPC architecture.

All three implementations of the PowerPC architecture are compared at the same frequency of 100MHz. The performance of PowerPC 620, as well as power consumption has been normalized to what it would have been at 100MHz. We can observe that the power has more than doubled and quadrupled respectively in transition from a relatively simple implementation (601+) into a true super-scalar 620. The respective performance has also improved by 50 to 60% (integer) and 30 to 80% (floating-point). However, the number of Specs/Watt has gone down dramatically-- one and two times as compared to 601+. Given that all the data for all the three implementation has been compared at 100MHz, we

Table 1. Comparison of PowerPC performance / power transition[8,9]

Feature	601+	604	620	Diff.
Frequency MHz	100	100	133 (100)	same
CMOS Process	.5u 5-metal	.5u 4-metal	.5u 4-metal	~same
Cache Total	32KB Cache	16K+16K Cache	64K	~same
Load/Store Unit	No	Yes	Yes	
Dual Integer Unit	No	Yes	Yes	
Register Renaming	No	Yes	Yes	
Peak Issue	2 + Br	4 Insts	4 Insts	~double
Transistors	2.8 Million	3.6 Million	6.9 Million	+30% /+146%
SPECint92	105	160	225 (169)	+50% /+61%
SPECfp02	125	165	300 (225)	+30% /+80%
Power	4W	13W	30W (22.5W)	+225%/+463%
Spec/Watt	26.5/31.2	12.3/12.7	7.5/10	-115%/ -252%

Table 2. Transistion from single issue MIPS R5000 to MIPS R10000 implementation of MIPS architecture[8,9]

Feature	MIPS R10000	MIPS R5000	Diff.
Frequency	200MHz	180MHz	~same
CMOS Process	0.35 /4M	0.35 /3M	
Cache Total	32K/32KB Cache	32K/32K Cache	~same
Load/Store Unit	Yes	No	
Register Renaming	Yes		
Peak Issue	4 Issue	1+FP	
Transistors	5.9 Million	3.6 Million	+64%
SPECint95	10.7	4.7	+128%
SPECfp95	17.4	4.7	+270%
Power	30W	10W	200%
SPEC/Watt	0.36/0.58	0.47/0.47	-31%/ 23%

are indeed comparing the inverse of Energy-Delay product which is a true measure for power efficiency of an implementation as shown in [7].

The comparable inefficiency in power-performance factor in transition from singe-issue to a super-scalar for MIPS processor architecture is shown in Table 2. The comparison shows a 31% decrease in power efficiency for the integer code but a 23% improvement for the floating-point.

Table 3 shows that the best trade-off between performance and power has been achieved in DEC Alpha 21164 implementation of their "Alpha" architecture. The table shows comparable efficiency for MIPS, PowerPC and HP processor implementations, slightly better for Sun UltraSPARC and substantially better power efficiency for Digital 21164.

The power efficiency of DEC 21216 was achieved through very careful circuit design, thus eliminating much of the inefficiency at the logic level. This was necessary in order to be able to operate at the frequency that is twice as high compared to other RISC implementations. However, no architectural features, other than their very careful implementations are contributors to the power efficiency of DEC 21164.

It is interesting to compare what a particular improvement means in terms of power. In Table 4 we are comparing the effect of increasing the cache size for IBM 401 and 403 processors. The measurement is normalized to 50MHz. The power-efficiency has dropped by a factor of close to two, resulting from increasing the caches. Similar findings are confirmed in the case of PowerPC architecture where the decrease in power efficiency is 60% as shown in Table 5.

Metrics:

Horowitz et al.[7] introduces Energy-Delay product as a metric for evaluating power efficiency of a design.

An appropriate scaling of the supply voltage results in a lower power, however, at the expense of the speed of the circuit. The energy-delay curve shows an optimal operation point in terms of energy efficiency of a design. This point is reached by various techniques, which are all being discussed in this paper.

The fabrication technology seems more important for the energy-delay that the architectural features of the machine. This finding is consistent with the fact that the processors' performance has been increasing four-fold per generation. Though we would expect a six-fold increase in performance: the frequency has been doubling per generation and the number of transistor tripling. This shows that the transistors have not been used efficiently and that the architectural features that are consuming this transistor increase have not been bringing a desired effect in terms of the energy-efficiency of the processors.

Power Tradeoffs in DSP and Embedded Systems:

A detailed power analysis of a programmable DSP processor and an integrated RISC and DSP processor was described in the papers by Bajwa and Kojima et al [2,3]. The authors have shown a module-wise breakdown of power used in the different blocks. Contrary to many opinions it was found that the bus power is significantly smaller compared to the data path. It was also shown in this paper how a simple switch of the multiplier inputs (applicable to Booth encoded multipliers only) can reduce multiplier power by 4-8 times. Instruction fetch and decode contribute a significant portion of the power in these designs and since signal processing applications tend to spend a very large portion of their dynamic execution time executing loops, simple buffering schemes (buffers/caches) help reduce power by up to 25% [10].

Table 3. Comparison of Performance/Power and 1/Energy*Delay for representative RISC microporcessors[8,9]

Feature	Digital 21164	MIPS 10000	PwrPC 620	HP 8000	Sun Ultra-Sparc
Freq	500 MHz	200 MHz	200 MHz	180 MHz	250 MHz
Pipeline Stages	7	5-7	5	7-9	6-9
Issue Rate	4	4	4	4	4
Out-of-Order Exec.	6 lds	32	16	56	none
Register Renam. (int/FP)	none/8	32/32	8/8	56	none
Transistors/ Logic transistors	9.3M/ 1.8M	5.9M/ 2.3M	6.9M/ 2.2M	3.9M*/ 3.9M	3.8M/ 2.0M
SPEC95 (Intg/FlPt)	12.6/18.3	8.9/17.2	9/9	10.8/18.3	8.5/15
Power	25W	30W	30W	40W	20W
SpecInt/ Watt	0.5	0.3	0.3	0.27	0.43
1/Energy*Delay	6.4	2.6	2.7	2.9	3.6

In Fig. 2, the power breakdown is shown for the integrated RISC+DSP processor. For the benchmarks considered, which are kernels for typical DSP applications, the CPU functions as an instruction fetch, instruction decode and address generation unit for the DSP. Hence the variability in its power is less.

Similarly, the power consumed in the memories is quite high (in spite of their being low power, segmented bit-line memories) and shows little variation. In the case of the DSP the power variation is more and is data dependent. The interconnect power (INTR) represents the top level interconnect power which includes the main busses (three data and three address) and the clock distribution network at the top level. Clock power alone contributes between 30 and 50% of the total power consumption depending on system load.

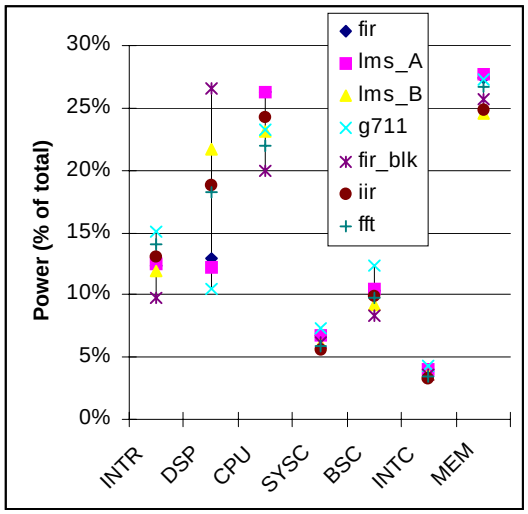
Table 4. A difference in power-performance factor resulting from increasing the size of cache[8,9]

Feature	401	403	Difference
Frequency	50MHz	66MHz (50MHz)	close
CMOS Process	0.5u 3-metal	0.5u 3-metal	same
Cache Total	2K-I / 1K-D	16K-I / 8K D	8x
FPU	No	No	same
MMU	No	Yes	
Bus Width	32	32	same
Transistors	0.3 Million	1.82 Million	600%
MIPS	52	81 (61)	+56% (+17%)
Power	140mW	400mW (303mW)	+186% (+116%)
MIPS/Watt	371	194	-91%

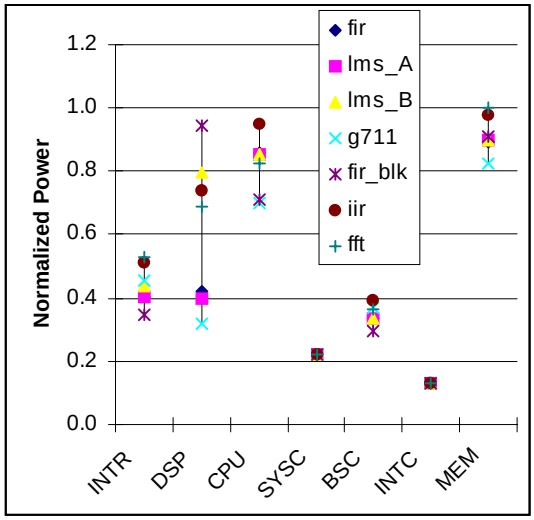
Table 5. The effect of increasing the cache size of PowerPC architecture[8,9]

Feature	604	620	Difference
Frequency	100MHz	133MHz (100MHz)	same
CMOS Process	0.5u 4-metal	0.5u 4-metal	same
Cache Total	16K+16K Cache	64K	~double
Load/Store Unit	Yes	Yes	same
Dual Intgr Unit	Yes	Yes	same
Reg- Renaming	Yes	Yes	same
Peak Issue	4 Instructions	4 Instructions	same
Transistors	3.6 Million	6.9 Million	+92%
SPECint92	160	225 (169)	+6%
SPECfp02	165	300 (225)	+36%
Power	13W	30W (22.5W)	+73%
Spec/Watt	12.3 / 12.7	7.5 / 10	-64%

The characteristics of embedded systems are quite different from those of desktop systems. For one, cost is a much more acute issue. Secondly, the computational load is a smaller well-defined set of tasks. For real-time signal processing applications, throughput behavior is typically more critical than minimum response time. These constraints dominate the design decisions. In many instances the cost of packaging is comparable to the cost of the die and using a more expensive package albeit with better heat dissipation capabilities is not an option. In the mobile arena, battery life and heat dissipation in compact designs (constricted space reduces airflow and hence the capacity to disperse heat) put downward pressure on power consumption of these processors. Depending on the application domain there are two broad approaches.



(a)



(b)

Fig. 2 *Module-wise breakdown of the chip power consumption for the kernel benchmarks for the integrated RISC+DSP processor, (a) as a percentage of the total (b) normalized*

Throughput and real-time constraints typically lead to more balanced systems, as in the case of DSPs (Harvard architecture, processor speed equal to bus speed). Balance here is a reference to a balance between Throughput and real-time constraints typically lead to more balanced systems, as in the case of DSPs (Harvard architecture, processor speed equal to bus speed). Balance here is a reference to a balance between processing speed, bandwidth and memory. Portable computing devices such as PDAs and handheld PCs form the other application domain, one in which the processors see a load similar to that of desktop systems in many respects. The StrongARM drops the processor core's clock frequency to be equal to that of it's bus' clock frequency when it makes accesses off-chip thereby curtailing it's power allowing its MIPS/Watt rating to scale.

Benchmarks are fraught with controversy and in the case of embedded systems where MIPS numbers are based on Dhrystones, it is especially meaningless. The Dhrystone suite can fit in roughly 4KB of memory. This makes the disparity or lack thereof between processor speeds and bus speeds noteworthy.

The biggest impact on performance/power is process technology. The StrongARM, which is at the high end of embedded and low power processors, benefits from DEC's process technology (same as the one used for the Alpha chips) and a full custom design. This is atypical of embedded processor design. As recently as 1.5 years ago, the SA-110 was available in 0.35 micron technology and 2/3.3V (core/IO). All of its competitors were available in technologies ranging from 0.5 to 1 micron and voltages between 3.3V and 5V. This is changing but the SA-110 and the SA-1100 have been able to maintain their leading position as low power processors by aggressively reducing the core's voltage (1.35V for the SA-1100), circuit techniques and edge triggered flip-flops. A threshold voltage of 0.35 has allowed a much lower operating voltage. Most other embedded processors have had higher threshold voltages and hence, correspondingly higher operating voltages. Over the next year or two embedded processors with lower threshold voltages and dual threshold designs will become more standard.

The ARM9TDMI which has adopted a SA-110-like, five-stage pipeline, as opposed to ARM's traditional three-stage design, and a Harvard architecture illustrates the advantages of a more balanced design and can now be clocked at 150 MHz at sub-watt power levels. Better task partitioning is possible in embedded systems, due to the applications requiring a small set of predictable tasks to be performed, allowing unused hardware to be shutdown. In DSPs, control overheads are minimized and the data-path power and activity dominates. In desktop processors by contrast the control power almost drowns out the variations in the data-path power [4]. Power analysis of DSPs and simple RISC processors show two main sources of power the data-path units (multiply-accumulate units) and memory or cache (Fig.2.).

Conclusion

The conclusion from the studies presented is that the best power-performance is obtained if the architecture is kept simple thus allowing improvements to be achieved by technology. In the other words, the architecture should not stay in the way of technology and whenever this is not the case we will experience a decrease in power efficiency.

The second finding that goes contrary to the common knowledge is that we should seek improvements via simple design but increasing the clock frequency rather than keeping the frequency of operation low and increasing the complexity of the design.

The current processors today have reached their limit in terms of power. Digital 21264 is an example of a processor which had a potential of higher operating frequency but had to lower it (to 600MHz) in order to keep the power contained. This situation was first reached by Digital "Alpha" processor but it is soon to be reached by all of the others.

More specialized systems, used in signal processing applications can benefit from re-configurable data-path designs. The main advantage is to reduce the clock and control overhead by mapping loops directly onto the re-configurable data-path.

Applications in signal processing where stream data- or block data-processing dominates it makes sense to configure the data-path to compute algorithm specific operations. The cost of configuration can be amortized over the data block or stream. Aggressive use of chaining (as in vector processing) can be used to reduce memory accesses resulting in designs that may be called re-configurable vector pipelines. Embedded architectures can, in the future, be expected to employ all or some of these techniques

Acknowledgment
Contribution of Dr. Raminder Bajwa of Hitachi Semiconductor Research Laboratories as well as support received from Hitachi Ltd. is greatly appreciated.

References

1. T. Kuroda and T. Sakurai *"Overview of Low-Power ULSI Circuit Techniques"*, IEICE Trans. Electronics, E78-C, No.4, April 1995, pp.334-344, INVITED PAPER, Special Issue on Low-Voltage Low-Power Integrated Circuits.
2. R. S. Bajwa, N. Schumann and H. Kojima, *"Power analysis of a 32-bit RISC integrated with a 16-bit DSP"*, SLPED'97.
3. H. Kojima, et al, *"Power Analysis of a Programmable DSP for Architecture/Program Optimization"*, Proceedings of the 1995 Low-Power Symposium.
4. V. Tiwari, S. Malik and A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Power Minimization", IEEE Trans. on VLSI Systems, 2(4):437-445, Dec. 1994.
5. Y. Sasaki, et al, *"Multi-Level Pass-Transistor Logic for Low-Power ULSIs"*, Proceedings of the 1995 Low-Power Symposium.
6. C. Tan, et al, *"Minimization of Power in VLSI Circuits Using Transistor Sizing, Input Ordering, and Statistical Power Estimation"*, Proceedings of the International Workshop on Low-Power Design, 1994.
7. M. Horowitz, et al, *"Low-Power Digital Design"*, Proceedings of the 1994 IEEE Symposium on Low-Power Electronics, 1994.
8. L. Gwennap, "RISC on the Desktop: A Comprehensive Analysis of RISC Microprocessors for PCs, Workstations, and Servers", MicroDesign Resources, 1994.
9. Microprocessor Report, several issues, MicroDesign Resources, Sebastopol, California.
10. R. S. Bajwa et al, "Instruction Buffering to Reduce Power in Processors for Signal Processing", IEEE Trans. VLSI Systems, 5(4):417-424, December 1997.

The Inherent Energy Efficiency of Complexity-Adaptive Processors

David H. Albonesi
Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627-0231
albonesi@ece.rochester.edu

Abstract

Conventional microprocessor designs that statically set the functionality of the chip at design time may waste considerable energy when running applications whose requirements are poorly matched to the particular hardware organization chosen. This paper describes how Complexity-Adaptive Processors, which employ dynamic structures whose functionality can be modified at runtime, expend less energy as a byproduct of the way in which they optimize performance. Because CAPs attempt to efficiently utilize hardware resources to maximize performance, this improved resource usage results in energy efficiency as well. CAPs exploit repeater methodologies used increasingly in deep submicron designs to achieve these benefits with little or no speed degradation relative to a conventional static design. By tracking hardware activity via performance simulation, we demonstrate that CAPs reduce total expended energy by 23% and 50% for cache hierarchies and instruction queues, respectively, while outperforming conventional designs. The additive effect observed for several applications indicates that a much greater impact can be realized by applying the CAPs approach in concert to a number of hardware structures.

1 Introduction

As power dissipation continues to grow in importance, the hardware resources of high performance microprocessors must be judiciously deployed so as not to needlessly waste energy for little or no performance gain. The major hardware structures of conventional designs, which are fixed at design time, may be inefficiently used at runtime by applications whose requirements are not well-matched to the hardware implementation. For example, an application whose working set is much smaller than the L1 Dcache may waste considerable energy precharging and driving highly capacitive wordlines and bitlines. Similarly, an application whose working set far exceeds the L1 Dcache size may waste energy performing look-ups and fills at multiple levels due to high L1 Dcache miss rates. The most energy-efficient (but not necessarily the best performing) cache organization is that which is well-matched to the application's working set size and access patterns. However, because of the disparity in the cache requirements of various applications, conventional caches often expend much more energy than required for the performance obtained. Other major hardware structures, such as instruction issue queues, similarly waste energy while operating on a diverse workload.

Complexity-Adaptive Processors (CAPs) make more efficient use of chip resources than conventional approaches by tailoring the complexity and clock speed of the chip to the requirements of each individual application. In [1], we show how CAPs can achieve this flexibility without clock speed degradation compared to a conventional approach, and thus achieve significantly greater performance. In this paper, we describe how CAPs can achieve this performance gain while expending considerably less energy than a conventional microprocessor.

The rest of this paper is organized as follows. In the next section, we discuss how the increasing use of repeaters in long interconnects creates the opportunity for new flexible hardware structures. Complexity-Adaptive Processors and then described in Section 3, followed by a discussion in Section 4 of their inherent energy efficiency. In Section 5, our experimental methodology is described. Energy efficiency results are discussed in Section 6, and finally we conclude and discuss future work in Section 7.

2 Dynamic Hardware Structures

As semiconductor feature sizes continue to decrease, to a first order, transistor delays scale linearly with feature size while wire delays remain constant. Thus, wire delays are increasingly dominating overall delay paths. For this reason, repeater methodologies, in which buffers are placed at regular intervals within a long wire to reduce propagation delay, are becoming more commonplace in deep submicron designs. For example, the Sun UltraSPARC-IIi microprocessor, implemented in a 0.25 micron CMOS process, contains over 1,200 buffers to improve wire delay [5]. Note that wire buffers are used not only in busses between major functional blocks, but within self-contained hardware structures as well. The forthcoming HP PA-8500 microprocessor, which is also implemented in 0.25 micron CMOS, uses wire buffers for the global address and data busses of its on-chip caches [3]. As feature sizes decrease to 0.18 micron and below, other smaller structures will require the use of

This research is supported in part by NSF CAREER Award MIP-9701915.

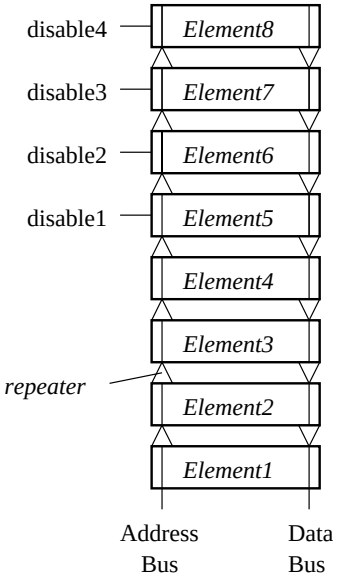


Figure 1: A dynamic hardware structure which can be configured with four to eight elements.

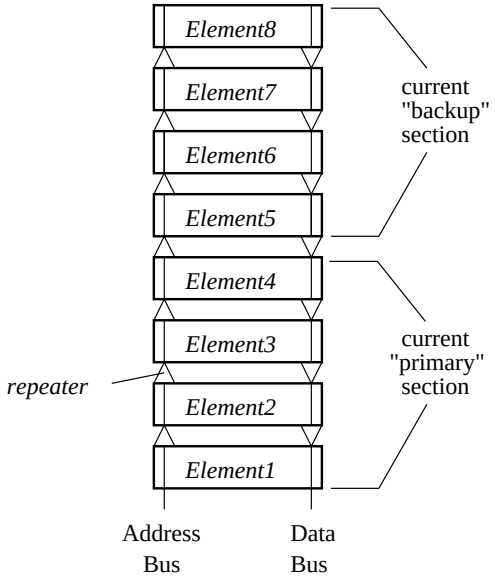


Figure 2: A dynamic hardware structure with configurable "primary" and "backup" sets of elements.

wire buffers in order to meet timing requirements [4].

For these reasons, we expect that many of the major hardware structures of future high performance microprocessors, such as caches, TLBs, branch predictor tables, and instruction queues, will be of the form shown in Figure 1. The hardware structure in this figure consists of replicated elements interconnected by global address/control and data busses driven using repeaters placed at regular intervals to reduce propagation delay. The isolation of individual element capacitances provided by the repeaters creates a distinct hierarchy of element delays, unlike unbuffered structures in which the entire wire capacitance is seen by every element on the bus. By employing address decoder disabling control signals as shown in Figure 1, we can make this structure *dynamic* in the sense that the complexity and delay of the structure can be varied as required by the application. For example, this structure can be configured with between four and eight elements, with the overall delay increasing as a function of the number of elements. Assuming this structure is on the critical timing path with four or more elements, if the clock frequency of the chip is varied according to the number of enabled elements[1], then the IPC/clock rate tradeoff of this structure can be varied at runtime to meet the dynamic requirements of the application. Due to their exploitation of repeater usage, such dynamic hardware structures can be designed with little or no delay penalty relative to a fixed structure.

An alternative to disabling elements is to use them as slower "backups" to the faster "primary" elements as is shown in Figure 2. Here, the difference between the primary and backup elements is the access latency in clock cycles. For example, the four primary elements in Figure 2 may be accessed in fewer cycles than the four backup elements, due to the latter's longer address and data bus delays. Such an approach may be appropriate, for example, for an on-chip Dcache hierarchy, in which the primary and backup elements correlate to L1 and L2 caches, and the division between them is determined as a function of the current working set size and the cycle time and backup element latency of each configuration.

[1] An alternative is to vary the latency in clock cycles of the structure.

3 Complexity-Adaptive Processors

Having discussed how repeater methodologies will lead naturally to the development of dynamic hardware structures, we now describe the overall organization of a Complexity-Adaptive Processor.

The overall elements of a CAP hardware/software system, shown in Figure 3, are as follows:

- Dynamic hardware structures as just described which can vary in complexity and timing (latency and/or cycle time);
- Conventional static hardware structures, used when implementing adaptivity is unwieldly, will strongly impact cycle time, or is ineffective due to a lack of diversity in target application requirements for the particular structure;
- Performance counters which track the performance of each dynamic structure and which are readable via special instructions and accessible to the control hardware;
- Configuration Registers (CRs), loadable by the hardware and by special instructions, which set the configuration of each dynamic structure as well as the clock speed of the chip; different CR values represents different complexity-adaptive configurations, not all of which may be practical;
- A dynamic clocking system whose frequency is controlled via particular CR bits; a change in these bits causes a sequence in which the current clock is disabled and the new one started after an appropriate settling period;

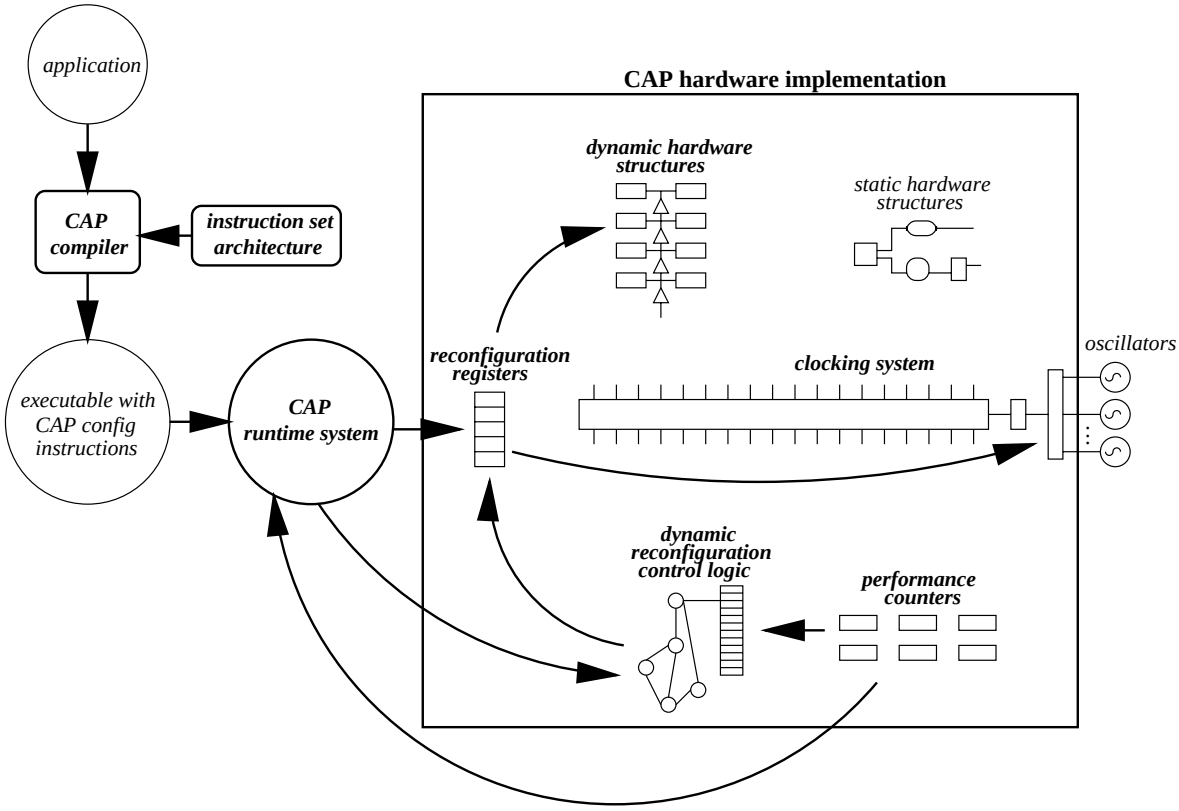


Figure 3: Overall elements of a CAP hardware/software system.

- The instruction set architecture (ISA) consisting of conventional instructions augmented with special instructions for loading CRs and reading the performance counters;
- Configuration control, implemented in the compiler, hardware (dynamic reconfiguration control logic), and runtime system, that acquires information about the application and uses predetermined knowledge about each configuration's IPC and clock speed characteristics to create a *configuration schedule* that matches the hardware implementation to the application dynamically during its execution.

The process of compiling and running an application on a CAP machine is as follows. The CAP compiler analyzes the application's hardware requirements for different phases of its execution. For example, it may analyze data cache requirements based on working set analysis, or determine the ILP based on the control flow graph. With this information, and knowledge about the hardware's available configurations, the compiler determines whether it can with good confidence create an effective configuration schedule, specifying at what points within the application the hardware should be reconfigured, and to which organizations. The schedule is created by inserting special instructions at particular points within the application that load the CRs with the desired configuration. In cases where dynamic runtime information is necessary to determine the schedule, this task is performed by the runtime system or the dynamic reconfiguration control logic. For example, TLB configuration scheduling may be best handled in conjunction with the TLB miss handler, based on runtime TLB performance monitoring, while the optimal branch predictor size may in some cases be best determined by runtime hardware. A major CAP design challenge is determining the optimal configuration schedule for several dynamic structures that interact with each other as well as with static structures, and which may be controlled by up to three different sources (compiler, runtime, and hardware).

Through these various mechanisms, the CRs are loaded at various points in an application's execution, resulting in reconfiguration of dynamic structures and changes in clock frequency. For runtime control, the performance counters are queried at regular intervals of operation, and using history information about past decisions, a prediction is made about the configuration that will perform best over the next interval. Assuming tens of cycles are required for each reconfiguration operation (based on the time to reliably switch clock frequencies), then an interval on the order of thousands of instructions is necessary to maintain a reasonable level of reconfiguration overhead.

4 Improving Energy Efficiency Via CAPs

There are two main aspects of CAPs that allow for reduced power consumption: the ability to disable or optimally allocate (between primary and backup sections) hardware elements, and the ability to control the frequency of the clock. One option is to explicitly manage these features in order to reduce power consumption.

For example, in a portable environment, when battery life falls below a certain threshold, a low power mode in which the processor is still fully functional can be enabled by setting all dynamic hardware structures to their minimum size, and selecting the slowest clock.

In addition, CAP designs have an inherent energy efficiency that is a byproduct of the way in which they optimize performance. Because the CAP hardware is configured to match the characteristics of each application, hardware structures are generally used more efficiently, and thus expend less energy, for a given task. It is this inherent energy efficiency of CAPs that follows naturally from optimizing performance that we explore in the rest of this paper.

5 Experimental Methodology

We examined the results of our preliminary performance analysis [1] of two-level on-chip Dcache hierarchies and instruction issue queues to estimate the relative energy expended by CAP and conventional approaches for these structures. For the Dcache hierarchy, we assumed a total of 128KB of cache, and for the CAP design, allowed the division between the L1 and L2 caches to be varied in steps of 8KB up to a total L1 Dcache size of 64KB on an application-by-application basis. We compared this approach with the best overall-performing conventional design: one with a 16KB 4-way set-associative L1 Dcache with the rest of the 128KB allocated to the L2 cache. We ran each benchmark on our cache simulator for 100 million memory references. The CAP instruction queue varied in size from 16 to 128 entries in steps of 16 entries. Unused entries were disabled. The best-performing conventional design contained 64 entries. We used the SimpleScalar simulator [2] and ran each benchmark for 100 million instructions. More details on the evaluation methodology and benchmarks used can be found in [1].

To estimate relative expended energy, we calculate the *activity ratio* for each approach by tracking the number and types of operations, and estimating the activity generated by each. For the two-level cache design, we track the number of L1 and L2 operations, and calculate the total number of cache banks activated considering the number activated for each operation and the L1/L2 configuration. We then take the ratio of the total activity for the CAP approach and for the conventional design. Because we use an exclusive caching policy, misses in the L1 Dcache to a valid location that hit in the L2 Dcache result in a swap between the two caches. In addition, the global miss ratio of the hierarchy remains constant due to the use of a random replacement policy. Thus, a CAP design that is optimized for performance in general also promotes energy efficiency by optimizing the amount of L1 Dcache allocated for a given application, and reducing L2 Dcache activity.

For the instruction queue, we did not have the ability to track the number of instruction queue accesses. However, because the rest of the design was almost ideal (large caches and queues, perfect branch prediction, plentiful functional units), we used the total number of cycles executed to approximate the relative number of instruction queue accesses for each benchmark and each approach (CAP and conventional). We then multiplied this number by the number of queue entries to get the total activity factor.

Although this approach is not exact, we believe that by tracking activity in this manner that we obtain a reasonable first-order approximation of relative expended energy.

6 Results

Table 1 shows event and activity counts as well as the activity ratio (CAP/conventional) for the ten benchmarks in which the CAPs configuration differs from the best conventional approach. The eleven benchmarks which use identical CAPs and conventional configurations are not shown as the expended energy is the same. The "L1 size (CAPs)" column denotes the CAPs configuration which performed best for each benchmark. The last column indicates the ratio of CAPs total activity to conventional total activity.

For five of the benchmarks, a CAPs configuration with an 8KB L1 Dcache outperforms the conventional approach using a 16KB L1 Dcache due to the former's faster cycle time, despite the increase in L1 misses and L1-L2 swaps incurred, as indicated in Table 1. Interestingly, the total activity count for the CAPs configuration is lower than the conventional approach. This is because for the conventional approach, twice as much L1 Dcache must be precharged and probed for each load operation (we make the simplifying assumption that only the selected way is activated on a store). This offsets the additional L2 probe and L1-L2 swap activity incurred with the CAPs configuration, and the fact that more L2 cache must be probed in the CAPs case on an L1 miss. This more efficient cache allocation reduces expended energy by 33% in the case of mgrid.

The tradeoff is different for the five remaining benchmarks where the CAPs L1 Dcache is larger than the conventional 16KB cache. Here, the reduction in L1 miss and L1-L2 swap activities must offset the additional L1 Dcache load activity for the CAPs configuration to expend less energy. This is true in all cases except for wave5, whose conventional cache experiences fewer total L1 misses than the other four benchmarks in this category. For benchmarks such as stereo and appcg whose requirements are not well-matched to the conventional organization, the energy savings with the CAP approach are significant: 44% for stereo and 62% for appcg. Because these benchmarks run significantly faster on the CAPs configuration as well [1], the reduction in the energy-delay product, an indicator of the efficiency with which a particular performance level is obtained, is even more striking: 70% for both benchmarks. Overall, 23% less energy is expended by the CAPs configuration for these benchmarks as a byproduct of performance optimization.

Table 2 shows the relevant data for the best-performing CAPs and conventional instruction queues for those benchmarks in which the CAPs and conventional configurations differ. Here, unused entries are disabled for the CAPs approach. In the cases in which the CAPs configuration performs best with fewer entries than the 64-entry conventional approach (due to the fact that the cycle time improvement overrides the IPC penalty incurred), this means that fewer elements are activated on each instruction queue access. However, more issue operations are required as the smaller window results in fewer instructions issued on average per issue operation. In all cases, the energy savings from acti-

benchmark	L1 size (CAP)	loads	stores	L1 misses		L2 misses	L1-L2 swaps		total activity		activity ratio
				conv	CAP		conv	CAP	conv	CAP	(CAP/conv)
m88ksim	8KB	64.6	35.4	2.48	2.97	2.41	2.43	2.48	370.5	261.6	0.71
compress	24KB	80.0	20.0	12.3	6.32	0.23	0.22	0.22	697.2	671.1	0.96
airshed	8KB	81.2	18.8	32.0	32.5	0.03	3.24	3.73	1275.0	1193.6	0.94
stereo	48KB	74.1	25.9	54.1	7.39	6.06	11.6	1.91	1910.2	1078.0	0.56
radar	8KB	61.2	38.8	1.51	4.20	0.50	0.87	3.54	328.8	295.5	0.90
appcg	64KB	4.84	95.2	12.0	0.49	0.47	11.9	0.49	474.6	181.9	0.38
swim	24KB	75.9	24.1	13.8	5.27	5.27	3.25	2.66	737.1	629.9	0.85
mgrid	8KB	95.4	4.60	4.55	4.64	4.12	1.45	1.45	511.7	344.9	0.67
applu	8KB	72.2	27.8	8.53	9.20	8.22	5.03	5.29	577.2	470.9	0.82
wave5	24KB	72.9	27.1	7.05	3.83	0.64	5.17	2.27	529.1	571.0	1.08
total									7411.3	5698.4	0.77

Table 1: Cache hierarchy event counts, total activity counts, and CAP/conventional activity ratio for each benchmark. All counts are in millions.

benchmark	IQ entries (CAP)	executed cycles		total activity		activity ratio
		conv	CAP	conv	CAP	(CAP/conv)
m88ksim	16	29.1	40.6	1862.6	649.1	0.35
compress	128	32.6	22.2	2088.9	2847.2	1.36
ijpeg	32	22.8	23.1	1461.9	738.0	0.50
airshed	32	23.7	25.0	1517.2	799.8	0.53
radar	16	110.1	141.8	7046.1	2268.9	0.32
appcg	16	48.4	49.8	3099.7	796.9	0.26
applu	128	27.6	19.8	1765.7	2535.0	1.44
fpppp	16	90.1	101.5	5766.3	1624.4	0.28
total				24608.5	12259.1	0.50

Table 2: Instruction queue executed cycles, total activity counts, and CAP/conventional activity ratio for each benchmark. All counts are in millions.

vating fewer elements overrides the energy cost of more queue accesses. This translates into as much as a 74% reduction in queue activity (in the case of appcg). Again, this benefit is achieved not through explicit power management, but simply as a byproduct of optimizing performance.

The opposite effect is observed for compress and applu which perform best with a larger 128-entry queue. The result is a significant increase in expended energy with the CAPs approach, despite the reduction in queue accesses. However, as these are the only two benchmarks using more entries than the conventional approach, the overall result is a 50% reduction in expended energy with the CAPs approach. Clearly, if more benchmarks performed best with more entries than the best average-performing configuration, then the energy savings would be less, perhaps significantly so. In addition, if configurations with more than 128 entries were available and some applications performed best with these configurations, then even a greater increase in expended energy would be incurred for these applications with the CAPs approach. However, it is likely that the inclusion of these applications into the benchmark suite would change the best-performing conventional approach to one with more entries. Thus, we expect that even with a wide range of benchmark behavior, that the CAPs approach will expend less energy overall due to its better use of hardware resources. However, the benefit is less clear with a structure in which elements are disabled than one in which the resources are allocated between primary and backup elements.

Examining Tables 1 and 2, we see that some applications, such as m88ksim, experience significant reductions in expended energy for both the CAP cache hierarchy and instruction queue. Thus, by applying the CAPs approach to other structures such as TLBs and branch predictors, we expect that a one to two order of magnitude reduction in expended energy is possible for applications whose hardware requirements are particularly poorly-matched to those in a conventional microprocessor design. A key point is that CAPs can achieve this improvement without adversely impacting the performance or expended energy of well-matched applications.

7 Conclusions and Future Work

The energy efficiency of a microprocessor is highly dependent on how well the hardware design matches the requirements of a particular application. In this paper, we have described how Complexity-Adaptive Processors inherently achieve energy efficiency as a byproduct of performance optimization by dynamically configuring their hardware organization to the problem at hand. By exploiting repeater methodologies used increasingly in deep submicron designs, CAPs achieve this benefit with little or no cycle time degradation over a conventional approach. By examining our performance results for a number of benchmarks, we found that the CAPs design reduced overall expended energy by 23% for cache hierarchies and 50% for instruction queues. We also discovered an additive effect for some benchmarks indicating that a much greater impact can be realized by applying the CAPs approach in concert to a number of hardware structures. In the future, we plan on obtaining results using more precise energy models for these and other hardware structures.

References

[1] D.H. Albonesi. Dynamic IPC/clock rate optimization. *Proceedings of the 25th International Symposium on Computer Architecture*, June 1998.

[2] D. Burger and T.M. Austin. The simplescalar toolset, version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[3] J. Fleischman. Private communication. November 1997.

[4] D. Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, 30(9):37–39, September 1997.

[5] K.B. Normoyle et al. UltraSPARC-IIi: Expanding the boundaries of a system on a chip. *IEEE Micro*, 18(2):14–24, March 1998.

Function Unit Power Reduction

Minimizing Floating-Point Power Dissipation Via Bit-Width Reduction

Ying Fai Tong, Rob A. Rutenbar, and David F. Nagle
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213
{yftong,bassoon}@ece.cmu.edu

Abstract

For many low-power systems, the power cost of floating-point hardware has been prohibitively expensive. This paper explores ways of reducing floating-point power consumption by minimizing the bit-width representation of floating-point data. Analysis of several floating point programs that utilize low-resolution sensory data shows that the programs suffer almost no loss of accuracy even with a significant reduction in bit-width. This floating point bit-width reduction can deliver a significant power saving through the use of a variable bit-width floating point unit.

1 Introduction

Floating point numbers provide a wide, dynamic range of representable real numbers, freeing programmers from the manual scaling code necessary to support fixed-point operations. Floating-point (FP) hardware is also very power hungry. For example, FP multipliers are some of the most expensive components in a processor's power budget. This has limited the use of FP in embedded systems, with many low-power processors not including any floating point hardware.

For an increasing number of embedded applications such as voice recognition, vision/image processing, and other signal-processing applications, FP's simplified programming model (vs. fixed-point systems) and large dynamic range makes FP hardware a useful feature for many types of embedded systems. Further, many applications achieve a high-degree of accuracy with fairly low-resolution sensory data. Leveraging these characteristics by allowing software to use the minimal number of mantissa and exponent bits, standard floating-point hardware can be modified to significantly reduce its power consumption while maintaining a program's overall accuracy.

For example, Figure 1 graphs the accuracy of CMU's Sphinx Speech Recognition System [Sphinx98] vs. the number of mantissa bits used in floating point computation. The left most point, 23 bits of mantissa, is the standard for a 32-bit IEEE FP unit. With 23 bits, the recognition accuracy is over 90%; but even with just 5 mantissa bits (labeled A), Sphinx still maintains over 90% word recognition accuracy. For Sphinx, there is almost no difference between a 23-bit mantissa and a 5-bit mantissa. In terms of power, however, a FP multiplier that uses only 5 mantissa bits consumes significantly less power than a 23-bit mantissa FP multiplier. This property, that programs can maintain accuracy while utilizing only a few bits for FP number representation, creates a significant opportunity for enabling low-power FP units.

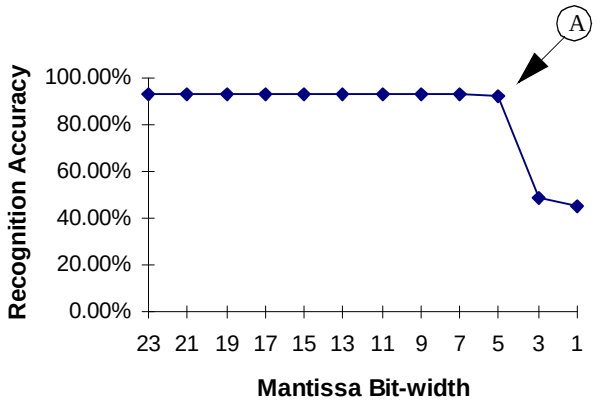


Figure 1: Accuracy of Sphinx Speech Recognition vs. Mantissa Bit-width

The goal of this work is to understand the floating-point bit-width requirements of several common floating-point applications and quantify the amount of power saved by using variable bit-width floating-point units. Section 2 begins our discussion by examining different aspects of the IEEE floating point standard that could lead to additional power savings. Section 3 presents the analysis of several floating point programs that require less bits than specified in the IEEE standard. In section 4, we describe the use of a digit-serial multiplier to design variable bit-width hardware and discuss the possible power savings. Finally, Section 6 outlines our conclusions and future work.

2 Background

2.1 IEEE 754 Floating Point Standard

One of the main concerns of the IEEE 754 Floating Point Standard is the accuracy of arithmetic operations. IEEE-754 specifies that any single precision floating point number be represented using 1 sign bit, 8 bits of exponents and 23 bits of mantissa. With double precision, the bit-width requirements of exponents and mantissa go up to 11 bits and 53 bits respectively.

In addition to specifying the bit-width requirement for floating point numbers, IEEE-754 incorporates several additional features, including delicate rounding modes and support for gradual underflow to preserve the maximal accuracy of pro-

This work was supported by the Defense Advanced Research Projects Agency under Order No. A564 and the National Science Foundation under Grant No. MIP90408457.

Dimension	Description
Reduction in mantissa bit-width	Reduce the number of mantissa bits at the expense of precision.
Reduction in exponent bit-width	Reduce the number of exponent bits at the expense of a smaller dynamic range.
Change of the implied radix	Increase the implied radix from 2 to 4 (or 16). This provides greater dynamic range but lower density of floating point numbers, potentially leading to power savings since fewer normalizing shifts are necessary.
Simplification of rounding modes	Full support of all the rounding modes is very expensive in terms of power. Some programs may achieve an acceptable accuracy with a modified low power rounding algorithm.

Table 1 : Design Dimensions for Floating Point Representation

grams. Nevertheless, the implementation of an IEEE compliant floating point unit is not always easy. In addition to the design complexity and the large area it occupies, a floating point unit is also a major consumer of power in microprocessors. Many embedded microprocessors such as the StrongARM [Dobberpuhl96] and MCore [MPR97] do not include a floating point unit due to its heavy implementation cost.

2.2 Accuracy Requirements and Workloads

For floating point applications that rely on sensory inputs, power savings can be obtained by modifying the floating point hardware's mantissa and exponent widths while maintaining sufficient accuracy for overall program execution. There are four conceivable dimensions that we can explore (see Table 1). Each of these dimensions allow us to make trade-off between program accuracy and the power consumption of the floating-point unit.

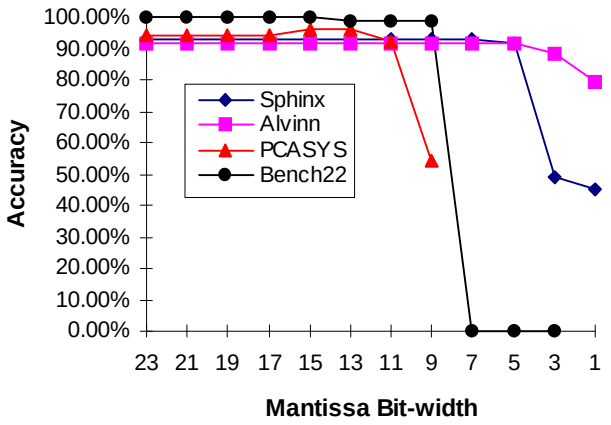


Figure 2: Program Accuracy across Various Mantissa Bit-widths

3 Experiments and Results

3.1 Methodology

To validate the usefulness and accuracy of reducing FP bit-widths, we analyzed four single-precision floating point programs (see Table 2). To determine the impact of different mantissa and exponent bit-widths, we emulated different bit-width FP units in software by replacing each floating-point operation with a corresponding function call to our floating-point software emulation package that initially implements the IEEE-754 standard. Careful modifications to the floating-point emulation package allowed us to simulate different mantissa and exponent bit-widths. For each bit-width, the emulation package was modified to use a smaller number of bits. Then, each program was run using the modified floating-point package and the results are compared to determine application accuracy.

3.2 Results

Figure 2 graphs the accuracy for each of the four programs across a range of mantissa bit-widths. None of the workloads displays a noticeable degradation in accuracy when the mantissa bit-width is reduced from 23 bits to 11 bits. For ALVINN and Sphinx III the results are even more promising; the accuracy does not change significantly with mantissa bit-width of 5 or more bits.

Workload	Description	Accuracy Measurement
Sphinx III	CMU's speech recognition program based on fully continuous hidden Markov models. The input set is taken from the DARPA evaluation test set which consists of spoken sentences from the Wall Street Journal. [Hwang94]	Accuracy is estimated by dividing the number of words recognized correctly over the total number of words in the input set.
ALVINN	Taken from SPECfp92. A neural network trainer using backpropagation. Designed to take as input sensory data from a video camera and a laser range finder and guide a vehicle on the road.	The input set consists of 50 road scenes and the accuracy is measured as the number of correct travel direction made by the network.
PCASYS	A pattern-level finger print classification program developed at NIST. The program classifies images of fingerprints into six pattern-level classes using a probabilistic neural network.	The input set consists of 50 different finger print images and the classification result is measured as percentage error in putting the image in the wrong class. The accuracy of the recognition is simply (1 - percentage error).
Bench22	An image processing benchmark which warps a random image, and then compares the warped image with the original one.	Percentage deviation from the original outputs are used as a measure of accuracy.

Table 2 : Description of Workloads

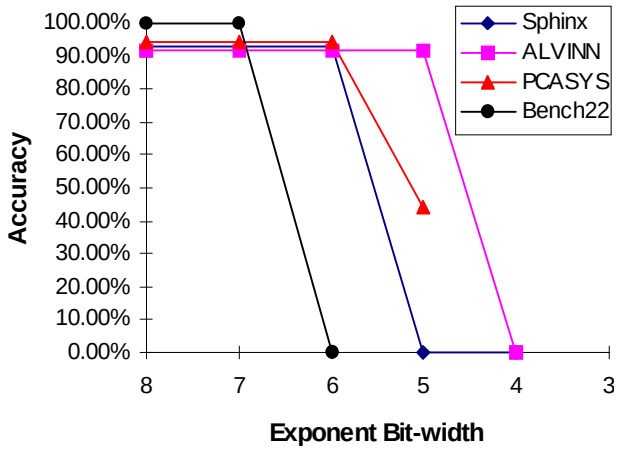


Figure 3: Program Accuracy across Various Exponent Bit-widths

Figure 2 and Figure 3 show that we can reduce both the mantissa and exponent bit-width without affecting the accuracy of the programs. This effect is especially prominent in the mantissa. This reduction of bit-width can be turned into a reduction in power dissipation with the use of appropriate arithmetic circuits

Figure 3 shows that each program's accuracy has a similar trend when the exponent bit-width is varied. With 7 or more exponent bits, the error rates remain quite stable. Once the exponent bit-width drops below 6, the error rates increase dramatically and in some cases the programs could not finish properly.

Many programs dealing with human interfaces process sensory data with intrinsically low resolutions. The arithmetic operations on these data may generate intermediate results that require more dynamic range, but not vastly more precision. This is different from many scientific programs such as wind tunnel simulation or weather prediction, which not only require a huge amount of precision and dynamic range but also delicate rounding modes to preserve the accuracy of the results.

For programs that do not need the dynamic range nor the precision of floating point arithmetic, the use of fixed-point arithmetic might well be a better choice in terms of chip space, operation latency, and power consumption. But for the programs we have analyzed, three of them require 6 bits or more of the exponents to preserve a reasonable degree of accuracy, which means they need more than the 32 bits of precision that fixed point arithmetic can offer. Simply using fixed point representation without additional scaling will not resolve the problem.

It should be noted that these complex applications were aggressively tuned by various software designers to achieve good performance using full IEEE representation. However, Figure 2 and Figure 3 suggest that significantly smaller bit-width FP units could be used by these applications without compromising the necessary accuracy. For instance, certain floating point constants in the Sphinx III code require more than 10 bits of mantissa to represent, but we modified those numbers so they can be represented using fewer bits during our experiment and yet this have little impact on the overall recognition accuracy. We believe that if the numerical behavior of these applications are adjusted to a smaller bit-width unit, we could get even better performance.

Multiplier	Area	Cycle Time	Latency/op
Wallace (24x24)	.777square mm	40ns	40ns
Digit-Serial (24x8)	.804 square mm	15ns	15ns

Table 3 : Timing and Area of the Two Multipliers

To perform 16 bits multiplication using the digit-serial multiplier, 2 cycles are needed which increases the total delay/op to 30ns. Similarly, 24 bits multiplication takes 3 cycles(45ns).

4 Power Savings by Exploiting Variable Bit-width Requirement

4.1 Multiplication with a Digit-Serial Multiplier

Since different floating point programs have different requirements on both the mantissa and exponent bit-width, we propose the use of a variable bit-width floating point unit[1] to reduce power consumption. To create hardware capable of variable bit-width multiplications (up to 24x24 bit), we used a 24x8 bit digit-serial architecture similar to the one described in Hartley and Parhi [Hartley95]. The 24x8 bit architecture allows us to perform 8, 16, and 24-bit multiplication by passing the data once, twice, or three times though the serial multiplier. A finite state machine is used to control the number of iterations through the CSA array.

To perform accurate power and timing measurements, the multiplier was described in Verilog and then taken to layout using our ASIC design flow (for a standard 0.5u process). Synopsys' Design Compiler was used to synthesize the multiplier's control logic. Next, the entire structural model was fed into Cascade Design Automation's physical design tool Epoch. A description of digit-serial arithmetic and the block diagram of the digit-serial multiplier can be found in the appendix.

We compare our variable bit-width multiplier with a baseline fixed-width 24x24 bit Wallace Tree multiplier. The layout of this Wallace Tree multiplier was generated by Epoch's cell generator in the same 0.5u process as used in the design of the digit-serial multiplier. The two multipliers are described in Table 3. Cycle time is estimated using Epoch's static timing analysis tool, Tactic, and is rounded to the nearest 5ns interval for the convenience of power simulation.

4.2 Power Analysis

For each design, a SPICE netlist was generated from layout and used to estimate power consumption with Avanti's StarSim. Determining the complete power dissipated in a multiplier requires the sensitization of all possible combinations of inputs, which means we need to have 2^{2N} input combinations where N is the number of inputs. Fortunately, it is possible to obtain a mean estimation of the power consumption using statistical techniques [Burch92]. In our approach, we ran 50 batches of vectors with each batch containing 30 vectors to insure a 95% confidence intervals. The energy dissipation is computed using the cycle time in Table 3. The test vectors are taken from some of the actual multiplication operands in Sphinx III.

Figure 4 graphs the energy/operation and latency/operation for the digit-serial multiplier. Both the energy/operation and latency/operation decrease linearly with the operand bit-width. This is different from [Callaway97], where a multiplier's power

1. Our current research focuses particularly on the floating point multiplier, since multipliers are usually the major consumer of power [Tiwari94].

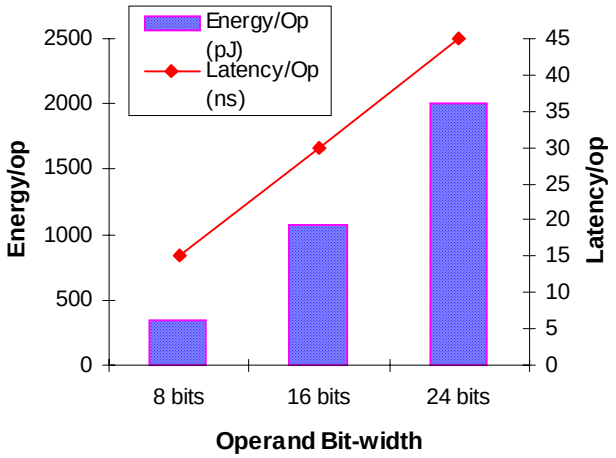


Figure 4: Performance of the Digit-Serial Multiplier

Both energy/op and latency/op of the digit-serial multiplier increase linearly with the operand bit-width. Digit-serial architecture allows us to perform variable bit-width arithmetic and save power when the bit-width requirement is less than that specified in the IEEE standard.

consumption decreases exponentially with the operand bit-width. The difference between the two results is due to the fixed structure (the 24x8 bit CSA array) of the digit-serial multiplier and the control circuitry needed to do iterative carry and sum reduction. This additional power dissipation is the penalty we pay for the flexibility of doing variable bit-width multiplication.

Figure 5 shows the potential power reduction for our three programs if we use the digit-serial multiplier as the mantissa multiplier. For 8-bit multiplication, the digit-serial multiplier consumes less than 1/3 of the power than the Wallace Tree multiplier (in the case of Sphinx and ALVINN). When 9 to 16 bits of the mantissa are required (in the case of PCASYS and Bench22), the digit-serial multiplier still consumes 20% less power than the Wallace Tree multiplier. The digit-serial multiplier does consume 40% more power when performing 24-bit multiplication due to the power consumption of the overhead circuitry.

Table 3 shows another benefit which is improved speed. When performing 8-bit or 16-bit multiplications, the operation's delay can be greatly reduced if the critical path of the circuit lies in the multiplier. This increases the throughput in addition to the energy saving.

4.3 Summary of Results

These power comparison results show the potential power savings achievable by using variable bit-width arithmetic units. It should be noted that the digit-serial multiplier is designed using an ASIC approach and is not as heavily optimized physically; the Wallace Tree multiplier was optimized for the process. This explains why the 24x8 bit digit-serial multiplier is actually slightly larger than the Wallace Tree multiplier. We believe that with a more careful implementation, both the power and area of the digit-serial multiplier can be reduced. In addition, [Chang97] presents several low power digit-serial architecture that can further reduce the power consumption of the digit-serial multiplier.

For software designers who know a program's precision requirements, code annotation could be used to allow the underlying arithmetic circuits re-configure themselves for variable bit-width operations. As Figure 4 and Figure 5 show, as long as the bit-width requirement is less than 16 bits, the 24x8 bit digit-

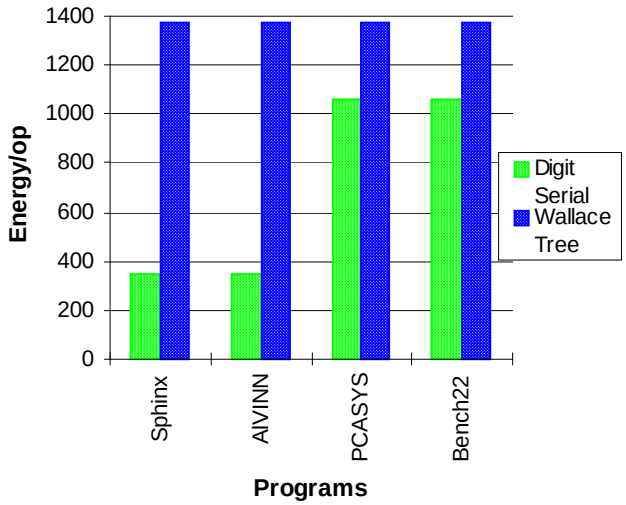


Figure 5: Power Reduction using Digit-Serial Multiplier

A reduction of 70% in energy/op is attainable for Sphinx and ALVINN with the use of digit-serial multiplier. Both Sphinx and ALVINN need only 5 bits of mantissa to be 90% accurate and thus an 8 bit operand bit-width is used for the digit-serial multiplier. PCASYS requires 11 bits of mantissa while Bench22 requires 9 bits and thus a 16 bit operand bit-width is used which results in a 20% energy/op reduction

serial multiplier consumes less energy than the Wallace Tree multiplier. Even for programs which require a large bit-width to maintain precision, there may be sections within the program that require a smaller bit-width requirement and thus can benefit from a variable bit-width unit.

5 Previous Work

The idea of reducing bit-width to save power has been employed in other areas of low power research. In [He97], it is shown that an average of more than 4 bits of pixel resolution can be dropped during motion estimation to obtain a power reduction of 70%. To reduce energy consumption of the I/O pins, [Musoll97] proposes sending only those address bits that have changed. For integer applications that do not need 32 bits of precision, the Intel MMX instructions allow arithmetic operations on multiple data simultaneously. Overall, the basic idea of bit-width reduction is to avoid waste.

6 Conclusions and Future Work

It is clear that floating point programs that use human sensory inputs may not need the entire 23 bits of mantissa and 8 bits of exponents as specified in the IEEE standard. Our software analysis shows no loss of accuracy across each of our programs when the mantissa bit-width is reduced to less than 50% of the original value. This large reduction in mantissa bit-width enables significant power reduction without sacrificing program accuracy. Further, the digit-serial multiplier results show that it is possible to obtain a substantial power saving by using a variable bit-width arithmetic unit.

Currently, we are looking at various digit-serial architectures and other means of exploiting the variable bit-width requirements of programs. Our future work will be directed towards investigating other characteristics of floating point programs

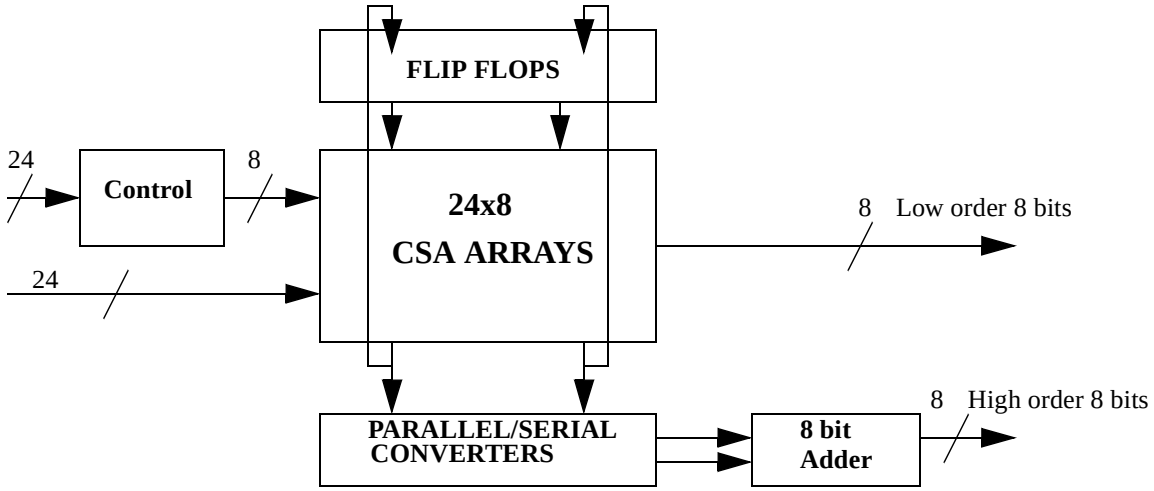


Figure 6: Block Diagram of a 24x8 Digit-Serial Multiplier

that may provide additional power savings. This includes using a simplified rounding algorithm and changing the implied radix.

7 Appendix

Arithmetic operations can be performed in different styles such as bit-serial, bit-parallel, and digit-serial. Bit-serial architecture processes data one bit at a time, saving hardware at the expense of processing speed. Bit-parallel circuits process all bits of the data operands at the same time with more hardware. Digit-serial arithmetic falls in between these two extremes by processing a fixed number of bits at one time. Figure 6 shows the block diagram of the digit-serial multiplier used in our experiment. For applications that require a moderate amount of throughput while having serious constraints on design space, digit-serial systems have become a viable alternative for many designers. Most of the digit-serial systems are used because of space limitation. Even though there is research on low-power digit-serial multipliers [Chang97], it has focused on comparing power consumption among different digit-serial architecture.

8 References

[Burch92] Burch R., Najm, F., Yang, P., Trick, T. *McPOWER: A Monte Carlo Approach to Power Estimation,* IEEE/ACM International Conference. on CAD, 1992.

[Callaway97] Callaway, Thomas K., Swartzlander, Earl E. *Power-Delay Characteristics of CMOS Multipliers*, IEEE 13th Symposium on Computer Arithmetic, 1997.

[Chang97] Chang, Y.N., Satyanarayana, Janardhan H., Parhi, Keshab K. *Design and Implementation of Low-Power Digit-Serial Multipliers*, IEEE International Conference on Computer Design, 1997.

[Dobberpuhl96] Dobberpuhl Dan. *The Design of a High Performance Low Power Microprocessor.* International Symposium on Low Power Electronics and Design, 1996.

[Hartley95] Hartley, Richard I., Parhi, Keshab K. *Digit-Serial Computation.* Norwell, Kluwer Academic Publishers, 1995.

[He97] He, Z.L., Chan, K.K., Tsui, C.Y., Liou, M. L., *Low Power Motion Estimation Design Using Adaptive Pixel Truncation*, International Symposium on Low Power Electronics and Design, 1997.

[Hwang94] Hwang, M., Rosenfeld, R., Theyer, E., Mosur, R., Chase, L., Weide, R., Huang, X., Alleva, F. *Improving Speech recognition performance via phone-dependent VQ codebooks and adaptive language models in SPHINX-II.* International Conference on Acoustics, Speech and Signal Processing, 1994.

[MPR97] *MCore Shrinks Code, Power Budgets*, Microprocessor Report11 (14), 27 October 1997.

[Musoll97] Musoll, E., Lang, T., Cortadella, J. *Exploiting the locality of memory references to reduce the address bus energy.* International Symposium on Low Power Electronics 1997.

[Sphinx98] SPHINX Group, Carnegie Mellon University, Pittsburgh, PA. *http://www.speech.cs.cmu.edu.speech/sphinx.html*

[Tiwari94] Tiwari, V., Malik, S. Wolfe, A. *Power Analysis of Embedded Software: a first step towards software power minimization*, IEEE Transactions on VLSI systems, vol.2, pp. 437-445, December 1994.

A Power Management for Self-Timed CMOS Circuits (FLAGMAN) and Investigations on the Impact of Technology Scaling

Thomas Schumann, Ulrich Jagdhold and Heinrich Klar

Abstract

One advantage of self-timed circuits is the fact that the performance of the chip depends on actual circuit delays, rather than on worst-case delays. This allows to exploit variations in operating temperature by reducing the power supply voltage to achieve an average throughput with a minimum of power consumption. Usually this average throughput is given by the clock frequency of the synchronous environment and a FIFO is used as a buffer interfacing the self-timed circuit to the synchronous environment.

Rather than adaptively adjusting the power supply voltage to the smallest possible value by using a state detecting circuit and a dc/dc converter additional to the FIFO [1], we propose to switch the supply voltage between two levels by using the Half-Full flag of the FIFO (FLAGMAN) [7]. Because two power supply voltages are already required on modern circuit boards for standard low-power ICs, e.g. µPs [2] and DSPs [3] for mobile systems, taking advantage of 2 supply voltages means zero-overhead for the board design.

By investigating real-life data of portable systems, as far as temperature variation is concerned, we give an analysis of power savings achieved by this technique. As a practical example we use a self-timed IC which already has been designed and fabricated: A 4bit carry-save multiplier [4]. It turns out that applying the proposed power management results in potential power savings of up to 40 % for that IC, depending on the corresponding operating temperature and the choice of the supply voltage levels.

1 Introduction

A self-timed circuit is defined as one where the transfer of data is not performed in synchrony with a global clock signal, but rather at times determined by the latencies of the pipeline stages themselves. Therefore communication signals (**Req**uest, **Ack**nowledge) between the pipeline stages are necessary. A handshake circuit is responsible for enforcing a protocol on these communication signals and the protocol ensures that the transfer of data only occur at current times. **Fig. 1** shows the principle of a pipelined self-timed circuit, each stage consisting of combinatorial logic in combination with storage elements and the handshake circuit.

Because the transfer of data is performed at times determined by the latencies of the pipeline stages, the throughput rate of the pipelined circuit depends on actual gate delays. This means that the throughput rate of an existing self-timed circuit depends on both, the supply voltage and the actual operating temperature (case temperature) of the

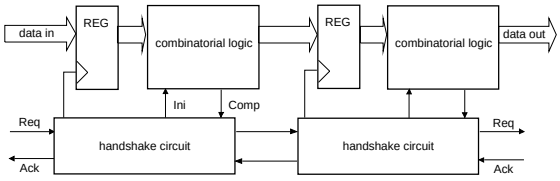


Fig. 1. A pipelined self-timed circuit

IC. This fact can be used to manage the power consumption in a very positive way: Assuming, the demanding average throughput rate is given by the clock frequency of the synchronous environment, variations in the operating temperature can be exploited by reducing the supply voltage rather then having longer idle times of the gates. Investigating the formula for the dynamic power consumption of one pipeline stage in a CMOS process:

$$P_{stage} = f_a \cdot C_L \cdot V_{DD}^2 \qquad (1)$$

where P_{stage} is the power consumption of one pipeline stage, f_a is the average operating frequency of the gates, C_L is the total switching capacitance of the stage, and V_{DD} is the supply voltage, it turns out that the supply voltage is the only parameter for optimization, assuming an existing circuit and a demand on the average operating frequency.

The proposed power management takes advantage of two supply voltages which are already required on modern circuit boards for portable systems. E.g. synchronous clocked, low-power µPs [2] are running on two supply voltages. The processor core operates at a reduced voltage internally while the I/O interface operates at the standard voltage 3.3 volts, which allows the processor to communicate with existing 3.3 volt components in the system. This trend is not only for µPs, but also for high-performance DSPs [3], developed for multi-function applications, e.g. wireless PDAs.

Section 2 illustrates the principle of the power management and gives a formula of the power reduction factor achievable using this technique. Section 3 focuses on an existing self-timed circuit, a 4bit array multiplier [4], and presents the power savings which can be obtained applying the power management. Measured data are compared to data calculated using the formula of the power reduction factor. Section 4 shows how technology scaling affects the achievable power savings.

2 Power Management

2.1 Principle

The proposed power management for self-timed circuits is shown in **Fig. 2**. The circuit is operating in a synchronous

T. Schumann and H. Klar are with the Institute of Microelectronics, Technical University of Berlin, D-10623 Berlin, Germany.
U. Jagdhold is with the Institute for Semiconductor Physics, D-15230 Frankfurt (Oder), Germany.

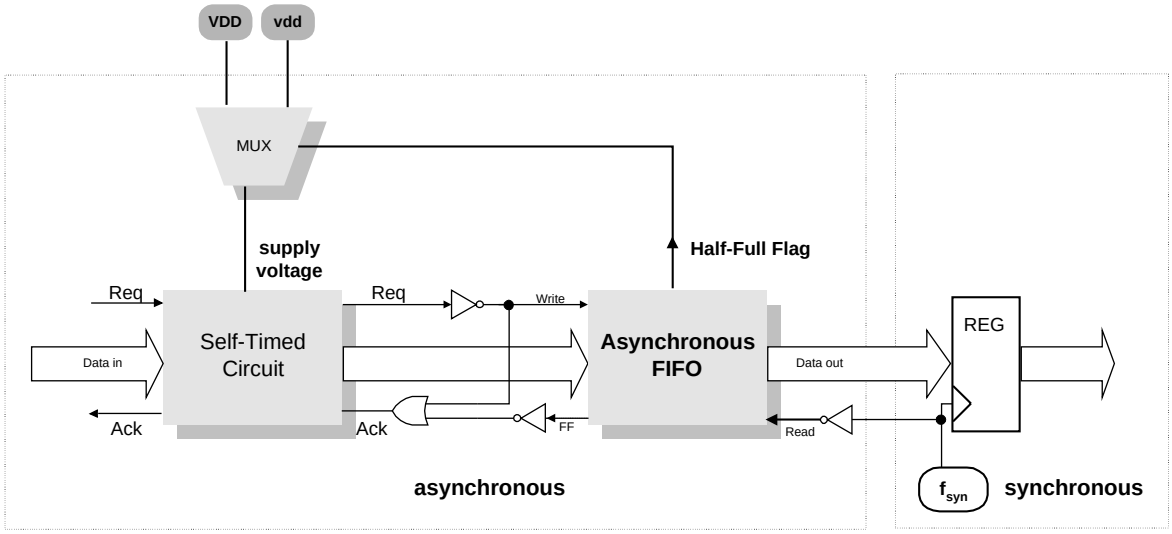


Fig. 2. Principle of the power management (FLAGMAN)

environment: The data stream input and output is controlled by the clock frequency f_{syn}.

Fig. 2 shows the output FIFO, working as a buffer interfacing the self-timed circuit to the synchronous environment.

The basic idea of the power management is to use the Half-Full flag of that FIFO as a signal which monitors the speed of the self-timed circuit: If the flag is set, the FIFO is running full and the supply voltage can be reduced. The output of the MUX changes to the reduced supply voltage.

If, although the reduced supply voltage is applied, the buffer runs full, the full-flag (FF) of the FIFO remains the acknowledge signal on the same logical state. This results in an idle time for the self-timed circuit. Even under worst case operating conditions the buffer must not become empty. This must be guaranteed when choosing the asynchronous circuit.

2.2 Power Reduction Factor

The basic idea, to slow down the operating frequency by switching to a second, reduced supply voltage which already exists on modern circuit boards, effects the power consumption in a very positive way. That is, interpreting (1), because the most important parameter controlling the power consumption in CMOS is the supply voltage. The power reduction factor R can be described by:

$$R(T) = \frac{P_{two\ \ voltages}(T)}{P_{fixed\ voltage}} \qquad (2)$$

where T is the case temperature of the IC.

In the following we are referring to the case temperature T as the operating temperature. The corresponding ambient and junction temperature can be calculated if the thermal resistances are known.

Using the basic equation (1) it follows:

$$R(T) = \frac{f_{vdd}(T)\cdot x(T)\cdot (b\cdot V_{DD})^2 + f_{VDD}(T)\cdot (1-x(T))\cdot V_{DD}^{\ 2}}{f_{syn}\cdot V_{DD}^{\ 2}} \qquad (3)$$

where $b\cdot V_{DD}$ is the reduced supply voltage, $f_{vdd}(T)$ is the throughput rate applying the reduced supply voltage under the actual operating temperature T, $f_{VDD}(T)$ is the throughput rate applying the standard supply voltage VDD under the actual case temperature T, f_{syn} is the clock frequency of the synchronous environment and x(T) the portion of the operating time during which the reduced supply voltage is applied.

To meet the requirement of the synchronous environment it must hold:

$$f_{syn} = f_{vdd}(T)\cdot x(T) + f_{VDD}(T)\cdot (1-x(T)) \qquad (4)$$

The power management is based on the fact that the gate delay in CMOS depends on both, supply voltage and operating temperature. The gate delay can be estimated [5] as

$$f_{VDD}(T) = \frac{I_{drain}}{C_L\cdot V_{DD}} = \frac{\beta(T)\cdot (V_{DD}-V_t(T))^2}{C_L\cdot V_{DD}} \qquad (5)$$

where $\beta(T)$ is the MOS transistor gain factor and $V_t(T)$ is the threshold voltage of the device.

Operating temperature influence circuit delays as follows:

$$\beta(T) = \beta_0\cdot (\frac{T}{T_0})^{-1,5} \qquad (6),$$

$$V_t(T) = V_{t0} - (4mV/°C)\cdot T \qquad (7)$$

where the exponent –1,5 and the factor 4 mV/°C are empirical values, reported in [6]. T_0 is the temperature at which the MOS device parameters β_0 and V_{t0} are defined.

For a given case temperature of the IC the power reduction factor depends on the chosen level of the second, reduced supply voltage. **Fig. 3** presents calculated data for a case temperature of 100 °C, assuming a self-timed circuit with a throughput rate 5 % higher than the clock frequency of the synchronous environment at 120 °C worst case temperature, operating at the standard 3.3 volts in a typical CMOS process with V_{t0} = 820 mV at 25 °C.

3 Application Chip

For applying the power management we use a self-timed IC which already has been designed and fabricated: a pipelined 4bit multiplier, using a modified micropipeline design style [4]. The asynchronous circuit has been implemented in a 1.2 µm CMOS process.

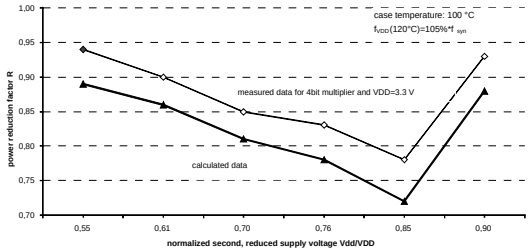


Fig. 3. Power reduction factor: calculated and measured data

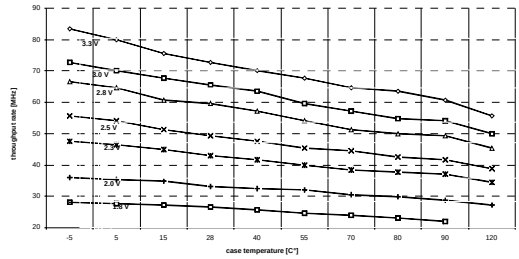


Fig. 4. Throughput rate of the self-timed multiplier applying different supply voltages

Fig. 4 shows data of the throughput rate for that specific design, over a wide range of temperature variations. The measurement was done by attaching a thermocouple bead to the center of the package top surface using highly thermally conductive cements.

Assuming a demand of 50 MHz on the average throughput, given by the clock frequency of the synchronous environment, **Fig. 5** presents the power savings achievable for that self-timed circuit by applying the proposed power management. It turns out that applying a very low second supply voltage is not the optimum power saving solution. This is because the operating time during which the reduced supply voltage is applied is smaller than the operating time for higher levels of the second supply.

For example, applying 1.8 V results in a power saving of 20 %, whereas applying 2.5 V results in 40 % power savings over a wide temperature range.

By applying the power management in a portable system which runs at different temperatures during the life-time of it battery we investigated potential power savings. In the following we are dealing with the impact of real-life data on the power reduction factor of the proposed power management. That a mobile system is running at different temperatures over the life-time of its battery is a well known fact. We found that the probability of running the system at temperature T can be described, to give a first order approximation for calculating a practical value for the expected power savings, by a Gaussian normal distribution:

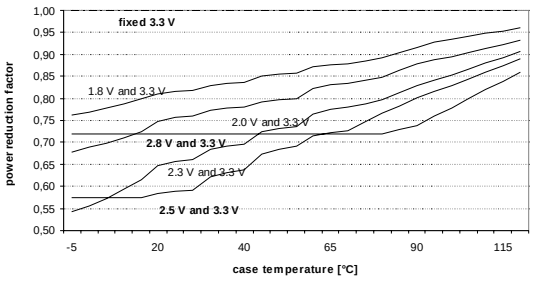


Fig. 5. Achievable power savings for the self-timed multiplier

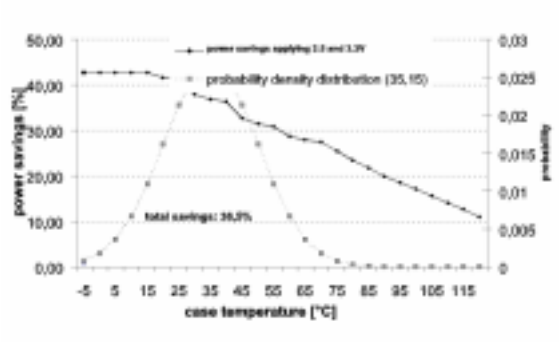


Fig. 6. Power savings dependence on the probability for running the IC at temperature T

$$f(T;\mu,\sigma) = \frac{1}{\sqrt{2\cdot\pi\cdot\sigma}}\cdot e^{-\left(\frac{(T-\mu)^2}{2\cdot\sigma^2}\right)} \qquad (8)$$

The average temperature μ is set to 35 °C, which means for our IC an ambient temperature of about 25 °C, assuming no airflow. The empirical variance σ is set to 15 °C, which means a wide range of operating temperature.

Fig. 6 shows both, the probability density distribution of running the IC at case temperature T, and the power reduction factor achievable by applying the power management on a circuit board with two supply voltages, 2.5 V and 3.3 V.

The expected total power savings for that IC, over a life-time of its battery, is about 36 %.

Tests of the circuit board confirm, that the Half-Full flag of a commercially available asynchronous FIFO monitors the speed of the selftimed multiplier.

Fig. 7 shows, that the supply voltage switches between two levels, as a result of the switching Half-Full flag. This is true if the on/off-delay of the MUX is less than 5.1 µs, assuming a 512 words FIFO and a 50 MHz clock frequency of the synchronous environment.

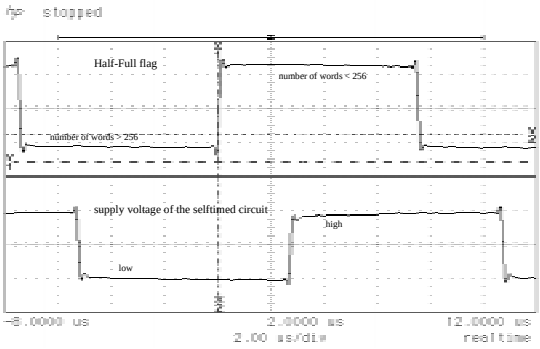


Fig. 7. Oszilloscope snap shot of Half-Full flag and supply voltage

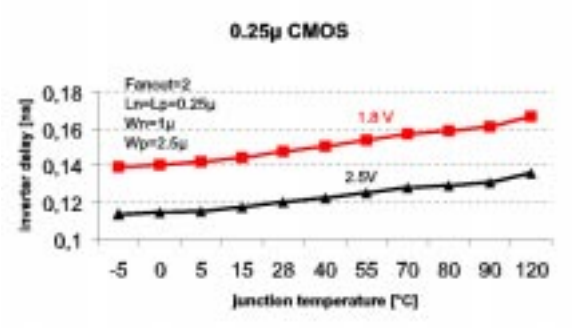


Fig. 8. Inverter delay versus junction temperature simulated in a 0.25µm CMOS process

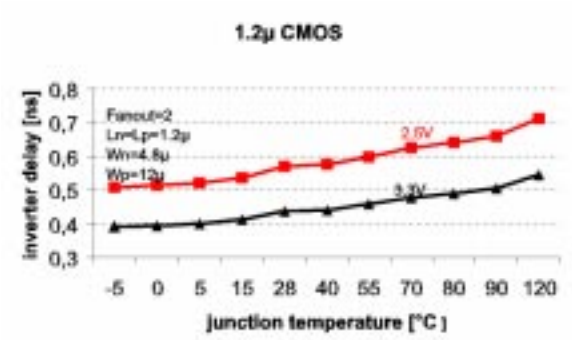


Fig. 9. Inverter delay versus junction temperature simulated in a 1.2µm CMOS process

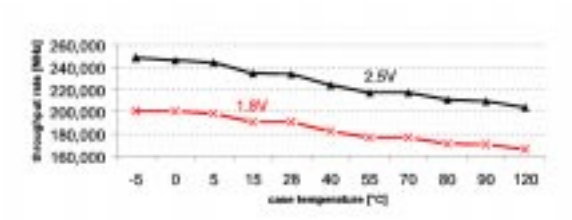


Fig. 10. Expected throughput rate of the self-timed multiplier implemented in a 0.25µm process

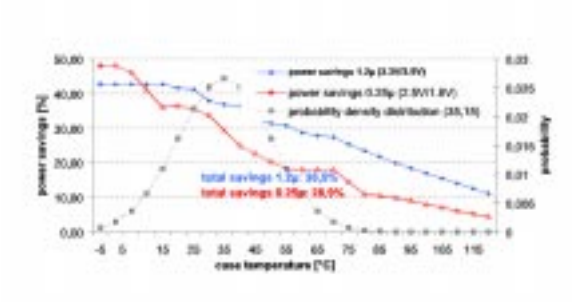


Fig. 11. Impact of technology scaling on the achievable power savings for the self-timed multiplier

4 Impact of Technology Scaling on Power Savings

The impact of technology scaling on the possible power savings is investigated by SPICE simulations. A standard 0.25 µm CMOS process, n-well, 5.5 nm gate oxide thickness, 0.5 V threshold voltage of the NMOS and -0.26 V threshold voltage of the PMOS device, is considered. The technology is optimized for a supply voltage of 2.5 V.

Fig. 8 shows data from SPICE simulations: The inverter delay (fanout = 2) for this 0.25 µm process is plotted versus the junction temperature of the device. For comparison **Fig. 9** shows simulation results for the 1.2 µm CMOS process, which has been used for the implementation of the self-timed multiplier. It turns out, that the variation of the gate delay between normal case (28 °C) and worst case (120 °C) temperature is for the 0.25 µm process 51 % less than for the 1.2 µm process.

So the question arises how much this affects the power savings applying the proposed power management. Considering the simulated inverter delays we calculated the throughput rate, the self-timed multiplier would perform, if it would be implemented in the high performance 0.25 µm process. **Fig. 10** shows data of the throughput rate applying 2.5 V and 1.8 V, respectively. It turns out that even under worst case condition a clock frequency of 200 MHz for the synchronous environment could be obtained by applying the standard 2.5 V supply to the self-timed IC. Under normal operating conditions the power management is responsible for switching the supply voltage to 1.8 V for a time interval, to achieve 200 MHz on average. This means 35 % power savings for a case temperature of 28 °C. **Fig. 11** shows data of the calculated power savings applying the proposed power management to the self-timed multiplier, implemented in the quarter micron CMOS process. An equal probability density distribution of running the IC at case temperature T as for the 1.2µm process is assumed.

Implementing the multiplier in a 0.25 µm CMOS process and applying the proposed power management results in expected power savings of about 28 %.

5 Conclusion

A power management for self-timed CMOS circuits is proposed, which takes advantage of the existence to two supply voltages on modern circuit boards.

The effectiveness of the power management is investigated by applying the technique to a self-timed multiplier IC, implemented in a 1.2 µm CMOS technology.
It also turns out that the power management results in significant power savings of up to 40 % by making use of the fact that the performance of a self-timed circuit depends on actual gate delay, rather than on worst case delay. Investigations also show, that the power savings for a deep sub-micron technology is reduced in comparison to a 1.2 µm technology, caused by the reduced influence of the junction temperature on the gate delay.

Acknowledgement

The authors would like to thank IDT, Inc. and MAXIM, Inc. for providing us with free samples of standard ICs for board design. They would also like to thank Ayman Bobo for his help designing and testing the circuit board.

References

[1] L. S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage", *IEEE Trans. on VLSI Systems*, vol.2, no. 4, pp. 391-397, Dec. 1994.
[2] intel®, "The mobile Pentium® processor fact sheet", http://www.intel.com/design/mobile, Aug. 1997.
[3] Texas Instruments®, "TMS320C6x Device Features and Applications", http://www.ti.com/sc/docs/dsps, Sept. 1997.
[4] T. Schumann, H. Klar, and M. Horowitz,"A fully asynchronous, high-throughput multiplier using wave-pipelining for DSP custom applications", *Mikroelektronik für die Informationstechnik, ITG-Fachtagung,*Germany, pp. 283-286, 1996.
[5] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design", *IEEE J. Solid-State Circ*., vol. 27, no. 4, pp. 473-484, Apr. 1992.
[6] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Reading, MA: Addison-Wesley, 2nd ed., 1992, pp. 61-69.
[7] T. Schumann and H. Klar,"Power Management for self-timed CMOS circuits using the half-full flag of an asynchronous FIFO", *Mikroelektronik für die Informationstechnik, ITG-Fachtagung,* Germany, 1998.

Hybrid Signed Digit Representation for Low Power Arithmetic Circuits

Dhananjay S. Phatak

Steffen Kahle, Hansoo Kim and Jason Lue

Electrical Engineering Department
State University of New York
Binghamton, NY 13902-6000
email: phatak@ee.binghamton.edu

Abstract

The Hybrid Signed Digit (HSD) representation introduced in [9] employs both signed and unsigned digits and renders the maximum length of carry propagation equal to the maximum distance between two consecutive signed digits. This representation offers a continuum of choices from two's complement representation on one extreme (where there are no signed digits) all the way to the conventional fully signed digit (SD) representation on the other extreme, wherein every digit is signed. The area-delay tradeoffs associated with each of the HSD formats have been analyzed in [9]. Possibility of reducing power consumption by utilizing the HSD representation was also mentioned therein.

This paper investigates the impact of the HSD representation on power consumption. Adder based on a well known fully SD implementation [2] (which is adopted in a microprocessor [4]) is compared with the adder based on HSD representation. Layouts of basic adder cells based on SD and HSD representations are *exhaustively* simulated to obtain the average number of transitions and power dissipation per addition. The results are then used to estimate power dissipation in adders of various word lengths and indicate that adopting HSD representation can lead to a reduction in power consumption over a fully SD representation.

The power (and area) reduction is achieved as a result of algorithm level change. This leaves room for additional power reduction by employing standard techniques at lower levels of the design process (such as circuit level, layout level etc). Furthermore, The HSD representation offers a continuum of choices to the designer. By increasing the distance between the signed digits, lower power dissipation can be achieved at the expense of slower speed. Thus, if the worst case delay is predetermined, the designer can select a hybrid representation that minimizes power dissipation under the delay constraint.

1 Introduction

The well known signed–digit (SD) number representation [1, 9, 10] makes it possible to perform addition with carry propagation chains that are limited to a single digit position, and has been used to speed up arithmetic operations. The advantages of SD representation can be described by drawing an analogy with the transforms (Fourier or Laplace transforms, for instance): certain problems of sufficient complexity make it worthwhile to pay the overhead of forward and reverse transforms because the advantages/gain in the transform domain more than offset the overhead. SD representation is analogous to a transformed domain wherein the addition is carry free. Moreover, the forward transform, (i.e., converting normal numbers into signed digits) is trivial. Only the reverse transform requires a delay equivalent to a full addition (which depends on the word length). If multiple operands are to be added, then going to an intermediate signed digit representation is advantageous because the additions are carry free. There is only one full carry propagation operation required to convert the final sum back to two's complement format. SD representation has been exploited to speed up most types of arithmetic circuits (multipliers, dividers, CORDIC processors, etc).

A carry save adder tree based on (3,2) counters achieves a reduction from 3 to 2 operands at each level. Hence the number of levels required for adding n operands is $\lceil \log_{\frac{3}{2}} \frac{n}{2} \rceil = \lceil \frac{\log n - \log 2}{\log 3/2} \rceil$ where $\lceil x \rceil$ denotes the ceiling of x, i.e., smallest integer greater than or equal to x and $\log z$ without any base indicates log to base 10. A signed digit tree, on the other hand needs $\lceil \log_2 n \rceil$ levels. If the delay associated with a carry-save stage is Δ_{FA} and that associated with a SD stage is Δ_{SD}, then the SD tree is (asymptotically) faster if $\frac{\Delta_{SD}}{\Delta_{FA}} \leq \frac{\log 2}{\log(3/2)} \approx 1.71$. Signed–digit adder trees are easier to lay out and route than Wallace trees [2]. In [2] a 64×64 bit multiplier based on a redundant signed–digit binary adder tree was shown to yield a smaller critical path delay than the corresponding Wallace tree multiplier with booth recoding. A similar design for a fast 54×54 bit multiplier was published in [3]. The cost for the speedup is higher area, since every (binary) signed digit needs two bits for encoding.

The advent of mobile computing favors circuits with low power dissipation. Hence, combined analysis of speed and power tradeoffs is attracting increasing amounts of research efforts [5, 6]. All the number representations considered in [5, 6] were special cases of the GSD representation [8].

In [9] a Hybrid Signed Digit (HSD) representation was introduced. It employs both signed and unsigned digits and reveals a continuum of choices of number representations between the extremes of fully SD on one hand to two's complement on the other. It is outside the GSD framework (but overlaps it) as shown in Figure 1. The area-delay tradeoffs associated with each representation were analyzed in [9] and show that a multiplier

based on HSD-1 representation (i.e., alternate digit positions are signed) has lower AT product than multipliers that employ fully SD trees. Possibility of reducing power consumption by utilizing the HSD representation was mentioned in [9].

This paper investigates the impact of the HSD representation on power consumption. Adder based on a well known fully SD implementation [2] (which is adopted in a microprocessor [4]) is compared with the adder based on HSD representation. Layouts of basic adder cells based on SD and HSD representations are *exhaustively* simulated to obtain the average number of transitions and power dissipation per addition. The results are then used to estimate power dissipation in adders of various word lengths and indicate that adopting HSD representation can lead to a reduction in power consumption ($\approx$ 11% or more depending on the distance between the signed digits) over a fully SD representation.

The power (and area) reduction is achieved as a result of algorithm level change. This leaves room for additional power reduction by employing standard techniques at lower levels of the design process (such as circuit level, layout level etc). Furthermore, The HSD representation offers a continuum of choices to the designer. By increasing the distance between the signed digits, lower power dissipation can be achieved at the expense of slower speed. Thus, if the worst case delay is predetermined, the designer can select a hybrid representation that minimizes power dissipation under the delay constraint.

Next section summarizes the preliminaries, describing the HSD representation. Section III derives the average number of transitions and power dissipation per addition in the basic adder cells based on SD and HSD representations. These averages are determined via exhaustive simulations of all possible cases and are used to estimate the power dissipation in adders of different lengths, and utilizing different HSD formats. Last section discusses the implications and presents conclusions.

2 Preliminaries: The HSD representation

Without loss of generality, we consider radix-2 HSD representation for the sake of illustration. HSD formats are illustrated in Figure 1. For radix $r = 2$ the signed digits can take any value in the set $\{-1, 0, +1\}$. The unsigned digits are like normal bits and can assume any of the two values $\{0,1\}$. The addition procedure presented in [9] enables a signed digit to stop an incoming carry from propagating further. Consequently, the carries propagate between the signed digits and the maximum length of carry propagation equals the distance between the signed digits. It can be verified that addition in such a representation requires the carry in between all digit positions (signed or unsigned) to assume any value in the set $\{-1, 0, 1\}$ as in the SD system. The operations in a signed–digit position are exactly the same as those in the SD case. For instance, let x_i and y_i be radix–2 signed digits to be added at the ith digit position, and c_{i-1} be the carry into the ith digit position. Each of these variables can assume any of the three values $\{-1, 0, 1\}$. Hence $-3 \leq x_i + y_i + c_{i-1} \leq +3$. This sum can be represented in terms of a signed output z_i and a signed carry c_i as follows:

$$x_i + y_i + c_{i-1} = 2c_i + z_i \qquad (1)$$

where $c_i, z_i \in \{-1, 0, 1\}$. In practice, the signed–digit output z_i is not produced directly. Instead, the carry c_i and an intermediate sum s_i are produced in the first step, and the summation $z_i = s_i + c_{i-1}$ is carried out in the second. Following the procedure in [9] guarantees that the second step generates no carry.

The operations in an unsigned digit position are as follows. Let a_{i-1} and b_{i-1} be the bits to be added at the $(i-1)$th digit position; $a_{i-1}, b_{i-1} \in \{0, 1\}$. The carry into the $(i-1)$th position is signed and can be $-1, 0$ or 1. The output digit e_{i-1} is restricted to be unsigned, i.e., $e_{i-1} \in \{0, 1\}$. Hence the carry out of the $(i-1)$th position must be allowed to assume the value -1 as well. In particular

$$\begin{array}{l} \text{if} \quad (a_{i-1} = b_{i-1} = 0 \quad \& \quad c_{i-2} = -1) \ \text{then} \\ \quad c_{i-1} = -1 \quad \text{and} \quad e_{i-1} = 1 \\ \text{else} \\ \quad a_{i-1} + b_{i-1} + c_{i-2} = 2c_{i-1} + e_{i-1} \\ \qquad \text{where} \ \ c_{i-1}, e_{i-1} \geq 0 \\ \text{endif} \end{array} \qquad (2)$$

The signed–digit positions generate a carry–out and an intermediate sum based only on the two input signed digits and the two bits at the neighboring lower order unsigned digit position. In the second step, the carries generated out of the signed digit positions ripple through the unsigned digits all the way up to the next higher order signed digit position, where the propagation stops. All the (limited) carry propagation chains between the signed digit positions are executed simultaneously.

The most significant digit in any HSD representation must be a signed digit in order to incorporate enough negative numbers. All the other digits can be unsigned. For example, if the word length is 32 digits, then, the 32nd (i.e., the most significant) digit is a signed digit. The remaining digits are at the designer's disposal. If regularity is not necessary, one can make the 1st, 2nd, 4th, 8th and 16th (and 32nd) digits signed and let all the remaining digits be unsigned digits (bits). The addition time for such a representation is determined by the longest possible carry–propagation chain between consecutive signed digit positions (16 digit positions; from the 16th to the 32nd digit in this example).

The HSD representation has another interesting property: there is no need to be restricted to a particular HSD format (with a certain value of d). The representation can be modified (i.e., the value of d can be changed) while performing addition (and consequently, other arithmetic operations) and this can be done in certain cases *without any additional time delay*. For instance, let x and y be two HSD operands, with uniform distances d_x and d_y, respectively, between their signed digits. Also assume that $(d_y + 1)$ is an integral multiple of $(d_x + 1)$ so that the signed digit positions of y are aligned with the signed digit positions of x (note that x has more signed digits than y under the stated assumption). Let $z = x + y$ be their sum, having a uniform distance d_z between its signed digits. The possible values of d_z which are interesting from a practical point of view are $0, d_x$ and d_y. If we set $d_z = d_y$ then the above addition will take exactly the same time as an addition of two HSD operands with uniform distance d_y producing an HSD result with distance d_y. Setting $d_z = d_x$ (and clearly, $d_z = 0$) will reduce the addition time even further, since the introduction of extra signed digits results in shorter carry propagation chains. For example, sup-

pose that $d_x = 0$ (all digits are signed) and $d_y = 1$ (alternate digits are signed). If d_z equals 1, then the delay required to perform the addition is the same as that required to add two HSD numbers with the same distance $d = 1$ to generate an output with $d_z = 1$. This format conversion flexibility (without any additional time delay penalty) can be useful, as illustrated by the multiplier design in [9].

3 Power consumption of basic adder cells

The adders considered are static CMOS circuits since these typically consume lower power than dynamic CMOS designs. The basic building block for the signed digit adders in [2, 4] is the efficient signed digit adder cell illustrated in [2]. The cell takes 2 signed digit operands and a signed carry as inputs and generates a signed digit sum a signed carry output. Since each variable takes two bits to encode, the cell has 6 inputs and 4 outputs and requires 42 transistors.

Building blocks of the HSD adder are the cells in signed and unsigned digit positions presented in [9]. The signed cell takes 2 signed digits and one signed carry and generates a signed digit output and carry, and has 6 inputs and 4 outputs and also requires 42 transistors. Cell in the unsigned digit position takes 2 unsigned digits (bits) and a signed carry as inputs and generates a signed carry and unsigned sum bit as the outputs. It has 4 inputs and 3 outputs and requires 34 transistors.

All cells were layed out using the MAGIC editor with the default scalable CMOS technology files. The average number of transitions as well as the average power consumption (per addition) of these cells was estimated both analytically as well as through an exhaustive simulation with IRSIM as described next.

Transition count via exhaustive testing of combinational circuits

Assume the circuit has n inputs. Then there are 2^n possible input patterns labeled P_0 through P_N where $N = 2^n - 1$.

For exhaustive testing: set the current input to P_0 and set next input to each of $P_1, P_2, \ldots, P_N$ and count transitions for each case. Do this for each $P_i, i = 0, \ldots, N$, accumulating transitions and power dissipation for each iteration. Dividing the accumulated sums by $N(N-1)$ (which is the total number of iterations) yields the (exhaustive) average number of transitions per input pattern applied to the circuit.

To theoretically estimate the number of transitions in the circuit, one needs the total number of transitions at the inputs. This number is determined as follows. Let the current input be P_i. We exhaust all possible next inputs $P_j, j \neq i$ and repeat this process for all possible i values. It turns out that the number of transitions on the inputs during exhaustive testing of P_i are the same for all i.

Let $P_i = (x_1 \ldots x_n)$. The number of patterns P_j that differ from P_i in exactly one position is ${}^nC_1 = \frac{n(n-1)}{1 \cdot 2}$ and each of these contributes 1 transition. Similarly the number of patterns differing from P_i in exactly k positions is ${}^nC_k = \frac{n(n-1)\cdots(n-k+1)}{1 \cdot 2 \cdots k}$ and each contributes k transitions. Hence the total number of transitions (on all inputs) in exhaustive testing of P_i is

$$\sum_{k=1}^{n} k \cdot {}^nC_k = n \cdot 2^{n-1} \tag{3}$$

The last equality can be derived by differentiating the expression for $(1+x)^n$ and setting $x = 1$ on both sides of the resulting identity.

Given $0 \leq i < 2^n$, the total number of transitions is

$$2^n \cdot (n \cdot 2^{n-1}) = n \cdot 2^{2n-1} \tag{4}$$

or the number of transitions for any single input is 2^{2n-1} since all inputs are symmetric.

Using the zero delay model in [7], the expected number of transitions $E[n_y(T)]$ at the output y of a logic module in time interval T is

$$E[n_y(T)] = \sum_{i=1}^{n} P(\frac{\partial y}{\partial x_i}) \cdot E[n_{x_i}(T)] \tag{5}$$

where $x_i, i = 1 \cdots n$ are the inputs to the module and $P(\frac{\partial y}{\partial x_i})$ is the probability that the Boolean difference $\frac{\partial y}{\partial x_i} = (y|_{x_i=1}) \oplus (y|_{x_i=0})$ is 1.

Using the total number of input transitions in exhaustive testing from equation (4) in the above equation and assuming that the primary inputs of the cells are independent leads to the the analytical estimates of the average number of transitions per addition for each of the cells that are shown in Table 1.

	Theoretical estimate	Simulation outputs	
	Average Transitions	Average Transitions	Average Power Dissipation ($\times 10^{-5}$ Watts)
Signed digits adder cell based on SD rep. [2]	7.94	11	8.171
Signed digits adder cell based on HSD rep. [9]	7.94	12	8.291
Unsigned digits adder cell based on HSD rep. [9]	7.46	9	5.126

Table 1: Comparison of average number of transitions and power dissipation in the basic cells. The simulation average was determined by exhaustively testing all possible input cases, excluding the don't care input combinations.

The simulator used was IRSIM 9.3. It counts transitions at ALL internal nodes and takes into account any glitches and/or extraneous transitions. Hence the number of transitions counted by the simulator are higher than the analytical estimates. The signed digit cells are designed assuming that 0 is encoded as "00", 1 as "01" and a -1 as "11". Hence, it was assumed that the input combination "10" would never occur as a signed digit

and was exploited as a "don't care" to simplify the logical expressions and arrive at the final cell design. These input combinations are therefore excluded from the simulations.

4 Power dissipation of SD and HSD adders

Exhaustive determination of average number of transitions and power dissipation per addition is impossible for adders of practical word lengths (32, 53 or 64). In fact, even a 3 digit SD adder has 14 inputs ($3 \times 2 = 6$ signed digits requiring 12 bits plus the two bits representing the carry-in into the least significant position). The number of cases to be considered (for exhaustive determination of averages) is $\approx 2^{142} = 2^{28}$ which is prohibitively large (even if don't care cases are excluded the exact count is $(3^6 \times 4)(3^6 \times 4 - 1) = 8500140$ or over 8.5 million). For 2 digit adders, however, the number of cases (excluding the "don't care" input combinations) is 104,653 which is manageable.

In the HSD-1 format, alternate digits are signed (the rest are unsigned). Hence the total power consumed by the cascade of one unsigned and one signed cells should be compared with the total power consumed by the cascade of two SD cells (each of these cover 2 digit positions in their respective representations). Thus, difference between HSD-1 and SD formats can be understood by considering two digit positions of each, which is another reason behind exhaustively simulating adders of word length 2. The results are shown in Table 2.

	number of cases	Average Transitions	Average Power Dissipation ($\times 10^{-5}$ Watts)
Cascade of two SD cells [2]	104653	23 (22)	14.36 (16.34)
Cascade of Signed (higher significant) and Unsigned HSD cells [9]	20591	21 (21)	12.81 (13.42)

Table 2: Comparison of average number of transitions and power dissipation in two-digit adders (word length of two digits). The number of cases excludes the patterns in which any of the signed digits gets the impossible input combination "10". The values in parenthesis are obtained by adding the corresponding values for individual cells from Table 1.

Comparison of Table 1 and Table 2 allows a testing of the "independence hypothesis". The carry signals between the adjacent cells are not independent and hence the average power dissipated by a group of cells may not be estimated simply as a sum of the average powers dissipated by the individual cells. This is seen in Table 2, where the values in parenthesis (which are the sums of average transitions and power dissipated by individual cells) differ from the values generated by actual simulation. The difference, however, is moderate (about 4.5% in the number transitions and 13% in the power dissipation values).

For word lengths beyond 2 digits, the power dissipation estimates must is obtained by extrapolating the results for two digit cascades. This is more accurate than extrapolating the results from individual cells. The estimates for word lengths of 24, 32, 53 (the number of digits in double precision floating point representation prescribed in IEEE 754 standard for floating point implementations) and 64 bits are summarized in Table 3.

Word Length	Estimate of Average Transitions per addition		Estimate of Average Power Dissipation per addition ($\times 10^{-5}$ Watts)	
	SD	HSD–1	SD	HSD–1
24	276	252	172.3	153.72
32	368	336	229.8	204.96
53	609.5	556.5	380.5	339.465
64	736	672	459.5	409.92

Table 3: Comparison of average number of transitions and power dissipation in SD and HSD adders. The estimates are obtained by extrapolating the values in Table 2.

It is seen that the power dissipated by the HSD-1 adder is about 12% less that the SD counterpart. The critical path delay for SD adder is equivalent to about 5 gates (two input NAND/NOR gates) whereas the delay of the HSD–1 adder is about 6 gates [9], irrespective of the word length.

Note that HSD-1 is just one of the several feasible formats within the HSD representation. Distance between signed digits can be increased to obtain further power reduction at the expense of speed. This is illustrated by the plots in Figure 2, for a word length of 24. The number 24 was chosen merely for the sake of illustration because it is divisible by several integers, yielding lot of possible HSD formats with uniform distance between the signed digits and also happens to be the number of significand bits in the IEEE standard for single precision floating–point numbers. In these figures, the measures of interest (transitions, power dissipation, delay, power $\times$ delay) are plotted as a function of d, the distance between adjacent signed digits. The point $d = 0$ corresponds to the SD representation, where every digit is signed.

In Figure 2, note that at every point between $d = 1$ and $d = 23$ (which corresponds to all digits except most significant digit unsigned), the number of signed digits is 2, since the most significant or 24th digit is signed and there is one additional signed digit in the word. As d is increased from 12 to 22, the position of the signed digit shifts from the 13th to the 23rd place. Thus the distance between signed digits is non–uniform when $12 < d \leq 22$. However, the total number of signed

digits and hence the total number of unsigned digits, remain 2 and 22, respectively, for d in this range. Hence, the power dissipation is nearly constant for all d values in the range from 13 to 22. The critical path, on the other hand, increases linearly with the longest distance between signed digits as illustrated in Figure 3–c. Also, beyond $d > 20$, the carry chain between signed digits is long enough to render the total delay (i.e., the propagation delay through the chain plus the complex gates at the ends) of the HSD adder *higher* than that of the ordinary ripple–carry adder. Finally from Figure 2-d, it is seen that the HSD–1 adder has almost the same power $\times$ delay product as the fully SD adder.

5 Conclusion

The Hybrid Signed Digit (HSD) representation is shown to lead to smaller power dissipation as compared with the fully Signed Digit representation. The power reduction is about 12% (or higher depending on the distance between the signed digits). Number of transitions, power dissipated, delay and the power $\times$ delay product were evaluated as function of the distance d between signed digits. The plots indicate that the HSD–1 adder has almost the same power $\times$ delay product than the fully SD adder.

The power (and area) reduction is achieved as a result of algorithm level change. This leaves room for additional power reduction by employing standard techniques at lower levels of the design process (such as circuit level, layout level etc). Possible future work includes investigation of optimal encodings for signed digits to reduce power dissipation.

References

[1] Avizienis, A. "Signed-digit number representations for fast parallel arithmetic". *IRE Transactions on Electronic Computers*, vol. EC-10, Sep. 1961, pp. 389–400.

[2] Kuninobu, S., Nishiyama, T., Edamatsu, H., Taniguchi, T., and Takagi, N. "Design of high speed MOS multiplier and divider using redundant binary representation". *Proc. of the 8th Symposium on Computer Arithmetic*, 1987, pp. 80–86.

[3] Makino, H., Nakase, Y., and Shinohara, H. "A 8.8–ns 54 $\times$ 54–bit Multiplier Using New Redundant Binary Architecture". In *Proceedings of the International Conference Computer Deign (ICCD), Cambridge, Massachusetts*, Oct. 1993, pp. 202–205.

[4] Miyake, J., et. al. "A Highly Integrated 40-MIPS (Peak) 64-b RISC Microprocessor". *IEEE Journal of Solid State Circuits*, vol. 25, no. 5, Oct. 1990, pp. 1199–1206.

[5] Nagendra, C., Owens, R. M., and Irwin, M. J. "Power Delay Characteristics of CMOS Adders". *IEEE Transactions on VLSI*, Sept. 1994, pp. 377–381.

[6] Nagendra, C., Owens, R. M., and Irwin, M. J. "Unifying Carry–Sum and Signed–Digit Number Representations for Low Power". In *Proceedings of International Symposium on Low Power Design, Dana Point, California*, Apr. 1995.

[7] Najm, F. "Transition Density, A Stochastic Measure of Activity in Digital Circuits". In *Proceedings of the 28th ACM/IEEE Design Automation Conference (DAC)*, 1991, pp. 644–649.

[8] Parhami, B. "Generalized signed-digit number systems: a unifying framework for redundant number representations". *IEEE Transactions on Computers*, vol. C-39, Jan. 1990, pp. 89–98.

[9] Phatak, D. S., and Koren, I. "Hybrid Signed–Digit Number Systems: A Unified Framework for Redundant Number Representations with Bounded Carry Propagation Chains". *IEEE Trans. on Computers, Special issue on Computer Arithmetic*, vol. TC–43, no. 8, Aug. 1994, pp. 880–891. (An unabridged version is available on the web via the URL http://www.ee.binghamton.edu/faculty/phatak).

[10] Srinivas, H. R., and Parhi, K. K. "A fast VLSI adder architecture". *IEEE Journal of Solid-State Circuits*, vol. SC-27, May 1992, pp. 761–767.

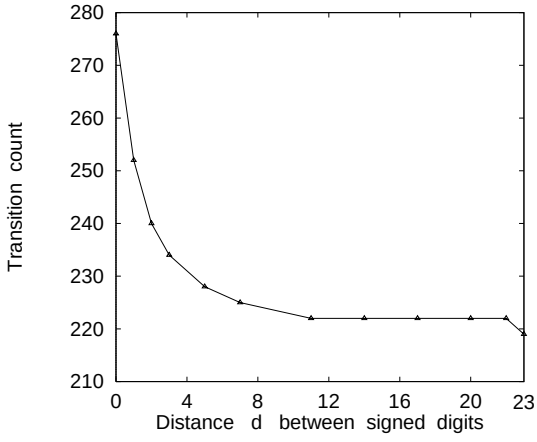


Figure 2–a : Number of transitions vs. the maximum distance d between two consecutive signed digits (wherever possible, the distance between signed digits is *uniform*). Word length is 24. $d = 0$ corresponds to the fully SD adder using the SD cell in [2]; and $d = 23$ corresponds to a ripple–carry adder. $d = 1$ corresponds to HSD–1 representation using the cells presented in [9]. Similarly, $d = 2$ corresponds to HSD–2, . . .

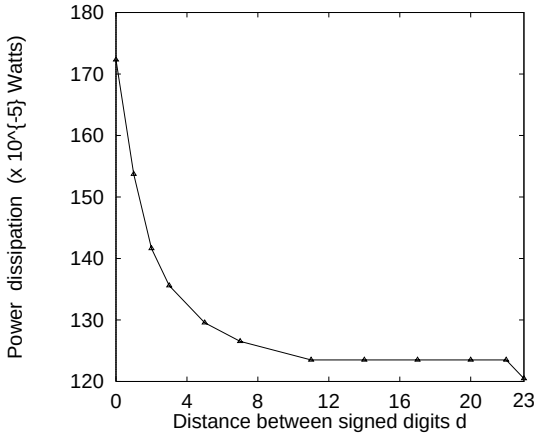
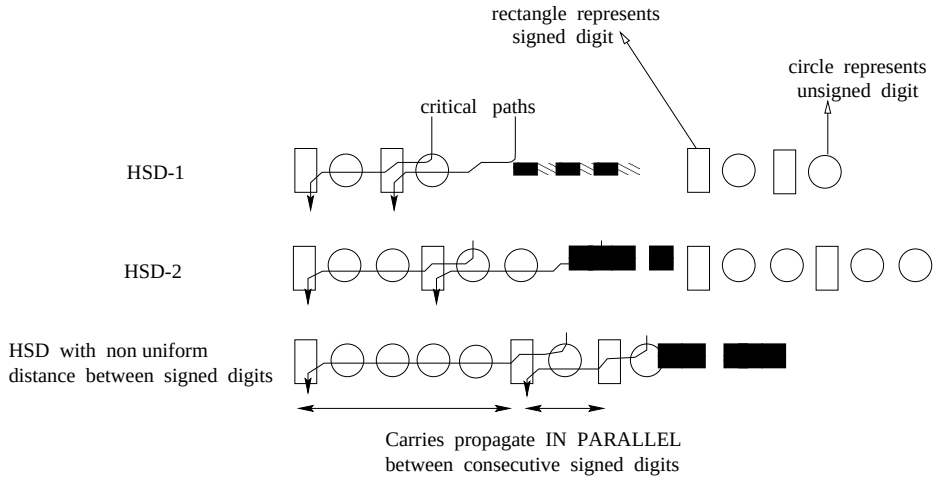
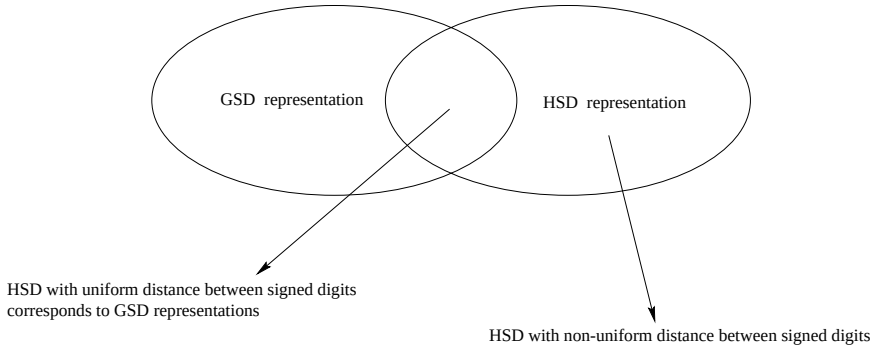


Figure 2–b : Power dissipation as a function of the distance d between two consecutive signed digits.



(a) Illustration of HSD formats



(b) Relationship between HSD and GSD number representations

Figure 1 : HSD formats and relationship between HSD and GSD number representations.
(a) Different HSD formats: HSD-k denotes a representation with k unsigned digits between every pair of consecutive signed digits. Last drawing illustrates a format with non uniform distance between signed digits.
(b) Relationship between HSD and GSD number representations.

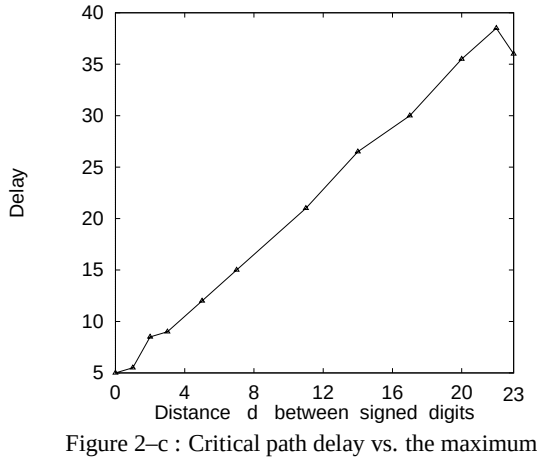


Figure 2–c : Critical path delay vs. the maximum distance d between two consecutive signed digits.

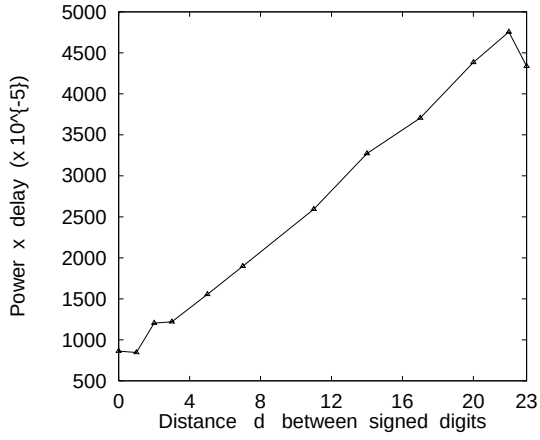


Figure 2–d : Power $\times$ delay as a function of the distance d between two consecutive signed digits.

Alternative Architectures

Power-Saving Features in AMULET2e

S.B. Furber, J.D. Garside and S. Temple.
Department of Computer Science, The University of Manchester,
Oxford Road, Manchester M13 9PL, UK.

Abstract

AMULET2e is a self-timed embedded system controller which is software compatible with the ARM6 microprocessor. Its design incorporates a number of power-saving features which can be disabled under software control, thereby allowing direct measurement of their efficacy. In this paper we review three of these features: two exploit the high frequency of sequential memory accesses to bypass unnecessary circuit functions in the main cache and the branch target cache; the third exploits the ability of clockless circuits to become completely inert when idle. These features are described and measurements on their effectiveness are presented.

1 Introduction

Designing for power-efficiency at the architectural level is a matter of minimising unnecessary circuit activity. Self-timed design, with its inherent data-driven characteristic, would appear to have an intrinsic advantage in this respect. However, when a processor is running under maximum load there is little redundant activity which can be avoided in order to reduce dissipation, whether the control is clocked or self-timed; but when the load is reduced or varies a self-timed design can adapt automatically, whereas a clocked design must employ explicit power management techniques which themselves cost power.

The AMULET microprocessors, developed at the University of Manchester, are self-timed implementations of the ARM architecture [1]. They were designed to demonstrate the feasibility and desirability of self-timed design. The advantages of self-timed design are not restricted to power-efficiency - they extend to electromagnetic compatibility, modularity, and an argument can be made on performance grounds - but in this paper we focus on the power-efficiency benefits as manifested in AMULET2e.

2 AMULET2e overview

AMULET2e is a self-timed controller designed for power-sensitive embedded applications. Throughout the design reasonable care was taken to minimise power wastage, and many techniques which are used on clocked ARM designs were re-employed in the self-timed context.

The key features of AMULET2e are a self-timed ARM-compatible microprocessor core (AMULET2), a 4 Kbyte on-chip memory, a flexible memory interface and a number of control functions. Its organisation is illustrated in figure 1. Off-chip accesses use a reference delay to ensure that timing requirements are met, with different memory regions being configurable to different bus widths and access timings (all expressed in terms of the reference delay).

3 Memory subsystem

The memory system comprises 4 Kbytes of RAM which can be memory mapped or used as a cache [2]. It is a composition of four identical 1Kbyte blocks, each having an associated 64-entry tag CAM. The CAM and RAM stages are pipelined so that up to two memory accesses may be in progress in the cache at any one time.

The cache is 64-way associative with four 32-bit words per line. It is write-through (for simplicity) and does not allocate on a write miss. Line fetch is performed with the addressed-word first and is non-blocking, allowing hit-under-miss, the line fetch being a separate, asynchronous task [3].

3.1 Cache power-efficiency

The organisation of the cache, based upon a highly associative cache with a segmented CAM tag store, follows the practice of many of the clocked ARM CPUs and is justified from a power-efficiency perspective as follows:

- the high associativity minimises misses, thereby minimising the high energy cost of off-chip accesses;
- the CAM tag store avoids the high power cost of the large number of active sense amplifiers in a set-associative cache;
- segmenting the CAM avoids the high energy cost of activating a single monolithic CAM;
- self-timing is used to delay activation of the data RAM sense amplifiers and to turn them off as soon as the data has been sensed.

As a final power-saving feature, the cache can be configured to cause the CAM lookup to be bypassed for sequential accesses to the same cache line. As, typically,

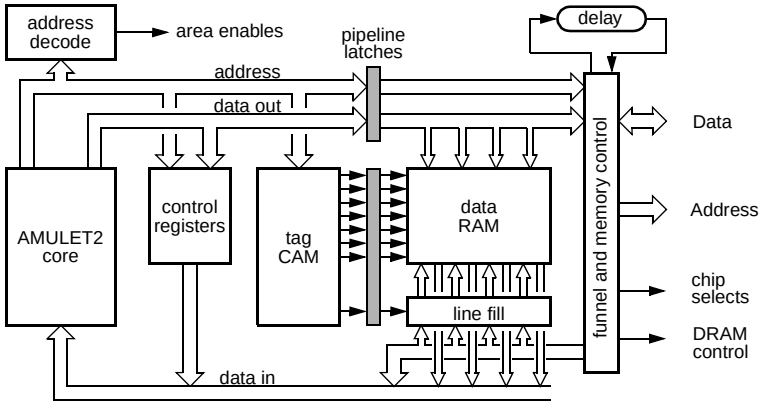


Figure 1: AMULET2e internal organization

75% of all ARM memory accesses are sequential to the preceding access, this saves a significant proportion of all CAM lookups.

Bypassing the CAM effectively removes a pipeline stage from the cache access, so it not only saves power but also results in a slightly faster cache access time and therefore increased performance. It is a simple matter to disable this optimization by presenting all addresses to the cache as if they are non-sequential, so the power and performance benefits of the CAM bypass mechanism can readily be measured.

4 Branch prediction

Branch prediction is used in the AMULET2 core to reduce the number of unwanted instructions prefetched, thereby increasing performance and reducing power consumption. The approach is based upon a branch target cache (BTC, see figure 2) which associates a particular instruction address (that of a previously discovered branch) with a target address. Subsequent references to this instruction then cause an automatic flow change.

The BTC needs no reference to the instruction itself and so is totally contained within the address interface. It acts in parallel with the PC incrementer (the normal next address 'predictor'), subverting the flow when it recognises an address. For simplicity, in AMULET2 the BTC always predicts a known branch as taken; despite this it more than halves the number of erroneously prefetched instructions in typical code

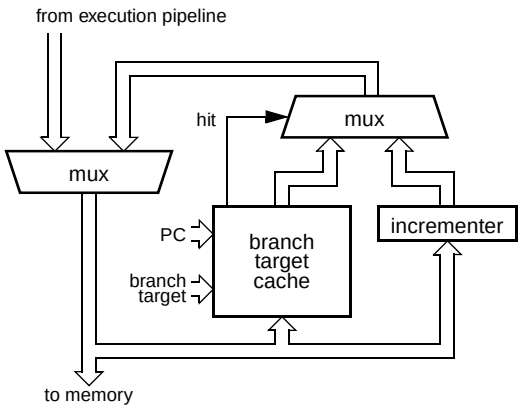


Figure 2: Branch target cache organisation

Interestingly, this unit is similar to the 'jump trace buffer' used on the MU5 mainframe computer [4] developed at the University of Manchester between 1969 and 1974; this too was an asynchronous machine.

4.1 BTC power-efficiency

In many embedded controller applications the processing power needs only to be 'adequate', whereas the electrical power consumption must be minimal. Whilst predicting branches saves power by reducing the number of unnecessary memory cycles, the branch prediction unit itself must dissipate power. An important consideration in designing the BTC for AMULET2 was that its own consumption should be low. Its simple architecture contributes to this, but the detailed design has also exploited features of the local environment for power saving.

The BTC caches the addresses of branches. The working set of branches is relatively small and sparse, thus a small, fully associative cache is appropriate, with one tag per branch. In AMULET2 a twenty entry CAM/RAM structure is employed, with the 'RAM' (which is rarely invoked) implemented as registers for speed. Addresses circulating in the address interface are compared with the CAM values to see if a branch is to be predicted. In contrast to an instruction cache, where hits are far more common than misses, most instructions are not branches so the expected result in the BTC is a cache miss.

If an address is sequential to the previous one it is unusual for its high order bits to change. Thus if, for example, the four least significant bits are ignored, the tag comparison changes only once in every sixteen addresses. AMULET2 exploits this by splitting the CAM into two (see figure 3), so that the more significant bits are compared only when such a change occurs, and most comparisons are reduced to a few bits. The full precharge/discharge cycle is only necessary in three cases:

- when a sequential cycle affects the more significant bits;

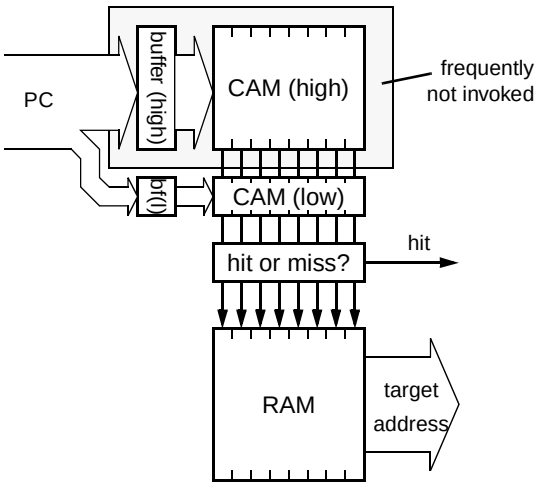


Figure 3: BTC structure

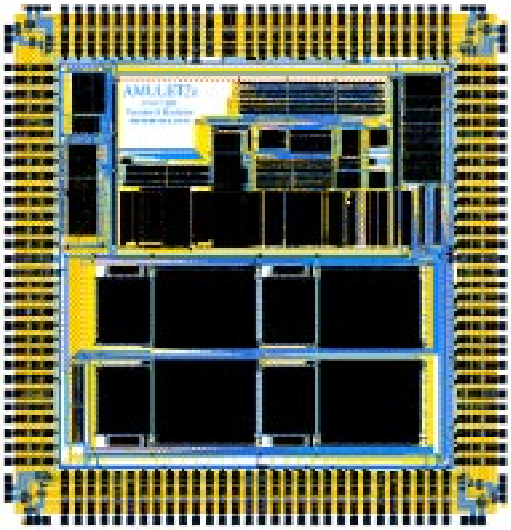


Figure 4: AMULET2e die picture

- when a branch is predicted and the next cycle is non-sequential;
- when an unpredicted branch arrives.

The first two cases are always known about in the *preceding* cycle, whilst the last causes a BTC write operation which does not require a preceding precharge, so in none of these cases is performance impacted. In fact the low-order lookup is somewhat faster than a full CAM access, so there is also a small performance benefit from this power-saving feature.

The BTC can be completely disabled and its performance and power contributions thereby measured. When the BTC is enabled the split-CAM power-saving feature can be enabled or disabled, so its contribution can also be measured.

5 'Halt'

An interesting property of a self-timed system is that it processes either at full speed or it halts, and it makes an instant transition between these two states. In its halted state the device effectively uses no power because the power consumption of digital CMOS circuits is usually negligible when they are not switching.

This has been exploited in AMULET2 by retrofitting a halt function to the existing instruction set. Most ARM programs enter an idle loop where an instruction continuously loops back to itself until an interrupt occurs. AMULET2 detects this instruction and a mechanism stalls a control loop in the execution pipeline. The stall propagates throughout the system, halting all activity. An interrupt causes immediate, full speed resumption. This feature is completely code-compatible with existing software.

The halt mechanism can be disabled and its effectiveness measured. It is different in nature to the previously mentioned optimizations as it only has an effect when the processor is idle, so it has no effect on benchmark programs that operate at peak throughput.

6 AMULET2e test results

AMULET2e has been fabricated (see figure 4) and successfully runs standard ARM code. It was fabricated on a 0.5 µm, 3-layer metal process. It uses 454,000 transistors (93,000 of which are in the processor core) on a die which is 6.4 mm square. The device in the fastest available configuration delivers 42 Dhrystone 2.1 MIPS with a power consumption of about 150 mW in the core logic (which includes the processor core and cache memory, but excludes the I/O pads) running at 3.3 V. This is faster than the older ARM710 but slower than the ARM810 which was fabricated at about the same time. It represents a power-efficiency of 280 MIPS/W, which is as good as either of the clocked CPUs.

6.1 BTC and cache power-save measurements

Two benchmark programs were used to measure the performance and power-efficiency benefits of the previously-mentioned architectural features:

- Dhrystone 2.1 is widely used to evaluate embedded cores, but we found that it has very unusual branching characteristics compared with any 'real' programs we used to during the architectural evaluation of the BTC (and we resisted the temptation to optimise the BTC for Dhrystone!);
- a set of sorting routines, combining insertion sort, shell sort and quick sort algorithms, running on a randomised data set, was used to generate alternative results. This program has very tight looping behaviour.

The results of these tests are summarised in Table 1. The base case is the processor running from cache with the BTC (and its power-saving feature) and the CAM bypass all disabled. Table 1 shows the improvement in performance (MIPS) and power-efficiency (MIPS/W) delivered by enabling each feature in turn (noting that the BTC power-save feature requires the BTC to be enabled), and their total effect when they are all enabled.

The cache CAM bypass mechanism delivers an improvement of around 5% in performance and 15% in power-efficiency on both benchmark programs. This is a

Feature	Dhrystone 2.1		Sort	
	MIPS	MIPS/W	MIPS	MIPS/W
BTC on	+5%	+0%	+22%	+15%
BTC power-save	+5%	+3%	+25%	+22%
CAM bypass	+6%	+15%	+5%	+14%
All on	+11%	+18%	+32%	+35%

Table 1: Experimental results

useful contribution from a simple circuit addition.

The BTC gives a modest 5% performance improvement on Dhrystone and a dramatic 22% on the 'Sorts' benchmark. 'Real' programs are expected to fall somewhere between these two. At the lower end it has little effect on power-efficiency (indeed, in some configurations we have observed it reduce power-efficiency). At the higher end it aids power-efficiency, but less than it enhances performance. The BTC power-save feature makes little difference to the Dhrystone performance but improves its power-efficiency by 3%. It improves the 'Sorts' performance by a further 3% and its power-efficiency by 7%. In all cases where we have observed the BTC have a negative impact on power-efficiency the BTC power-save feature has cancelled the loss and converted it into a power-efficiency gain.

6.2 'Halt' measurements

The operation of the 'Halt' feature is different in principle from the cache CAM bypass and the BTC power-save as it has no impact on power consumption when the processor is running flat-out, as it is when it is executing a benchmark program. It only comes into play when the processor is idle.

To judge its effectiveness we can observe the current consumed by the processor core logic when idling with 'Halt' disabled, which is around 50 mA if the idle loop is in the cache. With 'Halt' enabled this reduces to around 1 mA on the test card when centisecond timer interrupts are being handled, and to around 1 µA (leakage current) when interrupts are disabled. These results are summarized in Table 2.

7 Conclusions

AMULET2e is a self-timed embedded controller designed for power-sensitive applications. It incorporates a number of features which have been added to improve its power-efficiency, and which can be enabled and disabled under software control so that their contribution can be assessed.

We have described specific power-saving features in the branch prediction mechanism and cache which together reduce the core power consumption by around 20%, and also increase performance by around 10%. The branch prediction mechanism itself improves performance by between 5% and 25% and power-efficiency by between 0% and 22%, depending on the application.

We have also described the AMULET2e 'Halt' mechanism which exploits the self-timed control of the processor to reduce idle power to leakage current only, which is a factor 50,000 times lower than the idle state with 'Halt' disabled. The power consumption of the halted processor core is therefore a miserly 3 µW. As many embedded controllers spend a lot of their time halted waiting for input, interspersed with bursts of intense activity, it can be observed that an asynchronous implementation of a low power architecture can yield significantly lowered average power consumption, resulting in (for example) increased battery life in portable equipment.

Idle condition	Idd (mA)
'Halt' disabled	50
'Halt' enabled; centisecond timer interrupts	1
'Halt' enabled; interrupts disabled	0.001

Table 2: AMULET2e core idle currents

Whilst a clocked chip can make up some of this ground by using power management techniques to reduce its clock frequency, this requires software control and that software itself consumes power. On AMULET2e this minimal idle power comes easily and automatically, and can be exploited in the shortest of idle periods between bursts of activity.

8 Acknowledgements

The AMULET2e design work described in this paper was carried out as part of ESPRIT project 6909, OMI/DE-ARM (the Open Microprocessor systems Initiative - Deeply Embedded ARM Applications project). The authors are grateful for this support from the EU.

The authors are also grateful for support in various forms from Advanced RISC Machines Limited and for the fabrication service supplied by VLSI Technology, Inc. The chip was designed using EDA tools from Compass Design Automation, with accurate simulation using TimeMill from EPIC Design Technology, Inc., being vital to the final verification of the design.

Nigel Paver and Paul Day both made very substantial contributions to the design of AMULET2e and were jointly responsible for the 'Halt' mechanism described in this paper.

9 References

[1] Furber, S.B., *ARM System Architecture*, Addison Wesley Longman, 1996. ISBN 0-201-40352-8.

[2] Garside, J.D., Temple, S. and Mehra, R., "The AMULET2e Cache System", *Proc. Async'96*, Aizu-Wakamatsu, 208-217.

[3] Mehra, R. and Garside, J.D., "A Cache Line Fill Circuit for a Micropipelined Asynchronous Microprocessor", *TCCA Newsletter*, IEEE Computer Society, October 1995.

[4] Morris, D. and Ibbett, R.N., *The MU5 Computer System*, Macmillan, New York, 1979.

A Fully Reversible Asymptotically Zero Energy Microprocessor *

Carlin Vieri M. Josephine Ammer Michael Frank Norman Margolus
Tom Knight

MIT Artificial Intelligence Laboratory,
545 Technology Square, Cambridge MA 02139, USA

Abstract

The only way to compute with asymptotically zero power is reversibly. Recent implementations of reversible and adiabatic [17, 8] logic in standard CMOS silicon processes have motivated further research into reversible computing. The application of reversible computing techniques to reduce energy dissipation of current generation CMOS circuits has so far been found to be limited, but the techniques used to design reversible computers are interesting in and of themselves, and other technologies, such as Josephson Junctions and quantum computers, as well as future CMOS technologies, may require fully reversible logic. This paper discusses the design of a fully reversible microprocessor architecture.

Computing with reversible logic is the only way to avoid dissipating the energy associated with bit erasure since no bits are erased in a reversible computation. Low energy techniques such as voltage scaling lower the cost of erasing information. Techniques such as clock gating effectively reduce the number of bits erased. Reversible techniques have already been used to lower the cost of bit erasure for nodes that have a high cost of erasure, but the present work is directed at saving every bit, computing fully reversibly. The goal is to convert a conventional RISC processor to completely reversible operation. This investigation indicates where bit erasure happens in a conventional machine and the varying difficulty across datapath modules of computing without erasing bits.

The initial motivation for reversible computing research came from an investigation of fundamental limits of energy dissipation during computation [9]. The link between entropy in the information science sense and entropy in the thermodynamics sense, exhibited by Maxwell's demon [10], requires a minimum energy dissipation of $k_{\mathrm{B}}T \ln 2$, where k_{B} is Boltzmann's constant, when a bit is erased. Erasing a bit is a logically irreversible operation with a physically irreversible effect. A reversible computer avoids bit erasure.

Judicious application of reversibility in adiabatic circuits has already proven its usefulness in reducing energy dissipation [2]. This paper examines the complexity and difficulty in avoiding bit erasure entirely and discusses a set of techniques for designing reversible systems.

*This work is supported by DARPA contract DABT63-95-C-0130

1 Introduction

Power and energy dissipation in modern microprocessors is obviously a concern in a great number of applications. Riding the technology curve to deep submicron devices, multi-gigahertz operating frequencies, and low supply voltages provides high performance and some reduction in dissipation if appropriate design styles are used, but for applications with a more strict dissipation budget or technology constraints, more unusual techniques may be necessary in the future.

Adiabatic or energy recovery circuit styles have begun to show promise in this regard. Motivated by the result from thermodynamics that bit erasure is the only computing operation that necessarily requires energy dissipation, various techniques that either try to avoid bit erasure or try to bring the cost of bit erasure closer to the theoretical minimum have been implemented. To truly avoid bit erasure, and therefore perform computation that has no theoretical minimum dissipation, the computing engine must be reversible. Losses not associated with bit erasure are essentially frictional, such as the non-zero resistance of "on" transistors, and may be reduced through circuit design techniques such as optimally sized transistors, silicon processing techniques such as silicided conductors, and by moving charge through the circuit quasistatically such as constant current ramps in adiabatic circuits.

This paper discusses the engineering requirements of building a fully reversible processor. Fully reversible means that both the instruction set and the underlying circuit implementation will be reversible. Such a processor theoretically requires asymptotically zero dissipation, with dissipation per operation falling to zero as the clock period is increased to infinity. This assumes that an appropriate energy-recycling, constant-current power supply could be developed. The ISI "blip circuit" [1], stepwise capacitor charging [12], and MIT's transmission line clock drivers [3], are steps toward this end. This paper assumes that the clock drivers exist and the datapath currently being constructed uses Younis and Knight's [16] three-phase SCRL logic family in all circuits. Any complete analysis of power dissipation in an adiabatic circuit must include the dissipation in the power supply and control logic. This paper focuses on the architectural and circuit level engineering of a reversible system rather than the actual dissipation of such a system.

Following a more detailed discussion of the motivation behind this work and a few words about the terminology of the field, Section 4 covers instruction set

design, since the ISA and assembly language code must also maintain reversibility, Section 5 concerns the details of instruction fetch and decode, and Section 6 touches on some related work in the field and offers conclusions about reversible computing.

2 Why Build a Reversible Processor

A fully reversible processor must implement a reversible instruction set in a reversible circuit implementation. A reversible instruction set is one in which both the previous and next instructions are known for each instruction in a correctly written assembly language program. The dynamic instruction stream may be executed both forwards and backwards. The instruction set described here is a modification of the one designed in Vieri's master's thesis [15].

A reversible circuit implementation is one in which, in the asymptotic limit, charge flows through the circuit in a thermodynamically reversible way at all times. This is only possible if the circuit is performing a logically reversible operation in which no information is lost. A logically reversible operation is one in which values produced as output uniquely determine the inputs used to generate that output. Performing exclusively logically reversible operations is a necessary but insufficient condition for thermodynamically reversible operation. When performed using an adiabatic circuit topology, the operation is thermodynamically reversible.

Performing circuit-level operations in a thermodynamically reversible way allows the energy dissipation per operation to fall asymptotically to zero in the limit of infinitely slow operation. Conventional CMOS circuits have a minimum energy dissipation associated with each compute operation that changes the state of the output node, regardless of operation frequency. So-called "adiabatic" techniques, in which the dissipation per compute operation is proportional to the operation frequency, have shown themselves to be useful in practical applications [13, 2].

As mentioned above, adiabatic operation requires that the circuits perform logically reversible operations. If one attempts to implement a conventional instruction set in a reversible logic family, reversibility will be broken at the circuit level when the instruction set specifies an irreversible operation. This break in reversibility translates to a required energy dissipation.

3 A Few Words About Words

Much of the discussion about reversible computing is hampered by imprecision and confusion about terminology. In the context of this paper, terms will hopefully be used consistently according to the definitions in this section.

The overall field of reversible computing encompasses anything that performs some operation on some number of inputs, producing some outputs that are uniquely determined by and uniquely determine the inputs. While this could include analog computation, this paper is only concerned with digital computation. Reversible computation only implies that the process is deterministic, that the inputs determine the outputs, and that those outputs are sufficient to determine the inputs.

A particular design may be reversible at one or more levels and irreversible at other levels. For example, an instruction set may be reversible and yet be implemented in an irreversible technology. The test chip "Tick" [4] was an eight-bit implementation of an early version of the reversible Pendulum instruction set in an irreversible standard CMOS technology.

The specific implementation of a reversible computing system may vary greatly. Numerous mechanical structures have been proposed, including Drexler's "Rod Logic" and Merkle's clockwork schemes [11]. Younis and Knight's SCRL is a reversible electronic circuit style suitable for use in the implementation of a reversible computer.

Many papers refer to "dissipationless" computing. This is a misnomer and generally may be interpreted as shorthand for "*asymptotically* dissipationless" computing. This means that if the system could be run arbitrarily slowly, and if some unaccounted-for N^{th} order effect of the particular implementation does not interfere, in the limit of infinite time the dissipation per computation asymptotically approaches zero. Only reversible computations may be performed asymptotically dissipationlessly because bit erasure requires a dissipation of $k_{\mathrm{B}}T\ln 2$ regardless of operation speed. Any N^{th} order effects that prevent a reversible system from dissipating zero energy are essentially frictional in nature, rather than fixed, fundamental barriers.

4 The Pendulum Instruction Set

The particular implementation discussed here is known as the Pendulum processor. The Pendulum processor was originally based on the elegantly simple MIPS RISC architecture [7]. The register-to-register operations, fixed instruction length, and simple memory access instructions make it a good starting point for a radically different approach to instruction set design. For ease of implementation, and of course to maintain reversibility, the instruction set has been substantially modified. It retains the general purpose register structure and fixed length instructions, however.

The Pendulum processor supports a number of traditional instructions with additional restrictions to ensure reversibility. The instruction set includes conventional register to register operations such as add and logical AND, shift and rotate operations, operations on immediate values such as add immediate and OR immediate, conditional branches such as branch on equal to zero and branch on less than zero, and a single memory access operation, exchange. The direction of the processor is changed using conditional branch-and-change-direction instructions.

4.1 Register to Register Operations

Conventional general purpose register processors read two operands, stored in two possibly different registers, and perform some operation to produce a result. The result may be stored either in the location of one of the operands, overwriting that operand, or some other location, overwriting whatever value was previously stored there. This produces two difficulties for a reversible processor. First, writing a result over a previously stored value is irreversible since the information stored there is lost. Second, mapping from the information space of two operands to the space of two operands and a result will quickly fill the available memory. However, if the

result is stored in the location originally used for one of the operands, the computation takes two operands as input and outputs one operand and a result. This space optimization is not required for reversibility if the processor can always store the result without overwriting some other value, but it is a useful convention for managing resources.

An astutely designed instruction set will inherently avoid producing garbage information while retaining flexibility and power for the programmer. This leads to the distinction between expanding and non-expanding operations. Both types of instruction are reversible; the distinction is made only in how much memory these instructions consume when executed.

All non-expanding two operand instructions in the Pendulum instruction set take the form:

$$R_{sd} \leftarrow \mathcal{F}(R_{sd}, R_s) \tag{1}$$

where R_{sd} is the source of one operand and the destination for the result, and R_s is the source of the second operand.

By contrast, logical AND is not reversible if only the result and one operand are retained. Except for the special case of the saved operand having every bit position set to one, the second operand can not be recovered accurately and must be stored separately. Since operations like AND, including logical OR and shift operations, require additional memory space after execution, they are termed expanding operations. The problem then arises of how store that extra information. If the two operands continue to be stored in their original location, the result must be stored in a new location. It is still not permissible for the result to overwrite a previously stored value, so the result may either be stored in a location that is known to be clear or combined with the previously stored value in a reversible way. The logical XOR operation is reversible, so the Pendulum processor stores the result by combining it in a logical XOR with the value stored in the destination register, forming ANDX, ORX and so on. Logical ANDX and all other expanding two operand instructions take the form:

$$R_d \leftarrow \mathcal{F}(R_s, R_t) \oplus P \tag{2}$$

where R_s and R_t are the two operand source registers, R_d is the destination register, and P is the value originally stored in R_d.

Constructing a datapath capable of executing an instruction stream forwards and backwards is simplified if instructions perform the same operation in both directions. Addition, which appears simple, is complicated by the fact that the ALU performs addition when executing in one direction and subtraction when reversing. The expanding operations are their own inverses, since

$$P = \mathcal{F}(R_s, R_t) \oplus R_d \tag{3}$$

Non-expanding operations could take the same form as the expanding operations, performing `addx` and so on, simplifying the ALU, but programming then becomes fairly difficult. Only expanding operations, which require additional storage space, are implemented to consume that space.

Converting a conventional register to register instruction execution scheme to reversible operation is relatively simple. The restriction on which registers can be operands is minor, and implementation of an SCRL ALU is not particularly difficult. While a conventional processor erases a significant number of bits in these operations, preserving them is not difficult.

4.2 Memory Access

A reversible memory system, named XRAM, has been fabricated in a 0.5 μm CMOS silicon process. The system was intended to be a prototype of the Pendulum register file. Reversible memory system design is discussed in more depth elsewhere [14], and this section draws heavily on previous work by the authors.

From a system point of view, the only additional requirement of a reversible memory, beyond a traditional memory system's function, is that it not erase bits when it is read from and written to. The memory must of course perform as a random access memory, allowing bits to be stored and retrieved. Bit erasure can happen as a fundamental side effect of the operation of the memory or as a function of the particular implementation. For example, one can imagine a memory in which the externally visible values being stored and retrieved are never lost but the implementation of the memory is such that intermediate bits are erased internally.

A traditional SRAM architecture is based on read/write operations. An address is presented to the memory and, based on a read/write and possibly an enable signal, a word of data is read from or written to the memory array. Data may be read from any location an arbitrary number of times, and data written to a location overwrites that location's previously stored data.

Reading a value does not at first seem to be an irreversible operation. Reading from a standard memory creates a copy of the stored value and sends it to another part of the computing system. An arbitrary number of copies may be created this way. If, in a reversible system, the overall system can properly manage these copies, the memory need not be concerned with them. The larger system will, however, probably exhaust its ability to store or recover the bits generated by the production of an arbitrary number of copies. So it is a desirable feature of a reversible memory not to be a limitless source of bits when used in a larger system. It must be emphasized, however, that copy itself is *not* an irreversible operation.

A conventional memory performs explicit bit erasure during writes because the previously stored value is overwritten and lost. A reversible memory must save those bits somehow. The specific mechanism for this may vary. For example, reads may be performed destructively, as in a DRAM, to avoid producing copies of the data. The information is moved out of the memory rather than being copied from it.

During writes, the value which would be overwritten could be pushed off to a separate location, either automatically or explicitly under programmer control. This only postpones the problem until later since any finite capacity storage will be filled eventually.

If the separate location is accessible to the programmer, however, that data may either be useful or it may be possible to recover the space by undoing earlier operations. So if a write is preceded by a destructive read, the old information is moved out of the memory and into the rest of the system, and the new information replaces it in the memory. The old value has been *exchanged* for the new value. This type of eXchange memory architecture, or XRAM, is the memory access technique

used in the Pendulum register file and for data and instruction memory access. The instruction set supports a single exchange instruction which specifies a register containing the memory address to be exchanged and a register containing the value to be stored to memory and in which the memory value will be placed.

The essential insight of the XRAM is that performing a read and then a write to the same memory location does not lose any information. One data word is moved out of the memory, leaving an empty slot for a new value to be moved in. In general, *moving* data rather than *copying* is a valid technique in reversible computing for avoiding bit erasure on the one hand and avoiding producing large amounts of garbage information on the other.

4.3 Control Flow Operations

If programs consisted solely of register to register and memory access operations, programming and implementation would be relatively simple. Unfortunately, conditional branches are crucial to creating useful programs. The processor must be able to follow arbitrary loops, subroutine calls, and recursion during forward and reverse operation. A great deal of information is lost in conventional processors during branches, and adding structures to retain this information is very difficult.

Any instruction in a conventional machine implicitly or explicitly designates the next instruction in the program. Branch instructions specify if a branch is to be taken, and if so, what the target is. Non-branch instructions implicitly specify the instruction at the next instruction memory address location. To follow a series of sequential instructions backwards is trivial; merely decrement the program counter rather than incrementing it. Following a series of arbitrary jumps and branches backwards in a traditional processor is impossible: the information necessary to follow a jump or branch backwards is lost when the branch is taken. A reversible computer must store enough information, either explicity in the instruction stream or elsewhere, to retrace program execution backwards.

To reverse an instruction stream, some information must be available at the target location of a branch specifying how to undo the branch. During reverse operation, a target instruction must somehow be identifiable and the processor must be able to determine the "return address" of the branch. A challenge exists as to how to ensure reversibility even if the code is written incorrectly. A simple "illegal instruction" check to determine if the targeted instruction is not a proper target type is a legitimate technique, but data-dependent checks are generally not possible.

Requiring branches to target particular types of instructions to ensure branch reversibility, however, is a convenient technique for reversible processor design. Early designs proposed explicit "come-from" instructions [15] or a branch register and branch bit [5]. Space does not permit a complete discussion of possible techniques for performing jumps and branches reversibly, but the literature contains a number of examples that differ from the scheme presented here [6]. The discussion below refers only to the particular scheme used in the current version of the Pendulum processor.

Pendulum branch instructions specify the condition to be evaluated, either equal to zero or less than zero, the register containing the value to be evaluated, and a register containing the target address. The instruction at the target address must be able to point back to the branch address and know if the branch was taken or if the target location was reached through sequential operation. For proper operation, each branch instruction must target an identical copy of itself.

When a branch condition is true, an internal branch bit is toggled. If the branch bit is false and the branch condition is true, the program counter update (PCU) unit exchanges the value of the program counter and the target address. The target address must hold an identical branch instruction which toggles the branch bit and sequential operation resumes. The address of the first branch instruction is stored in the register file so that during reverse operation the branch can be executed properly.

5 Instruction Fetch and Decode

Reading from the instruction memory suffers from the same difficulty as reading from the data memory. Each copy created when an instruction is read must be "uncopied." If instruction fetch operations are performed by moving instructions rather than copying them, the instructions must be returned to memory when the instruction has finished executing.

After the instruction is read (or moved) from the instruction memory, the opcode is decoded, and a number of datapath control signals are generated. Just before the instruction is moved back to the instruction memory, these control signals must be "ungenerated" by encoding the instruction.

A certain symmetry is therefore enforced with respect to instruction fetch and decode. An instruction is moved from the instruction memory to the instruction decode unit. The resulting datapath control signals direct operation of the execution and memory access units. After execution, the control signals are used to restore the original instruction encoding, and the instruction may then be returned to the instruction memory.

The processor must be able to return the instruction to its original location, so its address must be passed through the datapath and made available at the end of instruction execution. Since the address of the next instruction must also be available at the end of instruction execution, the Pendulum processor has two instruction address paths. One path contains the address of the instruction being executed, and the program counter update unit uses it to compute the address of the next instruction. The second path contains the address of the previous instruction and the PCU uses it to compute the address of the current instruction. The output of the PCU is then the next instruction address and the current instruction address, which are the values required to return the current instruction to the instruction memory and read out the next instruction.

Performing these computations during branching instructions is very complex, especially when the processor is changing direction. Traditional processors throw away every instruction executed, and ensuring that the instructions are returned to the instruction memory is challenging.

6 Conclusions

This extreme approach to low power computing is clearly impractical in the near-term. All the primary

blocks of a traditional RISC processor erase bits, and retaining those bits presents varying levels of difficulty to the designer. These challenges present the opportunity to reexamine conventional RISC architecture in terms of bit erasure during operation. Register to register operations and memory access are relatively easy to convert to reversibility, but control flow and, surprisingly, instruction fetch and decode, are decidedly non-trivial. This knowledge may be used in traditional processor design to target datapath blocks for energy dissipation reduction.

References

[1] W. Athas, L. Svensson, and N. Tzartzanis. A resonant signal driver for two-phase, almost-non-overlapping clocks. In *International Symposium on Circuits and Systems*, 1996.

[2] W. Athas, N. Tzartzanis, L. Svensson, L. Peterson, H. Li, X. Jiang, P. Wang, and W-C. Liu. AC-1: A clock-powered microprocessor. In *International Symposium on Low Power Electronics and Design*, pages 18–20, 1997.

[3] Matthew E. Becker and Thomas F. Knight, Jr. Transmission line clock driver. In *Proceedings of the 1998 ISCA Power Driven Microarchitecture Workshop*, 1998.

[4] Michael Frank and Scott Rixner. Tick: A simple reversible processor (6.371 project report). Online term paper, may 1996. `http://www.ai.mit.edu/~mpf/rc/tick-report.ps`.

[5] Michael P. Frank. Modifications to PISA architecture to support guaranteed reversibility and other fetures. Online draft memo, July 1997. `http://www.ai.mit.edu/~mpf/rc/memos/M07/M07_revarch.html`.

[6] J. Storrs Hall. A reversible instruction set architecture and algorithms. In *Physics and Computation*, pages 128–134, November 1994.

[7] Gerry Kane and Joe Heinrich. *MIPS RISC Architecture*. Prentice Hall, 1992.

[8] J. G. Koller and W. C. Athas. Adiabatic switching, low energy computing, and the physics of storing and erasing information. In *Physics of Computation Workshop*, 1992.

[9] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Research and Development*, 5:183–191, 1961.

[10] J. C. Maxwell. *Theory of Heat*. Longmans, Green & Co., London, 4th edition, 1875.

[11] Ralph C. Merkle. Towards practical reversible logic. In *Physics and Computation*, pages 227–228, October 1992.

[12] L.“J.” Svensson and J.G. Koller. Adiabatic charging without inductors. Technical Report ACMOS-TR-3a, USC Information Sciences Institute, February 1994.

[13] Nestoras Tzartzanis and William C. Athas. Energy recovery for the design of high-speed, low power static RAMs. In *International Symposium on Low Power Electronics and Design*, pages 55–60, 1996.

[14] Carlin Vieri, M. Josephine Ammer, Amory Wakefield, Lars “Johnny” Svensson, William Athas, and Thomas F. Knight, Jr. Designing reversible memory. In C. S. Calude, J. Casti, and M. J. Dinneen, editors, *Unconventional Models of Computation*, pages 386–405. Springer-Verlag, 1998.

[15] Carlin J. Vieri. Pendulum: A reversible computer architecture. Master’s thesis, MIT Artificial Intelligence Laboratory, 1995.

[16] Saed G. Younis and Thomas F.. Knight, Jr. Practical implementation of charge recovering asymptotically zero power CMOS. In *Proceedings of the 1993 Symposium in Integrated Systems*, pages 234–250. MIT Press, 1993.

[17] Saed G. Younis and Thomas F. Knight, Jr. Asymptotically zero energy split-level charge recovery logic. In *International Workshop on Low Power Design*, pages 177–182, 1994.

Low-Power VLIW Processors: A High-Level Evaluation

Jean-Michel Puiatti, Christian Piguet *, Eduardo Sanchez, Josep Llosa †

Logic Systems Laboratory
Swiss Federal Institute of Technology
CH–1015 Lausanne, Switzerland
{Jean-Michel.Puiatti, Eduardo.Sanchez}@epfl.ch

Abstract

Processors having both low-power consumption and high-performance are more and more required in the portable systems market. Although it is easy to find processors with one of these characteristics, it is harder to find a processor having both of them at the same time. In this paper, we evaluate the possibility of designing a high-performance, low-consumption processor and investigate whether instruction-level parallelism architectures can be adapted to low-power processors. We find that an adaptation of high-performance architecture, such as the VLIW architecture, to low-power 8b or 16b microprocessors yields a significant improvement in the processor's performance while keeping the same energy consumption.

1 Introduction

In recent years, the need for ultra low-power embedded microcontrollers has been steadily increasing. This can be explained by the high demand for portable applications. Currently, a wide range of products, such as embedded microprocessors from 8b to 32b and DSP, can be found on the market and are well adapted to a wide range of applications. Eight-bit embedded microcontrollers can be found in the form of low-complexity circuits that have generally no pipeline, no cache memories, and a reduced level of performance. The Motorola 68HC11, Intel 80C51, and PIC 16Cxx are examples of such products. They consume between 5 and 50 miliwatts at 3 volts, have slow clock frequency (usually no more than 20 MHz), and take several clock cycles to execute an instruction [12]. On the other hand, high-end, low-power processors can be found in the form of pipelined 32b processors with small cache memories and a high level of performance. The DEC StrongArm 110, Motorola MPC821, and IBM PowerPC 401 are examples of these products. They consume between 100 and 1000 milliwatts at 3 volts and have a high clock frequency (between 25 and 200 MHz) [12][14]. Between these two categories there is a gap. Indeed, in spite of the need for low-power processors adapted to 8b and 16b applications, there are no 8b or 16b processors that reach a level of performance comparable to the 32b low-power processors.

In this paper, we evaluate the possibility of designing a high-performance, low-consumption 8b or 16b microcontroller. More precisely, we investigate whether the instruction-level parallelism (ILP) architectures (e.g., superscalar or VLIW) used in high-performance, high-consumption processors can be adapted to low-power, high-performance 8b or 16b microcontrollers. In order to quantify the potential improvements that can be obtained by these kinds of parallel architectures, we make a comparison between low-power scalar processors and low-power ILP processors in terms of both performance and energy efficiency. The metric used in our comparison is the Energy Delay Product [5], EDP, defined as the product between the total energy, E_T, needed to execute a task and the time, T_{exec}, needed to execute this task: $EDP = E_T \cdot T_{exec}$.

Ricardo Gonzalez and al. [5] showed that superscalar architectures with a compile-time scheduler do not improve the energy-efficiency level. Thomas D. Burd [3] argued that instruction-level parallelism (ILP) architectures do not improve the energy efficiency due to the control overhead and the unissued instructions (speculation). However, we believe that VLIW architectures adapted for low-power can help us obtain a better consumption-performance trade-off. These beliefs are motivated by new developments in the VLIW field, such as the new architectural solution HP/Intel IA-64 [6] and TI 'C6201 processors [15].

The remainder of this paper is organized as follows. Section 2 describes the characteristics of the CoolRISC 816, which is the processor of reference of our experiment. Section 3 presents the ILP architecture used in our experiment to increase the performance of the low-power processor of reference. Section 4 describes our experiment and shows the results. Finally, Section 5 concludes.

2 CoolRISC 816: A low-power 8b processor

The CoolRISC 816 [13] is an ultra low-power embedded 8b microcontroller developed by the Centre Suisse d'Electronique et de Microtechnique (CSEM). It has the following characteristics (core only): a harvard architecture (separate code and data memory), sixteen 8-bit registers, 22-b wide instructions, a clock frequency of up to 18 MHz, a typical consumption of 105 μW/MHz at 3

* Ultra Low Power Section, Centre Suisse d'Electronique et de Microtechnique, CH–2000 Neuchatel, Switzerland, piguet@csemne.ch

† Departament d'Arquitectures de Computadors, Universitat Politecnica de Catalunya, Barcelona, Spain, josepll@ac.upc.es

volts with a 0.5 μm three metal layers CMOS technology.

The CoolRISC 816 has a non blocking pipeline which allows it to execute an instruction every cycle without adding extra delay due to pipeline stalls. The main limitation from a performance point of view is the working frequency: the maximum clock frequency of the CoolRISC 816 core is 18 MHz, but generally the maximum working frequency is limited by the access time of the code memory. The code memory used in CoolRISC is ultra-low power at the cost of a slow access time.

The energy consumption of the CoolRISC 816 is distributed in three different parts: the core, the data memory and the code memory. Figure 1 shows the typical distribution of the energy consumption when CoolRISC is executing a program. This data was obtained by executing a set of programs and extracting the relative utilization of the core, data memory, and code memory. The set of programs used consisted of: a quicksort, a stringsort, a FFT, and a sine/cosine computation. The extracted average resource utilization is: 100% for the processor core, 100% for the code memory, and 40% for the data memory.

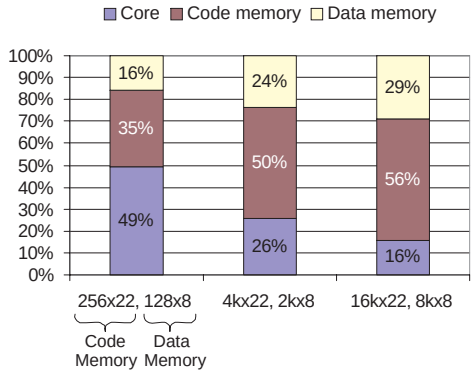


Figure 1: Energy consumption distribution in the CoolRISC 816.

Figure 1 shows that the energy consumed by the processor core corresponds to less than 50% of the total energy consumption and that the major sources of energy consumption are the memories.

3 Increasing Performance

Currently, high-performance processors use instruction-level parallelism (ILP) to increase their performance. Indeed, a scalar processor such as CoolRISC 816 executes sequentially a single flow of instructions. However, not all the executed instructions are interdependent and therefore several of these instructions can be executed in parallel.

Superscalar and VLIW [7] architectures are the two main types of architectures that exploit instruction-level parallelism to achieve a higher level of performance. Superscalar processors fetch several instructions in the code memory, analyze the dependencies between all of them, and, according to the resource availability and the instruction dependencies, schedule the instructions in the various units. Therefore, there is a considerable increase in the circuit complexity due to the instruction dispatch unit. VLIW architectures eliminate this increase in hardware complexity using a compile-time instruction scheduler. This means that the control of the instruction dependencies is made by the compiler. The VLIW compiler analyzes the code and, according to the processor's resources and the dependencies between instructions, generates very large instructions that contain several independent *meta-instructions* which will be executed in parallel. The processor has only to fetch and execute these very large instructions without checking the *meta-instruction* dependencies.

VLIW architectures have a major drawback, the code density of a VLIW processor depends on the available instruction parallelism. If there is no sufficient instruction parallelism to generate a VLIW instruction that uses all units, the non-used units will execute a NOP *meta-instruction*, which results in a considerable increase in code size, and therefore, in energy consumption. To solve the problem of the NOP insertion, the new generation of VLIW processors, such as the HP/Intel Merced [6], contains a special encoding technique which eliminates the extra-NOP insertion. Figure 2 illustrates this kind of technique. Each VLIW instruction contains several *meta-instructions* (four in our example) which could be dependent or independent instructions. An additional field is added to specify the group of *meta-instructions* that will be executed in parallel. The **unit number** field specifies which unit must execute the *meta-instruction*, and the **separator** bit between two *meta-instructions* within a *meta-instructions* is set to '0' if the two can be executed in parallel, to '1' if they must be executed sequentially. The hardware costs of the NOP elimination are the extra bits added to the code memory (3 bits per *meta-instruction* in our example) and the crossbar needed to send the *meta-instruction* to their corresponding unit. However, this technique prevents the increase in code size (and therefore of consumption) due to the extra NOP insertion.

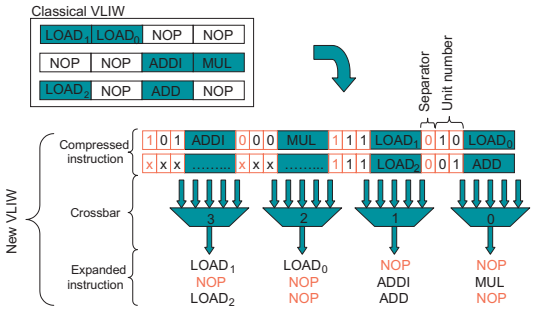


Figure 2: VLIW architecture: NOP elimination.

VLIW architectures can be divided in two main groups: (1) The heterogeneous VLIW architectures which are the most common among existing VLIW architectures. The term heterogeneous indicates that the units are different, which in turn means that a *meta-instruction* must be dispatched to a unit capable of executing it; (2) The homogeneous VLIW architectures which are VLIW architectures composed of several units which are able to execute any kind of *meta-instruction*. As the units are homogeneous there is no need for a crossbar to dispatch the *meta-instructions* to their corresponding units, and therefore there is no need for a **unit number** field.

4 Experimental part

This experiment aims at comparing 8-bit or 16-bit processors, such as scalar and VLIW architectures, and quantifying their differences in terms of both performance and energy consumption. Performance and energy consumption are computed for the execution of a benchmark suite composed of inner loops of several programs. Most of the execution time of a program is spent in inner loops: for example, on a HP-PA 7100 processor, 78% [8] of the execution time of the Perfect Club Benchmark Suite [2] is spent in inner loops. Therefore, the performance achieved and the energy consumed in inner loops are representative of the execution of the entire program.

The next subsections address the following issues: the compilation techniques used to obtain the executable code of inner loops; the benchmark used for our evaluation; the architectures compared in our experiment; the consumption model used to estimate the energy consumption; and finally the results of our experiment.

4.1 Compilation techniques

The executable code for the different architectures is obtained as follows: first, the dependencies graph of the instructions of the loop is generated; then, the instructions are scheduled depending on the constraints imposed by the data dependencies and by the architecture of the processor.

To generate the graphs of dependencies, we used principally the ICTINEO tool [1], which extracts the inner loops of a FORTRAN program and provides an optimized graph of dependencies for each inner loop.

Software pipelining has been used to schedule operations because it is the most effective compiling technique for loop parallelization. The software pipeline technique used is Swing Modulo Scheduling (SMS) [9]. SMS tries to produce the maximum performance and, in addition, includes heuristics to decrease the high register requirements of software pipelined loops [10]. When the number of registers required is higher than the available number, spill code [4] (i.e, instructions which temporarily save the contents of some registers into the data memory) has to be introduced, increasing energy consumption. When required, spill code was added in software pipelined loops using the heuristics described in [11].

Figure 3 shows the principle of software pipelining. In the sequential execution of a loop each iteration and each instruction are executed sequentially. Software pipelining rearranges the instructions, according to the dependencies and architectural constraints, in order to obtain a loop divided in SC stages (three in our example, from $S0$ to $S2$) which can be executed in parallel. Every stage is divided in II (Initiation Interval) cycles in which it is possible to execute one or more instructions.

4.2 Benchmark

As a benchmark, we used a set of 25 integer loops. These loops are divided into three groups. The first includes five integer loops which operate on 8-bit data: FIR filter, vector-matrix multiplication, vector-vector multiplication (dot), vector-vector addition, and function integration. The second consists of the same five integer loops operating on 16-bit data. Finally, the third group is composed of 15 16-bit integer loops of the Perfect Club Benchmark Suite [2].

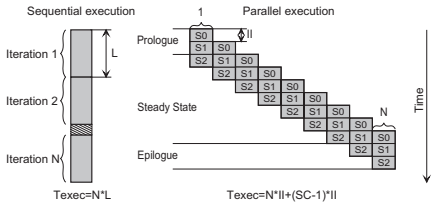


Figure 3: Parallel execution of a loop using software pipelining.

4.3 Compared architectures

The evaluated architectures are based on the CoolRISC architecture and use the same low-power memories that have been used with CoolRISC 816. These memories have the property of having no sense amplifiers, which eliminates static energy consumption. Therefore, there is no additional penalty due to the width of the instruction words.

- **C8** CoolRISC 816, processor of reference
- **C16** 16-b version of the **C8**
- **V8E1** VLIW heterogeneous, 8-b, 1 Load/Store, 2 ALUs, 1 Branch Unit
- **V8E2** VLIW heterogeneous, 8-b, 2 Load/Store, 2 ALUs, 1 Branch Unit
- **V8H1** VLIW homogeneous, 4 units, 1 acces to the data memory at the same time
- **V8H2** VLIW homogeneous, 4 units, 2 acces to the data memory at the same time
- **V16E1** 16-b version of the **V8E1**
- **V16E2** 16-b version of the **V8E2**
- **V16H1** 16-b version of the **V8H1**
- **V16H2** 16-b version of the **V8H2**

4.4 Consumption model

The consumption model is based on the utilization of resources. The energy needed to execute a task is computed by summing the energy consumed by the different resources. As CoolRISC 816 is our processor of reference, we base the energy consumption estimates on our in-depth knowledge of the energy consumption characteristics of the **C8** processor, which are extracted from the real implementation of the processor. Because the compared VLIW architectures use the NOP elimination technique, their instructions contain predecoded bits that indicate which units must work. Therefore, it is possible to halt the signals activity of all unused units. As a consequence, the units which do not execute a *meta-instruction* do not consume any energy. For the heterogeneous VLIW architectures the energy needed to execute a *meta-instruction* is estimated as the energy consumption of the operational part of the **C8** or **C16** processor (pipeline, decoder, register file accesses, ALU operation). For the homogeneous VLIW architectures the energy needed to execute a *meta-instruction* is estimated as the same energy needed to execute a scalar instruction in the **C8** or **C16**. This high-level modeling of the energy consumed during the execution of an instruction implies a certain loss of precision. However, one factor limits the

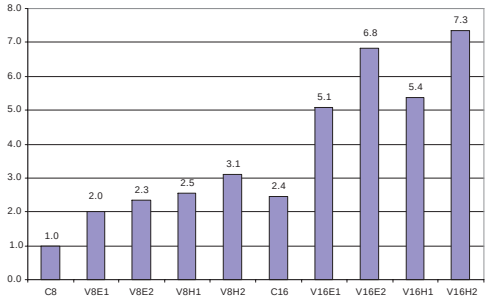


Figure 4: Speed-up comparison.

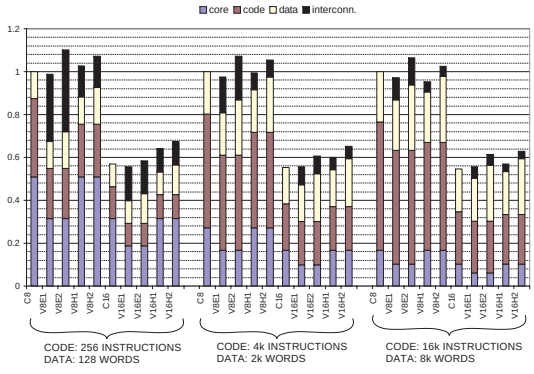


Figure 5: Energy comparison.

impact of estimate error: as described in Section 2, the energy consumed in the processor core represents only a small part (about 30%) of the total energy consumption.

The energy needed to perform a memory (code or data) access is estimated through a statistical energy consumption model of the memory architecture. This model takes into account the type of memory (RAM or ROM), its size (in words), its geometry (number of rows and columns), the width of the word, and the power supply voltage. The technological parameters are extracted from a 0.5 μ CMOS process. In our experiment we use the typical value of the energy consumption per memory access.

The extra consumption energy due to the interconnection is estimated using a statistical model of the energy consumption of the crossbar and of the circuit overhead due to the additional access ports of the register file.

4.5 Results

In this subsection we compare the performance and energy consumption of the architectures described in Subsection 4.3. All use the same power supply voltage (V_{dd}=3V) and clock frequency (imposed by the access time of the code memory). The experiment is repeated for several memory configurations.

Figure 4 compares the performance, in terms of speed-up, of the different processors with respect to the **C8** processor. Figure 5 shows the ratio between the energy consumption of the different processors and the **C8**, while processor executing our benchmark. It illustrates the energy consumption distribution. Figure 6 shows the ratio between the energy-delay product achieved by the

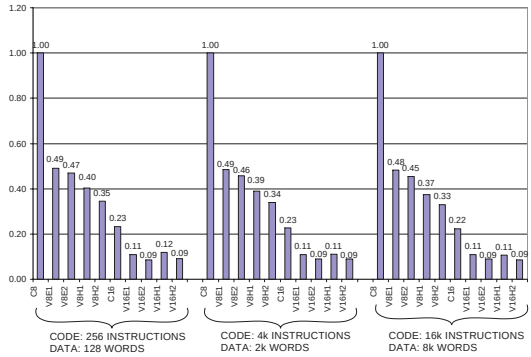


Figure 6: Energy-Delay Product comparison.

different processors and by the **C8** processor, while executing our benchmark.

From these three figures we can observe the advantage of the transition: (1) from a 8-bit to a 16-bit architecture, and (2) from a scalar to a VLIW architecture.

The transition from a 8-bit to a 16-bit architecture yields a major improvement in the energy-delay product (approximately a factor of four). This is a consequence of the smaller number of instructions required to execute the benchmark, which contains a majority of 16b data. Performance increases by a factor of about 2.4 while energy consumption decreases by a factor of about 1.7. This result shows the importance of having an architecture able to process efficiently the data of the application.

The transition from scalar to a VLIW architecture improves significantly the energy-delay product (by a factor varying between 2.0 and 2.8). Indeed, VLIW architectures achieve better performance while consuming more or less the same energy as scalar architectures. This observation is explained by a redistribution of the energy consumption: the increase in energy consumption of the VLIW core is compensated for by a decrease in the energy consumption of the code memory. The increase is due to the circuit overhead introduced by the interconnections (crossbar and register file). On the other hand, the decreased consumption of the code memory can be explained as follows: first, the employed memories do not have sense amplifiers, and therefore do not consume static energy (i.e., there is no penalty for the larger instruction words); second, a VLIW processor requires less energy to fetch a *meta-instruction* than a scalar architecture. In fact, as the energy consumed by the line decoder is independent of the width of the word, the energy needed to fetch four instructions simultaneously is less than four times the energy consumption needed to fetch one instruction.

The main difference between homogeneous and heterogeneous VLIW architectures is in terms of performance. The former reach a higher level of performance due to their higher machine parallelism; however, the downside is the higher core complexity, which entails a higher energy consumption. Therefore, if the ILP is insufficient, the homogeneous and heterogeneous VLIW architectures have a similar energy-delay product since there is no significant difference in terms of speed-up. On the other hand, if sufficient ILP can be extracted then the homogeneous architecture attains a higher level of performance (as is the case for our 8-bit processors), ul-

timately superseding the heterogeneous one with respect to the energy-delay product.

5 Conclusion

In this paper we have shown that an adaptation of high-performance architectures, such as the VLIW architecture, to low-power embedded 8b or 16b microcontrollers using low-power memories yields a significant improvement of the energy-delay product compared to a scalar processor. This improvement, by a factor varying between 2 and 3, is obtained through a redistribution of the energy consumption, which allows to obtain a higher level of performance while keeping the energy consumption at the same level. We have also shown the importance of using a processor adapted to the size of the data in order to minimize the number of instructions executed, which leads to a decrease in the energy consumption and in the time of execution.

Our results are based on loop parallelization and on a high-level energy consumption model, which allow us to validate the use of VLIW architectures for high-performance low-power processors and to identify which VLIW architecture provides the best results. The next step will be to develop a complete VLIW compiler and a prototype of such a low-power VLIW processor in order to compute the energy consumption more precisely.

Acknowledgments

We are grateful to Rosario Martinez, Jean-Marc Masgonty, Moshe Sipper, Gianluca Tempesti, the Logic Systems Laboratory team, and the processors design team of the CSEM (Centre Suisse d'Electronique et de Microtechnique) for our fruitful discussions. Finally, financial support from the National Science Foundation is gratefully acknowledged.

References

[1] Eduard Ayguadé, Cristina Barrado, Antonio González, Jesús Labarta, David López, Josep Llosa, Susana Moreno, David Padua, Fermín J. Reig, and Mateo Valero. Ictineo: A tool for research on ILP. In *Supercomputing'96*, November 1996.

[2] M. Berry, D. Chen, P. Koss, and D. Kuck. The Perfect Club benchmarks: Effective performance evaluation of supercomputers. Technical Report 827, Center for Supercomputing Research and Development, November 1988.

[3] Thomas D. Burd and Robert W. Brodersen. Energy efficient CMOS microprocessor design. In *Proceedings of the 28th Annual HICSS Conference*, volume 1, pages 288–297, January 1995.

[4] G.H. Chaitin. Register allocation and spilling via graph coloring. In *Proc., ACM SIGPLAN Symp. on Compiler Construction*, pages 98–105, June 1982.

[5] Ricardo Gonzalez and Mark Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1283, September 1996.

[6] Linley Gwennap. Intel,HP make EPIC disclosure. *Microprocessor Report*, 11(14), October 1997.

[7] Mike Johnson. *Superscalar microprocessor design*. Prentice Hall, 1991.

[8] J. Llosa. *Reducing the Impact of Register Pressure on Software Pipelined Loops*. PhD thesis, Universitat Politènica de Catalunya, January 1996.

[9] J. Llosa, A. González, E. Ayguadé, and M. Valero. Swing modulo scheduling: A lifetime-sensitive approach. In *IFIP WG10.3 Working Conference on Parallel Architectures and Compilation Techniques (PACT'96)*, pages 80–86, October 1996.

[10] J. Llosa, M. Valero, and Ayguadé. Quantitative evaluation of register pressure on software pipelined loops. *International Journal of Parallel Programming*, 26(2):121–142, 1998.

[11] J. Llosa, M. Valero, and E. Ayguadé. Heuristics for register-constrained software pipelining. In *Proc. of the 29th Ann. Int. Symp. on Microarchitecture (MICRO-29)*, pages 250–261, December 1996.

[12] Christian Piguet. Low-power 8-bit microcontrollers. Architectural and Circuit Design for Portable Electronic Systems, March 31-April 5 1997. Monterey, CA, USA.

[13] Christian Piguet, Jean-Marc Masgonty, Claude Arm, Serge Durand, Thierry Schneider, Flavio Rampogna, Ciro Scarnera, Christian Iseli, Jean-Paul Bardyn, R. Pache, and Evert Dijkstra. Low-power design of 8-b embedded CoolRISC microcontroller cores. *IEEE Journal Of Solid-State Circuits*, 32(7):1067–1078, July 1997.

[14] Jim Turley. Embedded vendors seek differentiation. *Microprocessor Report*, 11(1), 1997.

[15] Tim Turley and Harri Hakkarainen. TI's new 'C6x DSP screams at 1600 MIPS. *Microprocessor Report*, 11(2), February 1997.

Designing the Low-Power M•CORE™ Architecture

Jeff Scott, Lea Hwang Lee, John Arends, Bill Moyer

M•CORE Technology Center
Semiconductor Products Sector
Motorola, Inc. TX77/F51,
7600-C Capital of Texas Highway, Austin, TX 78731
{jscott,leahwang,arends,billm}@lakewood.sps.mot.com

Abstract

The M•CORE microRISC architecture has been developed to address the growing need for long battery life among today's portable applications. In this paper, we will present the low-power design techniques and architectural trade-offs made during the development of this processor. Specifically, we will discuss the initial benchmarking, the development of the Instruction Set Architecture (ISA), the custom datapath design, and the clocking methodology . Finally, we will discuss two system solutions utilizing the M•CORE processor, presenting power, area, and performance metrics.

1 Introduction

There is significant effort directed toward minimizing the power consumption of digital systems. Minimization techniques have been applied to various levels of abstraction from the circuit to the system level. The industry, coupled with contributions from the research community, has developed new architectures and microarchitectures [22,23], derivatives of existing architectures [1,15], compiler and compression optimizations [16,17], enhanced process technology [10], and new tools [11] and design techniques [5,7,14] to enhance overall system performance, which in this context translates to a decrease in milliwatts per MHz.

Existing architectures are not suitable for low-power applications due to their inefficiency in code density, memory bandwidth requirements, and architectural and implementation complexity. This drove the development of the new general purpose microRISC M•CORE architecture [18,21], which was designed from the ground up to achieve the lowest milliwatts per MHz. The M•CORE instruction set was optimized using benchmarks common to embedded applications coupled with benchmarks targeted specifically for portable applications. Compilers were developed in conjunction with the instruction set to maximize code density.

The benchmarks used to drive the development of the M•CORE architecture, as well as for making design trade-offs, are presented in Section 2. In Section 3, an overview of the instruction set is presented. Section 4 presents implementation details and various design methodologies. Section 5 presents power consumption statistics. Two system solutions based on the M•CORE microprocessor are presented in Section 6. Section 7 summarizes the paper.

2 Benchmarks

Embedded and portable benchmarks were used to make design trade-offs in the architecture and the compiler. The Powerstone benchmarks, which include paging, automobile control, signal processing, imaging, and fax applications, are detailed in Table 1.

Table 1: Powerstone benchmark suite

Benchmark	Instr. Count	Description
auto	17374	Automobile control applications
bilv	21363	Shift, and, or operations
blit	72416	Graphics application
compress	322101	A Unix utility
crc	22805	Cyclic redundancy check
des	510814	Data Encryption Standard
dhry	612713	Dhrystone
engine	986326	Engine control application
fir_int	629166	Integer FIR filter
g3fax	2918109	Group three fax decode (single level image decompression)
g721	231706	Adaptive differential PCM for voice compression
jpeg	9973639	JPEG 24-bit image decompression standard
pocsag	131159	POCSAG communication protocol for paging application
servo	41132	Hard disc drive servo control
summin	3463087	Handwriting recognition
ucbqsort	674165	U.C.B. Quick Sort
v42bis	8155159	Modem encoding/decoding
whet	3028736	Whetstone

3 Instruction Set Architecture

3.1 Overview

The M•CORE instruction set is designed to be an efficient target for high-level language compilers in terms of code density as well as execution cycle count. Integer data types of 8, 16 and 32-bits are supported to ease application migration from existing 8 and 16 bit microcontrollers. A standard set of arithmetic and logical instructions are provided, as well as instruction support for bit operations, byte extraction, data movement, and control

flow modification.

A small set of conditionally executed instructions are available, which can be useful in eliminating short conditional branches. These instructions utilize the current setting of the processor's condition bit to determine whether they are executed. Conditional move, increment, decrement, and clear operations supplement the traditional conditional branch capabilities.

The M•CORE processor provides hardware support for certain operations which are not commonly available in low-cost microcontrollers. These include single-cycle logical shift (LSL, LSR), arithmetic shift (ASR), and rotate operations (ROTL), a single cycle find-first-one instruction (FF1), a hardware loop instruction (LOOPT), and instructions to speed up memory copy and initialization operations (LDQ,STQ). Also provided are instructions which generate an arbitrary power of 2 constant (BGENI, BGENR), as well as bit mask generation (BMASKI) for generating a bit string of '1's ranging from 1 to 32 bits in length. Absolute value (ABS) is available for assisting in magnitude comparisons.

Because of the importance of maximizing real-time control loop performance, hardware support for multiply and divide is also provided (MULT, DIVS, DIVU). These instructions provide an early-out execution model so that results are delivered in the minimum time possible. Certain algorithms also take advantage of the bit reverse instruction (BREV), which is efficiently implemented in the barrel shifter logic [3].

3.2 16-Bit vs. 32-Bit

System cost and power consumption are strongly affected by the memory requirements of the application set. To address this, the M•CORE architecture adopts a compact 16-bit fixed length instruction format, and a 32-bit Load/Store RISC architecture. The result is high code density which reduces the total memory footprint, as well as minimizing the instruction fetch traffic.

Benchmark results on a variety of application tasks indicate that the code density of the M•CORE microRISC engine is higher than many CISC (complex instruction set computer) designs, in spite of the fixed length nature of the encodings. Fixed-length instructions also serve to reduce the CPU's control unit complexity, since instructions have a small number of well structured formats, thus simplifying the decoding process.

3.3 Low-Power Instructions

The M•CORE processor minimizes power dissipation by utilizing dynamic power management. DOZE, WAIT, and STOP power conservation modes provide for comprehensive system power management. These modes are invoked via issuing the appropriate M•CORE instruction. System level design dictates the operation of various components in each of the low power modes, allowing a flexible set of operating conditions which can be tailored to the needs of a particular application. For example, a system may disable all unnecessary peripherals in DOZE mode, while in STOP mode, all activity may be disabled. For short term idle conditions, the WAIT mode may disable only the CPU, leaving peripheral functions actively operating [18].

3.4 Instruction Usage

The M•CORE ISA was profiled by running the Powerstone benchmark suite on a cycle accurate C++ simulator. Table 2 shows the percentage of dynamic instructions utilizing the adder and barrel shifter, as well as the percentage of change of flow and load/store instructions.

Table 2: Dynamic instruction percentages

Type	Dyn. Inst. Percentage
Adder usage	50.23%
Barrel shifter usage	9.68%
Change of flow instructions[a]	17.04%
Load/store instructions	22.46%

a. 83.5% of change of flow instructions are taken.

4 Processor Implementation

The initial implementation of the M•CORE architecture was targeted at sub-2.0 volt operation from 0-40 MHz and achieves both low-power consumption and high-performance in a 0.36-micron low-leakage static CMOS process. In addition, a die size of just over 2 mm^2 was achieved by utilizing a synthesizable control section and full-custom datapath. The first implementation is also being produced in a 0.31-micron process for a cellular application and has been targeted for production in a 0.25-micron process for other embedded applications. Other low-power process technologies are being investigated including a Silicon On Insulator (SOI) process [2].

Power consumption was minimized in the first implementation of the M•CORE architecture by: (i) gating clocks; (ii) utilizing delayed clocks to minimize glitching; and (iii) optimizing the clock tree. Powermill [12], Spice, and Focus [8], a transistor sizing power optimization tool, were used to optimize the circuits.

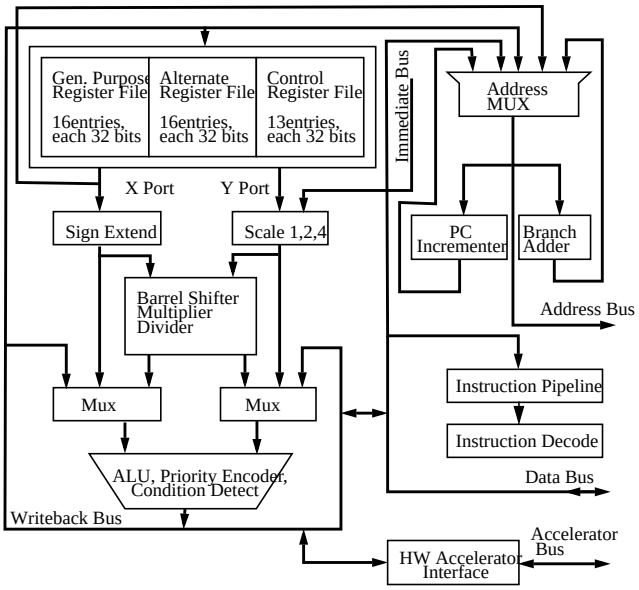


Figure 1: M•CORE datapath

4.1 Datapath Design

The M•CORE datapath consists of a register file, an instruction register, an adder, a logical unit, a program counter, a branch adder, and a barrel shifter. A block diagram of the datapath is shown in Figure 1.

4.1.1 Synthesizable vs. Custom

Synthesis methodologies have the advantage of decreased time to market, fewer resource requirements, and the ability to quickly migrate to different technologies. Custom designs lack those benefits, but can achieve faster performance, consume less power, and occupy less area.

Research was conducted to determine whether to pursue a synthesizable or custom datapath design. An energy-delay metric [13] was used to develop a custom 32-bit adder. Our studies showed that in going from a custom to a synthesized adder, transistor count increased by 60%, area increased by 175%, and power consumption increased by 40%. This research coupled with similar block-level comparisons led to the development of a full-custom datapath.

4.1.2 Speed, Area and Power Trade-off

Focus [8] was used to optimize speed, area and power in the custom datapath. Focus takes a user-specified circuit and timing constraints and generates a family of solutions which form a performance/area trade-off curve. Given a set of input vectors, Focus allows the user to invoke Powermill interactively to measure power at any given solution. The user is then allowed to select the solution with the desired performance, area, and power characteristics.

Focus uses a dc-connected component-based static timing analysis technique to propagate the arrival times in a forward pass through the circuit, and the required times in a backward pass. At each node, a slack is generated as the difference between the required and the arrival times (negative slack denotes a violation of a constraint).

Focus performs a hill climbing search starting from a minimum sized circuit (or a given initial circuit). In each step, Focus picks a set of candidate transistors to size based on their sensitivity to the overall path delay, and iteratively and gradually approaches a solution that will meet the timing constraints[8].

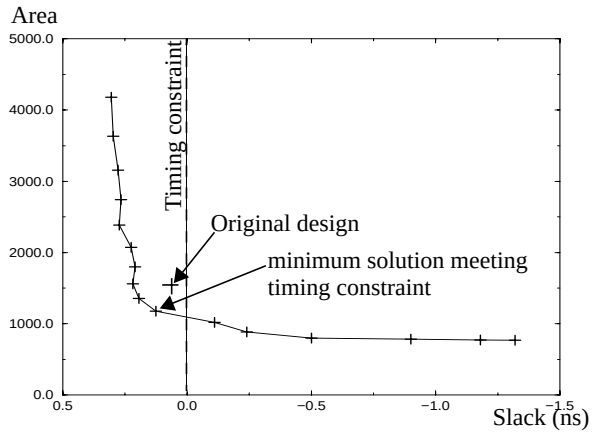


Figure 2: Speed and area trade-off

Figure 2 shows an example of how Focus was used to design the logical bit in the ALU. The vertical axis is expressed in some normalized area cost, while the horizontal axis represents the slack in nanoseconds.

The figure shows that Focus gives a family of solutions. It also shows that faster timing can be achieved at the expense of higher area cost. Utilizing this family of solutions, a solution was selected to meet the timing constraints with optimal area and power savings.

4.1.3 Floorplanning

Floorplanning of the datapath blocks was based on bus switching activity, timing, and block usage percentages derived from the Powerstone benchmarks. Blocks having output busses with high switching activity, tight timing requirements, and high usage were placed close together to minimize the routing capacitance. When searching through the solution space of block placement, timing constraints were used as hard constraints, while switching activities, weighted by their appropriate routing distances and block usage, were used as a cost function in the iterative search process.

In the final floorplan, shown in Figure 3, each horizontal line represents a multiple bit (up to 32 bits) data bus route. The datapath could be roughly partitioned into two halves, separated by the register file: the left half includes the address and data interface, the instruction register, the branch adder, and the address generation unit; the right half includes the execution units.

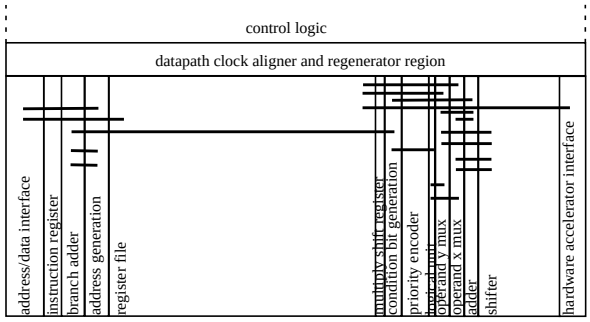


Figure 3: Datapath floorplan

4.1.4 Register File Design

An architectural decision was made to have sixteen general purpose registers based on the limited opcode space. An alternate set of sixteen 32-bit registers is provided as well, to reduce the overhead associated with context switching and saving/restoring for time-critical tasks. This alternate register file can be selected via a bit (AF) in the processor status register (PSR). Upon detecting an exception, the hardware places bit 0 of the exception vector table entry into the AF bit in the PSR. This allows the exception handler to select either register file for exception processing with no cycle penalty [20]. Hardware does not preclude access to the alternate register file in user mode, resulting in a larger working set of registers. This can result in a reduced number of memory fetches, thereby saving power.

In supervisor mode, an additional 5 supervisor scratch registers can be accessed by means of the move-to-control-register (MTCR) and move-from-control-register (MFCR) instructions.

The register file also includes: (i) feed forwarding logic; (ii) immediate muxes to produce various constants; and (iii) scale logic for computing load/store addresses.

4.1.5 Barrel Shifter Design

As noted earlier, approximately 10% of the instructions executed in the Powerstone benchmarks utilize the barrel shifter. In addition, barrel shifters are typically one of the highest power consuming modules in datapath designs. A comparison was made between two competing barrel shifters to determine which one offered the least current drain within a given performance requirement.

"Snaked Data" Barrel Shifter

"Snaked data" barrel shifters contain a 32x32 array of transistors. For precharged shifter outputs, the array uses nmos transistors. The term "snaked data" comes from the fact that input data (IL[2:0],IR[3:0]) is driven diagonally across the array, as shown in Figure 4. Control signals (S[3-0]) to select the amount of the shift are driven vertically and outputs (O[3:0]) are driven horizontally. Data enters the shifter through multiplexers with selection based on the type of shift operation, either rotates, logicals, or arithmetics. The control signals are generated from a 5-32 decoder for 32-bit shifters, where 5 bits represent the amount to shift and can be inverted to select shifts in the opposite direction.

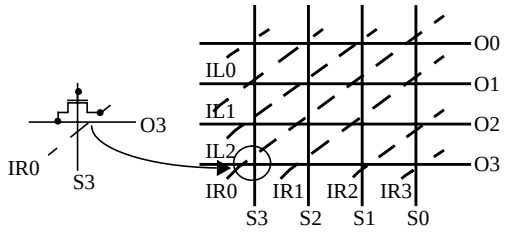


Figure 4: 4-bit "snaked data" barrel shifter

"Snaked Control" Barrel Shifter

"Snaked control" barrel shifters contain a 32x32 array of transistors. For precharged shifter outputs, the array uses nmos transistors. The term "snaked control" comes from the fact that control signals (SL[3:1],SR[3:0]) are driven diagonally across the array as shown in Figure 5. Shifter outputs (O[3:0]) are driven vertically and inputs (I[3:0]) are driven horizontally. Control signal assertion is based on the type of shift operation, either rotates, logicals, or arithmetics and the amount of the shift. The control signals require a 5-32 decoder, where 5 bits represent the amount of shift, as well as a greater-than decoder for arithmetic operations.

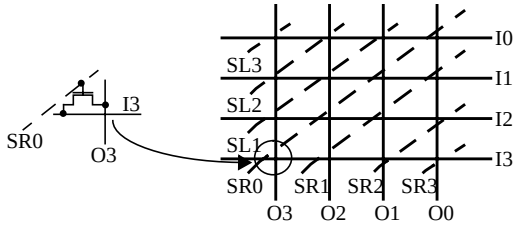


Figure 5: 4-bit "snaked control" barrel shifter

Barrel Shifter Selection

There are three components of routing in either shifter design. Table 3 shows the routing breakdown for the two 32-bit shifters.

Table 3: Snaked control vs. snaked data routing

Shifter Type	Input Data Routing	Output Data Routing	Control Routing
Snaked control	12,896	173,056	171,962
Snaked data	171,962	24,576	173,056

Control signal power consumption is negligible in both cases, since the selections are one-hot. Even though the total length of power consuming data routing for the two designs is nearly the same, the "snaked control" barrel shifter actually has significantly less routing capacitance, 5.9 pF vs. 7.4 pF, because of metal layer usage.

It would appear that the "snaked control" would be the ideal choice for a power conscious design. However, further analysis of the Powerstone benchmarks was required. Many benchmarks perform similar types of shifts in succession. In fact, 64% of the shifts are either logical shift left to logical shift left, or logical shift right to logical shift right. Under these conditions, the "snaked data" shifter maintains constant data input into half of the array throughout the entire algorithm. In these circumstances, the "snaked data" shifter toggles over 30% less capacitance then the "snaked control" barrel shifter.

4.2 Clocking System Design

Clock gating saves significant power by eliminating unnecessary transitions of the latching clocks. Furthermore, it also eliminates unnecessary transitioning of downstream logic connected to the output of the latches. In the datapath, delayed clocks are generated for late arriving latch input data in order to further minimize downstream glitching.

Power due to clocking can vary anywhere from 30% to 50% in a power-optimized design [4]. As more and more datapath power optimizations were added to the design, the percentage of power due to clocking increased. Therefore, it was crucial to aggressively attack clock power consumption.

The M•CORE processor uses a single global clock with local generation of dual phase non-overlapping clocks. The global clock proceeds through two levels of buffering/gating; a clock *aligner* and *regenerator*, as shown in Figure 6 and Figure 7 respectively. The aligner generates the dual phase clocks, while the regenerator provides local buffering.

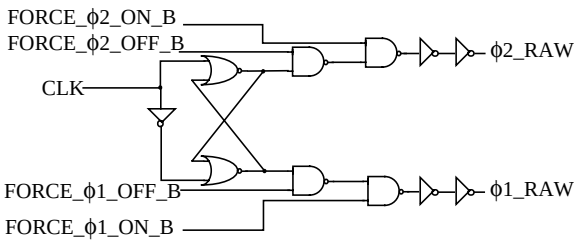


Figure 6: Clock aligner

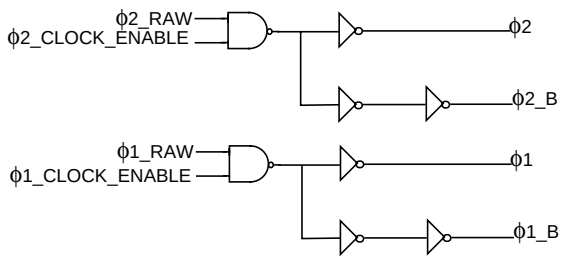


Figure 7: Clock regenerator

Clock gating can be performed at the aligners and/or the regenerators. This allows for complete or partial clock tree disabling. Besides shutting down the system clock input completely, gating at the aligners allows for maximum power savings by shutting down a collection of regenerators. In the instruction decoder, for example, the clocking for over 60 latches and flip-flops is disabled during pipeline stalls.

5 Power Measurement

Powermill and Pathmill were used to perform the following tasks: (i) measuring power usage of each block to detect any design anomalies; (ii) detecting nodes with excessive DC leakage current; (iii) locating hot spots in the design; and (iv) identifying nodes with excessive rise and fall times. Various design changes were made based on these findings.

5.1 Processor Power Distribution

M•CORE processor power was measured using Powermill on a back-annotated taped out implementation of the architecture. Clock power is 36% of the total processor power consumption, as shown in Figure 8. It is important to note that due to measurement constraints, the datapath regenerator clocking power is included in the datapath percentage, not the clocking power percentage. This clocking power would push the clock power percentage even higher. This is a good indication that the datapath and control sections are well optimized for power. It is also important to note that through utilization of the low-power mode instructions, the system clock can be completely shut down.

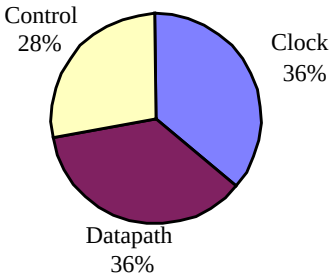


Figure 8: Processor total power distribution

5.2 Datapath Power Consumption

Figure 9 shows the breakdown of the datapath power consumption. The register file consumes 16% of total processor power consumption and 42% of the datapath power consumption. It is the largest consumer of datapath power and occupies 46% of the datapath area as well. The barrel shifter consumes less than 4% of the total processor power consumption and 8% of the datapath power consumption.

The address/data interface contains the local bus drivers. The instruction register drives the instruction decoder in the control section. The operand muxes drive the ALU, priority encoder, and condition detect logic. The address generation unit contains the program counter and dedicated branch adder circuitry (see Figure 1).

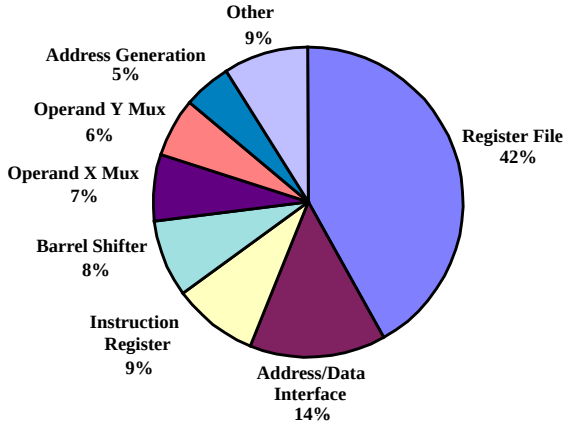


Figure 9: Datapath power consumption

6 System Solutions

6.1 MMC2001

The MMC2001 low power platform incorporates an M•CORE integer processor, an on-chip memory system, a OnCE™ debug module, an external interface module, and peripheral devices [19]. It includes a section of 20K gates that are customizable. The MMC2001's production version is implemented in a 0.36 micron, triple-layer metal low-leakage static CMOS process. The on-chip memory system consists of 256 KBytes of ROM and 32 KBytes of SRAM. The device measures 7.23 mm on a side or 52 sq. mm. The part is packaged in a 144 pin Thin Quad Flat Package (TQFP) and achieves 34 MHz performance at 1.8 V.

The MMC2001, when running out of internal SRAM, consumes on average less than 30 mA at 2.0 V, which translates into less than 60 mW at 34 MHz for the complete system. On average, this implementation of the M•CORE processor consumes 7 mA, which translates to a 0.40 mW/MHz rating. When the part is put into STOP mode, the part consumes less than 50 microamps with the clock input shut down. When the part goes into a low voltage detect condition with just the oscillator, SRAM backup and time-of-day timer running, the part consumes less than 5 microamps.

6.2 MMC56652

The MMC56652 Dual-Core Baseband Processor incorporates

an M•CORE integer processor, an on-chip memory system, a OnCE™ debug module, an external interface module, and peripheral devices [9]. It includes a 56600 Onyx Ultra-Lite 16-bit digital signal processor, program and data memory, and peripheral devices. The MMC56652's production version is implemented in a 0.31 micron, triple-layer metal static CMOS process. This device consists of 8 KBytes of ROM and 2 KBytes of SRAM to support the M•CORE processor. The chip measures 7.4 mm on a side or 55 sq. mm. The part is packaged in a 196 plastic ball grid array (PBGA) and was designed for 16.8 MHz performance at 1.8 V.

The MMC56652 when running out of internal SRAM consumes on average less than 9 mA at 1.8 V, which translates into less than 16.2 mW at 16.8 MHz for the complete system. On average, this implementation of the M•CORE processor consumes 2.8 mA, which translates to a 0.30 mW/MHz rating. The part consumes less than 60 microamps in STOP mode.

7 Conclusion

The M•CORE architecture development represents four years of research in the area of low-power embedded microprocessor and system design. This research encompassed all levels of abstraction from application software, system solutions, compiler technologies, architecture, circuits, and process optimizations.

The instruction set architecture was developed in conjunction with benchmark analysis and compiler optimization. A full-custom datapath design with clock gating was utilized to minimize the power consumption. Two system solutions integrating the M•CORE microprocessor were presented along with power, area, and performance metrics.

Further research will involve allocating unutilized opcode space, along with additional compiler enhancements. Benchmarks will continue to be added to the Powerstone suite reflecting future portable low-power applications. Efforts will be made to drive novel design techniques and process technologies into a production environment in order to achieve higher levels of power reduction and performance.

8 References

[1] *An Introduction to Thumb*, Advanced RISC Machines Ltd., March 1995.

[2] D. Antoniadis, "SOI CMOS as a Mainstream Low-Power Technology," *Proc. IEEE Int'l. Symp. Low Power Electronics and Design*, August 1997, pp. 295-300.

[3] J. Arends, J. Scott, "*Low Power Consumption Circuit and Method of Operation for Implementing Shifts and Bit Reversals*," U. S. Patent, number 5,682,340, October 28, 1997.

[4] R.S. Bajwa, N. Schumann, H. Kojima, "Power Analysis of a 32-bit RISC Microcontroller Integrated with a 16-Bit DSP," *Proc. IEEE Int'l Symp. Low Power Electronics and Design*, August, 1997, pp. 137-142.

[5] L. Benini, G. Micheli, "Transformation and Synthesis of FSMs for Low-Power Gated-Clock Implementation," *Proc. IEEE Int'l. Symp. on Low Power Design*, 1995, pp. 21-26.

[6] J. Bunda, D Fussell, R. Jenevein, W. C. Athas, "16-Bit vs. 32-Bit Instructions for Pipelined Microprocessors," University of Texas, Austin, October 1992.

[7] A. Chandrakasan, S. Sheng, R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, April 1992, pp. 473-484.

[8] A. Dharchoudhury, D. Blaauw, J. Norton, S. Pullela, J. Dunning, "Transistor-level Sizing and Timing Verification of Domino Circuits in the PowerPC Microprocessor," *Proc. IEEE Int'l. Conf. Computer Design*, Austin, Texas, October 1997.

[9] *DSP 56652 User's Manual*, Motorola Inc., 1998.

[10] D. Frank, P. Solomon, S. Reynolds, and J. Shin, "Supply and Threshold Voltage Optimizations for Low Power Design," *Proc. IEEE Int'l. Symp. Low Power Electronics and Design*, August 1997, pp. 317-322.

[11] J. Frenkil, "Issues and Directions in Low Power Design Tools: An Industrial Perspective," *Proc. IEEE Int'l Symp. Low Power Electronics and Design*. August 1997, pp. 152-157.

[12] C. Huang, B. Zhang, A. Deng, B. Swirski, "The Design and Implementation of Powermill," *Proc. IEEE Int'l Symp. Low Power Design*, 1995, pp. 105-109.

[13] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-Power Digital Design," *Proc. IEEE Int'l. Symp. on Low Power Electronics"*, October, 1994, pp. 8-11.

[14] M. Igarashi, K. Usami, K. Nogami, F. Minami, "A Low-Power Design Method Using Multiple Supply Voltages," *Proc. IEEE Int'l. Symp. Low Power Electronics and Design*, August 1997, pp. 36-41.

[15] K. Kissell, "MIPS16: High Density MIPS for the Embedded Market," Silicon Graphics MIPS Group, 1997.

[16] M. Kozuch and A. Wolfe, "Compression of embedded system programs," *Proc. IEEE Int'l. Conf. on Computer Design*, 1994.

[17] C. Lefurgy, P. Bird, I. Chen, and T. Mudge, "Improving Code Density Using Compression Techniques," *Proc. IEEE Int'l. Symp. on Microarchitecture*, December, 1997, pp. 194-203.

[18] *M•CORE Reference Manual*, Motorola Inc., 1997.

[19] *MMC2001 Reference Manual*, Motorola Inc., 1998.

[20] W. Moyer, J. Arends, patent pending, "*Method and Apparatus for Selecting a Register File in a Data Processing System*," U. S. Patent, filed September, 1996.

[21] W. Moyer, J. Arends, "RISC Gets Small," *Byte Magazine*, February 1998.

[22] "NEC Introduces 32-bit RISC V832 Microcontroller," NEC Inc., May 1998.

[23] J. Slager, "SH-4: A Low-Power RISC for Multimedia and Personal Access Systems," *Microprocessor Forum*, October 1996.

Author Index