

Dual Issue Power PC FXU

Jayakumaran Sivagnaname, Rahul Rao and Richard B. Brown
{jsivagna, rmrao, brown@eecs.umich.edu}
EECS Dept., University of Michigan, Ann Arbor, MI 48109-2122

Abstract - *The design of a 32-bit dual-issue 4-way super scalar PowerPC fixed point unit microprocessor (PUMA) in a 0.18mm TSMC process is described. The architectural details of the PUMA system are described. The design considerations and the implementation details are elaborated. This processor will be used as a design vehicle for analysis and development of circuit design techniques in SOI technology.*

1. Introduction

A single issue, out-of-order super scalar fixed-point unit (FXU) based on the PowerPC instruction set was designed as part of the PUMA project at the University of Michigan. The processor implements a majority of the integer instructions of the PowerPC ISA. The FXU was created as a test vehicle for Motorola's 0.5 μ m Complementary Gallium Arsenide Process (CGaAs)[1]. Its approach was to provide advanced micro-architecture techniques in a small transistor budget. The architecture was designed to find the optimal number of execution units, issue-width, branch prediction, etc. while reducing the total transistor count.

PUMA went through several design stages as the architecture was developed, but wound up being implemented in two forms. The first implementation is the single-issue machine that was designed in behavioral verilog. The code was hand-optimized and mapped to a 0.35 μ m process. This architecture is slightly different from the optimal architecture found in [2] pertaining to the issue width and the size of some of the supporting functional units. The purpose of the CMOS version of the PUMA processor was to test the processor in a known good process. The second implementation of PUMA was the CGaAs version. It is a scaled down version of the CMOS processor to make it adhere to the smaller transistor budget of CGaAs.

We selected the PUMA FXU processor as the design vehicle for studying circuit implementations in Partially Depleted Silicon-on-Insulator (PD-SOI) technology. The design has been modified to allow dual-issue for increased performance and implemented in a 0.18 μ m CMOS TSMC process. This document describes the PUMA system and the modifications made for this purpose. The final product is intended to be a soft IP core which could be used as a processor for design projects, as a benchmark design for future CAD

tool research, in addition to being used for the analysis and development of circuit design techniques in SOI.

2. PUMA PowerPC FXU Architecture

This section documents the PUMA PowerPC architecture. A detailed documentation can be found in [2].

A simplified block diagram of the FXU is shown in Fig. 1. The processor has a split level-1 cache and unified off-chip level-2 cache. The chip interfaces the level-2 cache through a 128-bit data bus and a 32-bit address bus. The address bus sends the requested load or store address to the second level memory management unit. Data is written on the 32-bit bus and read across the 128-bit bus. The cache line is 128-bits, so a full line is read for each second-level access. The data memory access queue (DMAQ) is the portal for the 128-bit bus and routes data and instructions to the respective on-chip cache. The instruction/control portion of the machine is composed of an instruction cache, fetch unit, decoder, and branch predictor. The decoded instructions are issued to the execution core through the dispatch unit and written back to the register file or re-order buffer. Four functional units have been implemented in the processor: one branch unit, two ALUs, and one load-store unit. The following sections describe each of these modules in greater detail.

2.1. Stream Unit

The main function of the stream unit is to pre-fetch instructions and feed a continuous stream of instructions to the FXU units. It has an eight entry buffer and each entry of the buffer can hold 4 words (or 16 bytes). When an icache miss occurs, the stream buffer is checked to determine if the requested instruction address is present in the stream buffer. If there is a stream hit, the instruction is immediately forwarded to the icache and fetch unit. In case of stream miss, the whole stream buffer is flushed and the stream unit interacts with the IMAQ unit to fetch instructions from level 2 cache. A critical word first approach is employed while accessing the higher level of memory. The stream unit fetches the entire block of eight lines adjacent to the most recent stream miss address and stores it in the stream buffer. In addition, the stream unit initializes the instruction cache during the start up routine.

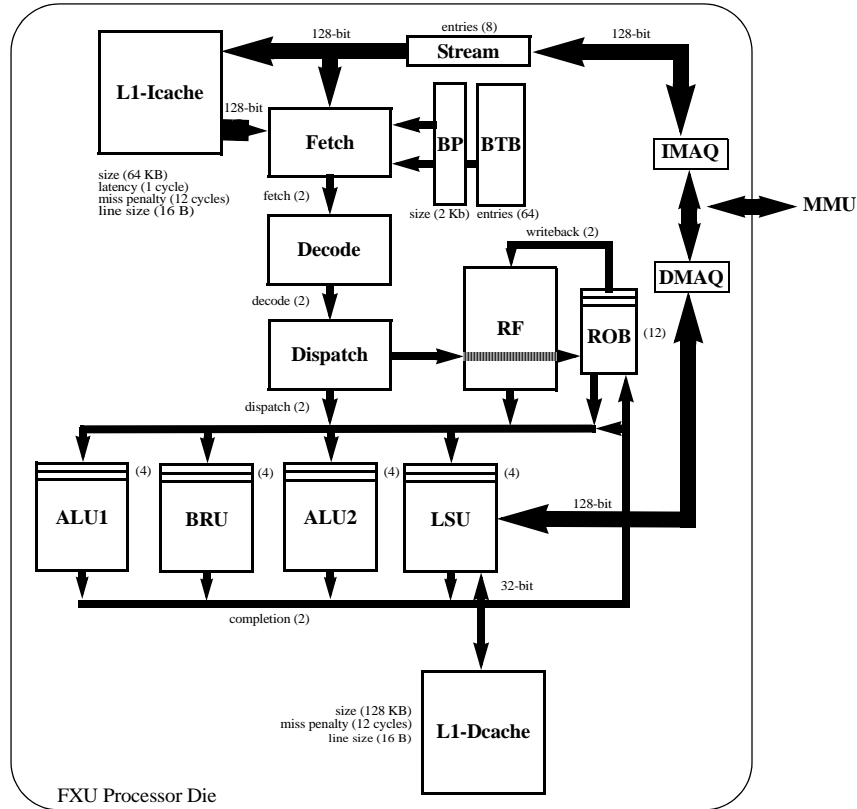


Figure 1. PUMA FXU Block Diagram

2.2. Instruction Memory Access Queue (IMAQ)

The IMAQ serves as an interface between the DMAQ and the stream unit. It maintains a buffer similar to the stream unit. While it continues to receive requests from the stream (unless its buffer is full), these instruction requests are passed on to the DMAQ. Once the DMAQ responds, the stream is notified of the availability of the requested instruction. The IMAQ operates in 3 clock domains: CLK (for interfacing with stream unit), MMU_CLK (to send requests to DMAQ) and IMMU_RESP_CLK (to receive response from the higher level of memory).

2.3. Data Memory Access Queue (DMAQ)

The DMAQ unit is the interface between the FXU chip and the memory management unit (higher level of memory). It forwards instruction requests (from IMAQ) and data requests (from the data cache and the execution core) to the MMU. It maintains a buffer similar to the stream unit. The data requests are given a higher priority as compared to the requests from IMAQ. The DMAQ also does the initialization of the data-cache during the start up routine.

2.4. Instruction Cache (Icache)

This is the level-1 128KB instruction cache organized as a direct mapped cache having 8192 lines with a cache line size of 128 bits. The instruction cache is written each time there is a hit in the stream buffer and the address is determined by the stream buffer entry. Each entry in the instruction cache contains the data input from L2 (set of 4 instructions), address tag of the current instruction from L2 and the valid/invalid bit. In case of a Icache read, the appropriate program counter bits from the fetch unit are used to index the Icache.

2.5. Data Cache (Dcache)

The 128Kb level-1 data cache is also organized as a direct mapped cache of 8192 lines with a cache line size of 128 bits. The L1 data cache data can be written from either the L2 cache or the processor core. The lower two bits from the address are used for selecting a particular 32-bit word within a line. Individual bytes in a word can be written to by asserting the corresponding bits in write enable signal. While writing from the L2 cache, an entire line is written (128 bits), at a location determined by the load address previously dispatched to the higher level of memory (and maintained in the

DMAQ). When writing from the core, a particular word is selected for writing at the given location depending on the lower two address bits (as the execution unit handles 32-bit data). The read address for the Dcache is determined by the load store unit in the execution core and the particular word from the selected line is read out.

2.6. Branch Target Buffer (BTB)

A 64-entry branch target buffer is used to predict a branch in the fetch stage itself by comparing the program counter of the present instruction (from the fetch unit) to the tag stored in the BTB corresponding to the present fetch index. BTB is written by the core unit when it needs to be updated in case of a mis-predicted branch.

2.7. Branch Predictor (BP)

The branch predictor unit is used for branch prediction in the fetch stage. It interfaces with the fetch unit and the core. A 1-KB predictor is implemented using a global sharing (gshare) scheme. The gshare predictor is basically an implementation of the global branch history and global pattern history scheme where the 1024 entry pattern history table is indexed using a hash of the program counter and the 10-bit branch history register [2]. The branch history register is updated each time an exception is flagged by the core, while the pattern history table is updated when each branch instruction is committed.

2.8. Fetch

The fetch unit is responsible for fetching the appropriate instructions and passing them to stages down the pipeline. The fetch unit determines the current program counter, fetches the instruction from the icache or stream buffer and sends two instructions per clock cycle to the decode_issue unit. The fetch unit has three

sub-units: `pc_select`, `fetch_latch` and `instr_buffer`. The organization of the fetch unit is shown in Fig.2.

`pc_select` is responsible for selecting the appropriate program counter value for the next instruction. This is determined depending on the presence of an exception (from the core) and the prediction from the branch predictor and branch target buffer.

The `fetch_latch` module within the fetch unit reads in values from the `pc_select`, stream buffer, branch predictor, branch target buffer and the Icache (instruction, valid bit, tag, etc.). It checks for the presence of stalls in the `instr_buffer`. The `fetch_latch` determines whether the current instruction is present in the Icache or stream buffer in the absence of stalls and selects the appropriate data in case of a hit. In case of a miss, it generates a miss flag that is used by the stream unit to send the requests to IMAQ. It also outputs the PC value, the instruction and other condition bits to the `instr_buffer`.

The `instr_buffer` consists of two entries, each capable of holding an entire cache line, allowing a total of eight instructions to be buffered. This enables the `fetch_latch` and the preceding units to process the next instruction while the subsequent units in the pipeline are experiencing a stall condition.

2.9. Decode_Issue Unit

The `decode_issue` unit translates the 32-bit PowerPC instructions into unit-operands (uops) to be processed by the execution core and sends two instructions per cycle to the dispatch unit. The `decode_issue` unit primarily consists of two simple decoders followed by a finite state machine.uops. A block diagram of the simple decoder is shown in Fig.3.

The decoder essentially decodes the complex PowerPC instruction into simple uops, some of which result in multi cycle unit operands. The decoder consists of a primary decoder (`idecode`) which translates the PowerPC instruction into the first unit-operand and an index

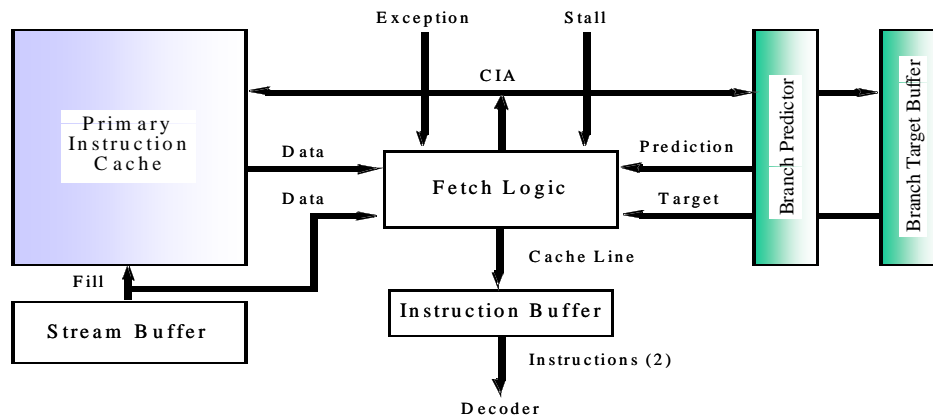


Figure 2. Fetch Unit Organization

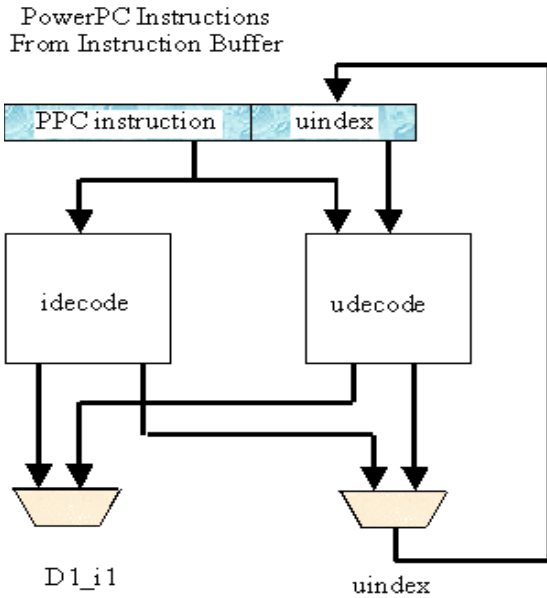


Figure 3. Simple Decoder Block Diagram

(uindex) to the next unit-operation in the sequence. The secondary decoder (udecode) uses the PowerPC instruction and the index to produce subsequent unit-operands. When the secondary decoder has finished translating the PowerPC instruction, control is returned to the idecode to begin the processing of the next PowerPC instruction.

The PUMA instruction set consists of 130 instructions, of which 70 are composed of single unit operations and the rest have multiple uops. According to [2], the decoder for the dual-issue processor should consist of two copies of the single-issue decoder. This would issue two instructions per cycle only when both the

instructions are single unit operation instructions. Even if one of the instructions is a multi-unit operation instruction, only one instruction would be issued to the dispatch unit each clock cycle. Since the static instruction set consists of roughly 50% of instructions which are multiple unit-operation instructions, this design would cause a bottleneck and degrade the performance of the dual-issue processor to the single-issue processor for almost 75% of cases. To overcome this limitation, the decoder was re-designed such that it is capable of issuing two unit operations per cycle. This is done by enabling the idecode and the udecode to generate two instructions each per cycle. Additional control signals are generated by each of the decoders which indicate whether the original PowerPC instruction was a single uop instruction, dual uop instruction, or a multi uop instruction. A finite state machine selects the appropriate instructions from these decoders based on these signals and sends them to the dispatch unit. The overall organization of the Decode_Issue unit is shown in Fig.4.

The selection procedure works as follows:.

1. If the PowerPC instructions obtained from fetch are single uop instructions, then both of them would be passed in the same clock cycle (same as the design suggested in [2])
2. If one of the instructions is a single uop instruction and the other is a multiple uop instruction then the single uop would be sent in one clock cycle and the two uops of the multiple uop instruction would be sent in the next clock cycle.
3. If both the instructions are multiple uop instructions, then two uops from one of the decoded instruction

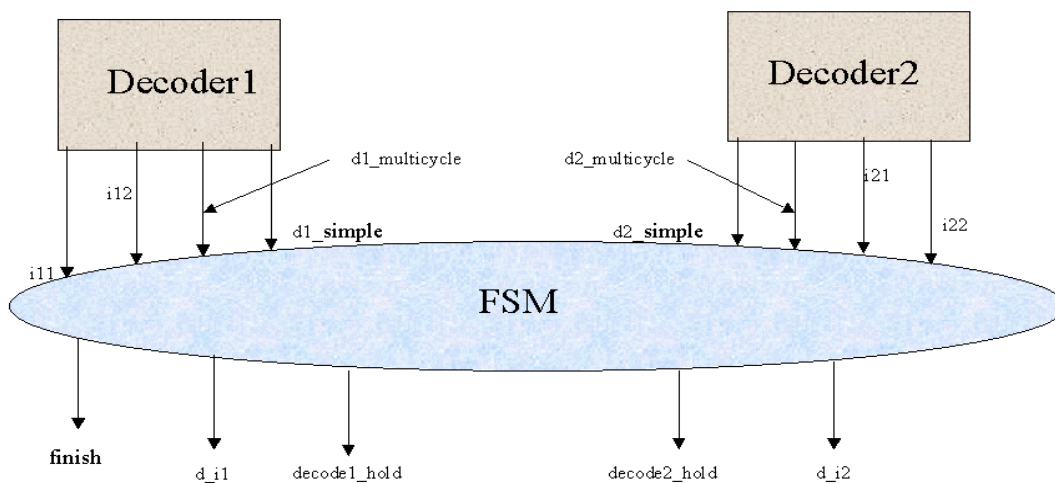


Figure 4. Decode Issue Organization

would be sent per clock cycle followed by the decoded UOPs from the other instruction.

2.10. Dispatch Unit

The dispatch unit performs the scheduling of the instructions. It is responsible for routing the uops and the required operands to the reservation stations in the functional units. Fig. 5 shows the communication between the dispatch unit and the other blocks in the processor.

The decode_issue unit specifies the required operands by producing an architectural (or scratch) register index corresponding to each source operand. The dispatch unit uses this index to access the register file and the reorder buffer. The output of the register file reflects the most recent state of the machine on the basis of the last committed instruction. The reorder buffer maintains speculative state information of the in-flight instructions. If the reorder buffer detects a match of the index to a speculative state, it will return the value (or a tag) corresponding to the result. The reservation stations use this tag to obtain the value when the results are forwarded on the common data bus after execution. If no speculative state is found, the dispatch unit forwards the output of the register file to the reservation stations as the source operands.

The dispatch unit is implemented as a finite state machine. The dispatch unit determines the availability of free entries in the re-order buffer. If more than two entries are available, two instructions are issued per clock cycle. If the re order buffer has only one free entry, the dispatch unit issues the first instruction and issues the next instruction when the next entry is freed in the reorder buffer and generates a stall to the preceding stage in the pipeline. When no free entries are avail-

able, both the instructions are held until some instruction finishes execution and an entry is available in the re order buffer.

Since the reservation stations in the functional unit can accept only one instruction per cycle, the dispatch unit checks if both the instructions are intended for the same functional unit. If both the instructions are ALU instructions, one instruction is sent to each of the ALUs. However, if both of the instructions are either control instructions or load-store instructions, the first instruction is issued and the second instruction is issued in the subsequent cycle.

The dispatch unit also checks for stalls in the reservation stations of the functional stations (indicating that no free entry exists in the reservation stations). In such a case, the dispatch unit stall the current instructions until a free entry is available in the reservation station.

2.11. Execution Core

The execution Core consists of the functional units and their associated reservation stations. Four functional units are implemented in the execution core, namely two Arithmetic and Logic Units, one Branch Resolution Unit, and one Load Store Unit.

Each functional unit has a set of reservation stations (distributed reservation station scheme) and a result register. When all the source operands necessary for the uop are ready, the uop proceeds from the reservation stations to the functional unit (in the order in which it was pushed into the reservation station). The reservation stations thus act as a buffer between instruction schedule and instruction issue, wherein the dispatch unit can schedule subsequent instructions, while the current instruction waits in the reservation station for its operands.

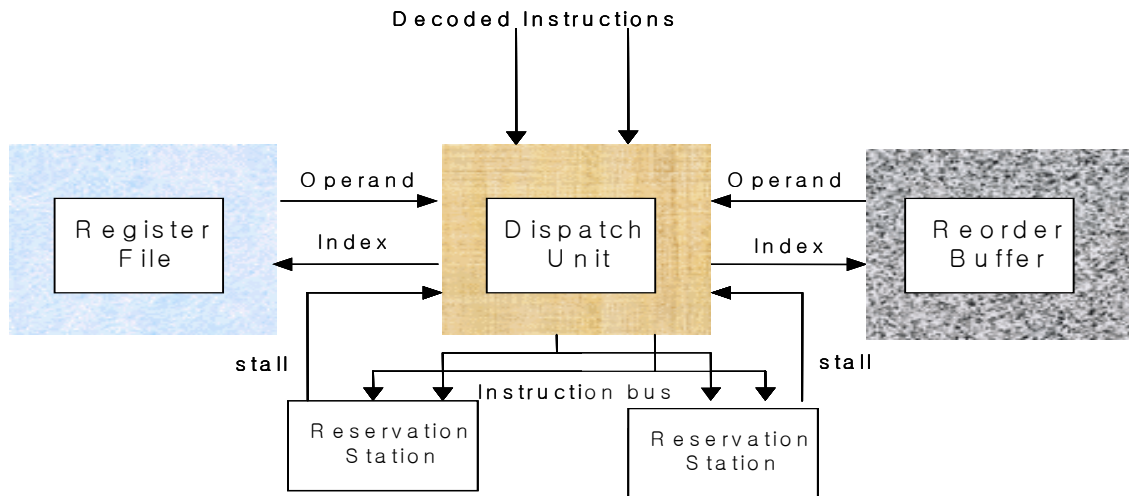


Figure 5. Dispatch Organization

All ALU and control uops are executed in a single cycle. The load store unit is a multi cycle unit designed to interface with the DMAQ in case of a dcache miss. A miss status hold register is employed in the load store unit that enables load instructions that hit the dcache to complete execution while a dcache miss is being serviced by the DMAQ and higher level of memory. The result register is used to prevent contention on the common data bus. The results are held in the result register until a common data bus is available for broadcast. The presence of result registers divides the execution delay and the writeback delay into two clock cycles, thereby eliminating a potential critical path.

2.12. Reorder Buffer

The reorder buffer is used to hold the status of all in-flight instructions and the register mappings. A modified Tomasulo's algorithm is employed for out of order execution management. The reorder buffer maintains speculative state information and is used to track the dependencies between the in-flight instructions and the instruction that needs to be scheduled. The basic organization of the reorder buffer is shown in Fig.6.

The reorder buffer contains a 12-entry first in first out (FIFO) buffer that keeps track of the current instruction sequence and is used to commit instructions. The FIFO is written into when the instruction is scheduled and the entry is popped out of the FIFO when the instruction is committed. Two entries can be pushed and popped out of the FIFO in each clock cycle.

The register mappings are maintained in the Register Alias Table (RAT). During instruction schedule, the reorder buffer maps the destination architectural regis-

ter into a unique physical register. Thus, each in-flight instruction in the machine has a unique physical register entry in the reorder buffer. This physical register entry is used to identify the dependencies during instruction schedule and the entry to be emptied on instruction commit. The speculative state contents of the reorder buffer are cleared when an exception occurs.

2.13. Write Back Unit

The write back unit identifies the exception conditions and updates the architectural state of the processor. Architectural state is updated by committing the speculative state results from the reorder buffer to the architectural register file. All instructions are committed in order based on the FIFO entries present in the reorder buffer. The write back unit can commit two instruction every clock cycle.

A dependency check is performed to ensure that the architectural register is updated in correct order when both the committed instructions update the same architectural register. The write back unit also generates the write enable signals for the front end prediction units (branch predictor and branch target buffer) in case of an exception or branch.

2.14. Register File

The register file consists of 32 general purpose registers and 16 miscellaneous registers. A dual - write port register file has been implemented that enables two instructions to be committed per clock cycle. Four read ports have been provided in order to access the source operands for two instructions per cycle. The miscellaneous registers store the machine state information and also include the scratch registers that are used to store the intermediate results in case of a multi-cycle complex instruction. The register file is written to during the second phase of the write back cycle at the negative edge of the clock cycle.

3. Pipeline Details

The nine stage PUMA pipeline is shown in Fig.7. The first stage, ICA is the instruction cache access in which the instruction cache and branch predictor are accessed and the next PC value is formed. ICH, Instruction Cache Hit, determines whether the cache or stream buffer have been hit and selects between the two for the correct instruction. IB, Instruction Buffer, is where instruction words are aligned for decoding. These three stages of the pipeline are the fetch unit. D1 is the first stage in the decode cycle, and includes the translation of the PowerPC instructions into simple unit operands. D2 is the second stage in the decode cycle, in

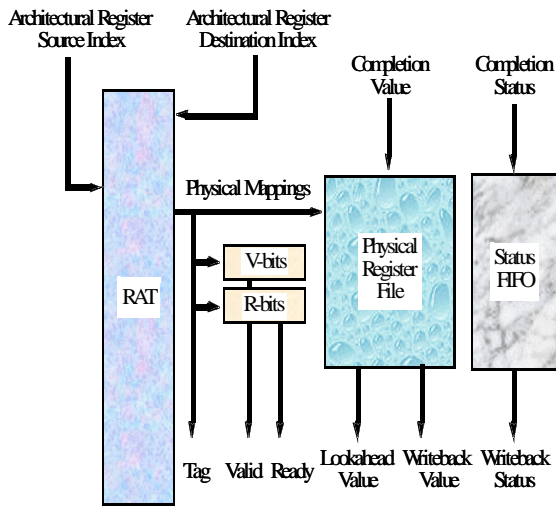


Figure 6. Reorder Buffer Organization

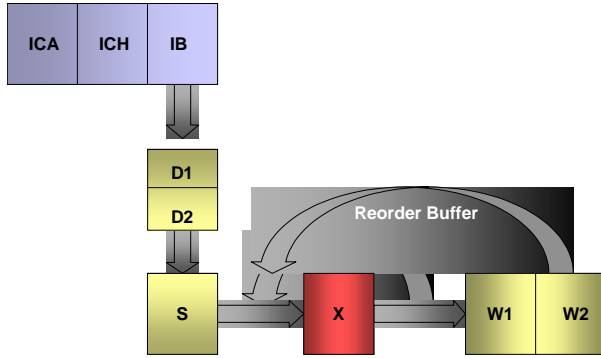


Figure 7. Pipeline Stages

which the decoded instructions are selected for issue by the finite state machine.

S is the schedule stage where register file and reorder buffers are accessed and instructions and operands are routed to the reservation stations of the appropriate functional unit. X is a single-cycle execute stage. W1 or Writeback 1, is the stage where results of dependent instructions are forwarded to the reservation stations and the reorder buffer is updated. W2 or Writeback 2, is when the architectural state or register file and control registers are updated. All of the stages are single cycle assuming no cache misses or branch mis-prediction.

4. Implementation and Testability

The FXU processor has been implemented in a 0.18 μm TSMC process in order to verify the design in a known reliable CMOS process. The standard ASIC flow was used for the implementation. The behavioral verilog was verified using the test set up described in section 5 and synthesized using Synopsys Design Compiler. The physical layout was done using Silicon Ensemble while Cadence Virtuoso was used for the design rule check and layout versus schematic check. Table 1 summarizes the modules that make up FXU. Artisan SRAM Compilers were used to generate the instruction and data cache. The register file was implemented using negative-edge-triggered flip-flops due to the lack of dual-write port memory compilers.

To enhance the observability and contrallability of internal modules and critical signals, nine scan chains have been incorporated in the design. A partial scan multiplexed flip-flop scheme was employed for scan chain insertion. Three scan chains are dedicated to the execution unit (one each for the load-store unit, branch resolution unit, and one that passes through the two arithmetic logic units).

Module	Width	Height	Gates
Branch Predictor	581.46	750.96	6976
Branch Target Buffer	574.86	574.56	5720
Decode	570.9	378	5784
Dispatch	499.62	498.96	1866
Dmaq	359.62	226.8	1535
Execution Core	1330.86	1169.28	31167
Fetch	415.14	468.72	3666
Imaq	450.78	282.32	2301
Reg File	1032.9	927.36	18563
Reorder Buffer	894.3	1002.96	15717
Reorder Buffer Fifo	772.86	624.96	9414
Stream	584.1	650.7	2464
Writeback	268.62	307.4	2096

Table 1 Dual Issue FXU Statistics

One scan chain each is devoted to the decode and dispatch unit of the processor. The register file, reorder buffer and the write back stage are tested using a single scan chain. One scan chain is used for the front-end of the processor (fetch, stream and branch predictor units). The remaining two scan chains pass through the IMAQ and DMAQ units.

5. Test Flow

The test flow is shown in Figure 8. The process begins by executing the PUMA_checker. The PUMA_checker can take a user program or randomly generate a program which it compiles to a memory image. This image is loaded into the memory spaces of both the RTL compiled program **puma** and the PowerPC simulator PSIM [3]. PUMA_checker then steps each machine through one instruction at a time starting at the base instruction zero. At some specified interval, one instruction being the default, it compares the registers and memory between the two simulators. If there is an error, the RTL model must be analyzed and fixed and the process repeated.

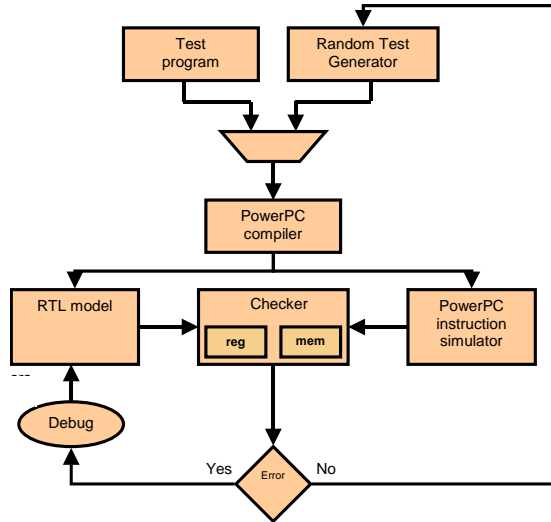


Figure 8. Verification Environment

This setup was used to verify the correctness of the behavioral verilog and the post synthesis netlist. The functionality of the FXU process was verified for over 10^9 instructions.

6. Conclusion

In this report, the design and implementation issues of a 4-way superscalar, out-of-order execution PowerPC FXU processor have been described. The PUMA architecture and the necessary modifications employed to convert the existing single issue machine into a dual issue one have been mentioned. Several changes had to be incorporated in the decode logic, schedule stage and the execution core in order to allow for dual issue. The cache sizes were increased to reduce the cache miss ratio and thereby improve the instructions per cycle. The design is intended to be used as a test vehicle for the analysis of circuit design techniques in SOI and as a soft IP core for further projects.

7. References

- [1] "CGaAs PowerPC FXU", Alan J. Drake et. al, Design Automation Conference (37th DAC), June 2000.
- [2] "A Micro architecture for Resource-Limited Superscalar Microprocessors", Todd D. Basso, Tech. Report N. SSEL-289, Solid State Electronics Laboratory, EECS Department, University of Michigan, 1999
- [3] Motorola Inc., *PowerPC Microprocessor Family: The Programming Environments*, 1994
- [4] A. Cagney, PSIM: PowerPC Simulator, <ftp://ftp.ci.com.au/pub.psim>, 1994