Timing Analysis of Digital Systems with Gated Clocks

by

David Van Campenhout and Trevor Mudge

CSE-TR-257-95
August 1995

Timing Analysis of Digital Systems with Gated Clocks

David Van Campenhout and Trevor Mudge

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109-2122

August 1995

Abstract

The SMO model for analyzing the temporal behavior of general synchronous systems has proven to be very effective. This paper extends the model to systems in which the clock signals can be gated (conditional). We describe a method for verifying the timing of such systems. Our approach hinges on the use of information about design intent in the analysis to obtain good accuracy. Ingredients for a timing verification tool that allows designers to take advantage of gated clocks are developed. The system is partitioned into datapath and control sections. The datapath is analyzed using the conditional delay model. Gated clocks allow multi-cycle behavior. Symbolic finite state machine traversal techniques are employed for checking the validity of multi-cycle paths.

1 Introduction

1.1 Static timing analysis

There are two main approaches for verifying the temporal behavior of circuits: 1) simulation-based timing analysis, and 2) static timing analysis. Approaches based on simulation, often termed dynamic timing analysis, have the advantage of being able to handle a wider variety of circuits. They can also take into account the full functional behavior of the circuit. Their main disadvantage is the incompleteness of the analysis. To obtain a reasonable coverage of the complete circuit behavior, a very large number of simulations are necessary. Static timing analyzers try to overcome this limitation. Static timing analyzers abstract some details of the temporal behavior of a circuit. This allows them to consider all behaviors of the circuit at once. In practice, a superset of the real circuit behavior is considered, resulting in a conservative (pessimistic) analysis. For example, false paths are usually not accounted for. Static timing analyzers are also more restrictive in the variety of circuits that can be analyzed. Early static timing analysers were limited to fully synchronous circuits without any clock skew.

A comprehensive model for analyzing synchronous systems was proposed by Sakallah, Mudge and Okulotun [1]. The model has been extended to handle clock skew. But, the model assumes that all synchronizers are clocked by a multiphase clocking system in which all clocks have the same periodicity. This paper extends this model to handle gated clocks.

1.2 SMO model

The SMO model assumes a multiphase clocking system. All phases have a common clock period. During that clock period, each phase makes exactly one rising and one falling transition. The circuit is modeled by a network of synchronizers (either level-sensitive latches, or edge-triggered flip-flops). The synchronizers are connected by arcs corresponding to the combinational logic. The arcs are labeled with the minimum and maximum delay through the logic. The synchronizers are characterized by their setup and hold times, their clock phase, and their clock skew. For clock schedule verification, the variables in the SMO model are the minimum and maximum arrival time of signals at the input of the synchronizers, and the minimum and maximum departure time of signals at the output of the synchronizers. For clock schedule optimization, the clock period and the event times of the rising and falling transition of each phase are variable as well. The relationship between these variables combined with the setup and hold constraints of the latches constitute a set of constraints that can be relaxed to linear constraints. The verification problem can be solved by constraint relaxation. The optimization problem was originally tackled by linear programming. However, Szymanski [2] proposed a more efficient approach. First the minimum cycle time set by the loops is computed. Subsequently all relevant constraints with respect to that cycle time are generated. The resulting linear program is much smaller in size.

1.3 Gated clocks

In some systems the signals clocking the synchronizers are not directly generated by a central source. Derived clocks (gated clocks or secondary clocks) are formed by combining a primary clock signal with a local control signal. Typically the primary clock is AND-ed with the control signal. Hence the clock can be selectively disabled. Advantages for this design approach include the ability to save power, the flexibility to implement some logic efficiently and also the ability to create multi-cycle paths. Signals propagating along multi-cycle paths are allowed several clock cycles, before the result is captured in the output latch along the path. The presence of such paths would be normally reported as violations in the SMO model. In this report we propose a method to deal correctly with gated clocks and multi-cycle paths.

2 Previous work

Ishii [3] describes methods for retiming systems that contain precharged circuit structures and/or gated clock signals. He derives a set of constraints which all take the form of inequalities between the delay of a path and the length of a time interval which is bounded by a pair of clock edges. He notes that these inequatities are not significantly different from those obtained during the analysis of regular purely synchronous systems. This analysis does not consider multi-cycle paths, and circuits exhibiting multi-cycle paths do not comply with Ishii's definition of proper operation. No functional analysis is performed.

Kawarabayashi et al. [4] consider CPU-like circuits which are partitioned into a datapath and a controller. A rather restrictive datapath model is used. The combinational logic between the latches is partitioned in subblocks interconnected by multiplexors. These multiplexors, as well as the signals gating the clocks are steered by the controller. The controller does not contain any gated clocks. A state transition graph of the controller is extracted directly from the netlist. The state transition graph is traced to determine the number of clock cycles available for datapath signals to propagate along paths between two consecutive latches.

Gupta et al. [5] consider the problem of multi-cycle paths in microprocessor-based designs. They construct a composite state-transition graph (CSTG). The circuit state is defined by output signal values on each synchronizer at any time. This definition of state allows them to handle very general circuit behaviors, e.g. no predefined clock schedule is required. Each timing check, which specifies a minimum separation between two events, is verified by tracing the CSTG (tracing through time) and tracing the network (tracing through space). To benefit from the incremental nature of the design process, the constraints are derived symbolically. The symbolic delay parameters are substituted for numerical values at the end. The approach is limited to circuits with a relatively small CSTG.

Our work begins with the assumption that high-level design information is available. High-level design knowledge is used to partition the circuit into a datapath and a controller section. Combinational datapath sections are analyzed using an approach similar to [8]. Our datapath model is more general than the multiplexer-based datapath model used in [4]. The controller is transformed into a classical synchronous machine without any gated clocks. Symbolic state traversal techniques, following the approach of [12], are used to verify the conditions governing the validity of multi-cycle paths.

This paper is organized as follows. Section 3 motivates the use of gated clocks, discusses some topologies using gated clocks, and states conditions for proper operation of gated clocks. Our analysis methodology is discussed in the next section. These techniques are illustrated with an example in Section 5. Concluding remarks follow in the last section.

3 Gated Clocks

3.1 Motivations for Designing with Gated Clocks

Gated clocks can be used to reduce power consumption. In CMOS circuits, the static power consumption is generally negligible compared to the dynamic power consumption: most of the power dissipation is due to the switching of circuit nodes. In synchronous systems, even though the circuit is idle, the clock signals are always switching, causing significant power dissipation. This can be reduced by blocking the clock signals before they reach the synchronizers during those times that the circuit is idle. This can be achieved by gating the clock signals, i.e., the primary clocks are AND-ed with control signals, thus forming secondary clocks, which steer the synchronizers.

A second motivation for using gated clocks is the creation of multi-cycle combinational operation. Under multi-cycle operation, combinational blocks are given a time budget of several clock cycles to produce results. This is achieved by holding the synchronizers which constitute the inputs of the combinational block. A multi-cycle operation mode may be the preferred design style for complex functional units which are used infrequently. Due to the infrequent use, there is no point in pipelining the unit. Not only do pipeline latches consume a non-negligible amount of real estate, but the setup and hold time of these latches and the propagation delay through them may even increase the latency of the unit. In a non-pipelined approach without gated clocks, the complexity of the unit may result in the unit setting the maximum clock frequency. Clearly, multi-cycle operation provides a solution for such cases. One may note that wave-pipelining can be employed as well. However, in that case, the combinational block has to be designed with special care, so as to avoid collisions between separate waves. In this paper, we will not consider wave-pipelined structures in our analysis.

Finally, gated clocks can lead to a more compact physical design. The use of gated clocks allow certain functions to be implemented in a very area-efficient manner. A 32-bit register which has to retain its content for some cycles, and clock in new values in other cycles can be implemented with a multiplexer which selects either the new value or the present value. The same funcitonality can be accomplished by gating the clock of the register with the same control signal.

Many designers avoid gated clocks in their designs for two important reasons. First, gating the clocks introduce extra skew between the signals clocking the synchronizers. In high frequency designs, the skew may pose an insurmountable problem. In any case, correct operation of the circuit has to be verified carefully. Second, present static timing verification tools are not able to deal with designs containing gated clocks properly. Therefore designers often have to rely on simulations. Although this approach offers the greatest flexibility as to the variety and detail of circuits which can be treated, it suffers from the major drawback of being incomplete. Simulation based approaches are not able to exhaustively verify the complete circuit behavior for circuits of any non-trivial size.

One of the goals of this work is to allow safe design of systems with gated clocks, so that the advantages of gated clocks can be exploited.

3.2 Circuit Structures and Regimes of Operation Using Gated Clocks

Consider a section of a circuit comprising a latch feeding a block of combinational logic and a latch being fed by that combinational logic. Depending on the characteristics of the clock signals steering the input and the output latch, four regimes of operation can be distinguished.

3.2.1 Input latch gated, output latch gated

This type of structure is typically used to achieve multi-cycle behavior. The input latches are gated in order to hold the values of these latches for a certain number of clock cycles. The output latches are gated as well. The relationship between the two gating functions is such that the output latch samples its inputs only after the input latch has been holding its value for a certain number of clock cycles. Consequently, the combinational logic between these latches has multiple clock cycles to propagate signals.

3.2.2 Input latch gated, output latch not gated

Although the output latch operates on every clock, multi-cycle operation is still possible in this type of circuit. In contrast to the previous case, the output synchronizer will sample the output of the combinational logic even when the combinational logic is not ready with its computation. Thus, meaningless values will be latched in the output latch. In the overall design, the values held by the output latches are don't cares in the cycles immediately following the latching of the input latches. Conventional timing verifiers would signal timing violations. This illustrates the necessity to incorporate high level information in timing verification.

3.2.3 Input latch not gated, output latch gated

In this configuration, the purpose of the gated clock of the output latch is to provide a cheap way of squashing unwanted or erroneous data.

3.2.4 Input latch not gated, output latch not gated

This is the classical topology in purely synchronous systems. Multi-cycle behavior is still possible, if wave-pipelining is allowed, but requires very careful design of the combinational blocks between the latches.

Note that the configuration where both the input and the output latches have their clocks gated is the most general, as all four modes of operation can be realized by this configuration.

3.3 Secondary Clock Generation: Conditions for Proper Operation

In this section we define what we mean by well-formed secondary clocks. Primary clocks enable latches during one interval of the clock cycle, and disable these latches during the complementary interval. For secondary clocks we require that during a clock cycle the secondary clock must either disable the latch for the whole clock cycle, or it must enable the latch during one interval of the cycle and disable the latch during the complementary interval. Moreover, we demand that all events on a secondary clock are triggered by events on a single primary clock. This definition implies that a secondary clock signal must be free of glitches. This can be assured by 1) allowing only a single path from that secondary clock to the primary clock that it is derived from, and 2) requiring that the side-inputs on that single path are stable between the enabling trigger event of the primary clock, and the next disabling event of the primary clock. It also follows from the definition that for a positive (negative) level sensitive latch clocked by a secondary clock, the secondary clock signal must be functionally an AND (OR) of some control signals and either a primary clock, or the inverse of a primary clock. The actual implementation of the network generating the secondary clock may not look like an AND (OR) gate, but the functional property can be easily checked as follows. Let $\psi = f(\phi, \vec{q})$ be a secondary clock derived from the primary clock ϕ, where $\vec{q}$ is a vector of state variables and primary inputs. Let f_{ϕ} and $f_{\bar{\phi}}$ be the cofactors of ψ with respect to ϕ and $\bar{\phi}$ respectively. Then, for a positive (negative) level sensitive latch clocked by ψ, either $f_{\phi} = 0$

($\bar{f}_{\phi} = 0$), or $f_{\bar{\phi}} = 0$ ($\bar{f}_{\bar{\phi}} = 0$). In [4] the same functional constraint is imposed on secondary clocks, but the dual for negative level sensitive latches was omitted. Moreover, that paper doesn't mention the topological constraint that there be only a single path from the secondary clock to the primary clock. Without this constraint, it is very hard to verify that the secondary clock is indeed glitch-free.

4 Analysis Tools and Methodology

4.1 Datapath Model

In classical static timing analysis, the circuit is abstracted to a directed graph. The nodes of this graph correspond to the synchronizers in the circuit, and the arcs represent combinational logic connecting two corresponding synchronizers. The arcs are labeled with the minimum and maximum delay between the synchronizers corresponding to the starting point and end points of the arc. These delays can be obtained in various ways. A simple topological analysis can be used to provide the delays. Alternatively, paths can be verified using the floating-mode sensitization criterion.

Circuits exhibiting multi-cycle operation require a more sophisticated model for performing static timing analysis. In case different functional units with different delays share the same input and output synchronizers, it is necessary to differentiate among them. The activation of the functional units is orchestrated by the controller. High level information is used to indicate the appropriate control signals. The delay through the combinational block is expressed as a function of these control signals. This approach is very similar to the hierarchical timing analyzer described in [8]. For each pair of connected synchronizers, two sets of tuples are computed. Each tuple in the first set gives the maximum delay from the first synchronizer to the other, and under which settings of the control signals that maximum delay is exhibited. Similarly, the second set contains minimum delays. This model is more flexible and can handle a wider variety of circuits than the simple mux-model described in [4].

4.2 Transformations on the Controller: The Mux Transform

For functional analysis, as described in the next section, it is necessary to extract the finite state machine underlying the circuit. The presence of gated clocks complicates that extraction, but typically the gated clocks that occur in the controller are not used to establish multi-cycle paths. In such cases the circuit can be transformed into a circuit without gated clocks which is equivalent as far as timing and functionality is concerned. This so-called mux transform is illustrated in Figure 1. After the transformation, the extraction of the underlying state machine is trivial. One may note that this mux transform could be applied to the complete circuit (both datapath and controller). Subsequently, a classic static timing analyzer could be used to verify the timing. However, in that case such a timing analyzer would report multi-cycle paths as violations, or alternatively the analyzer would have to remove sequential false paths from the circuit. The first alternative is undesirable whereas the other option is not viable for non-trivial circuits.

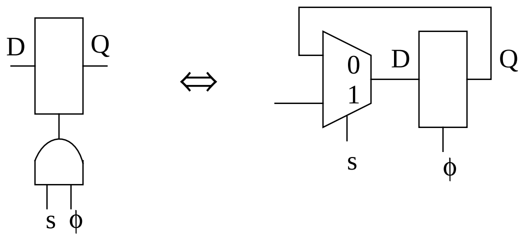


Figure 1: Mux transform

.

4.3 Symbolic State Transition Graph Traversal

Many problems pertaining to sequential machines involve traversing the state transition graph of a finite state machine. For example, in formal verification, temporal properties of sequential machines are checked [11]. In another example, some approaches to test pattern generation for sequential machines [10] rely on the computation of the set of all reachable states from the reset state. In early work, the state transition graph of the machine was built explicitly. However, as the number of synchronizers increases, the size of the state transition graph increases drastically. State traversal techniques which are basically enumerating all states in the graph are therefore not practical for non-trivial machines. Implicit traversal methods [12] exploit the fact that for the above mentioned applications, states exhibiting a certain property do not need to be treated individually but rather as a group. The key idea behind these methods is to represent a subset of the n-dimensional boolean space by a boolean function. This so-called characteristic function indicates for every point of the n-dimensional space whether it belongs to the subset, or not. The manipulation of subsets can then be reduced to the manipulation of boolean functions. For instance, the characteristic function of the intersection of two subsets is given by the logical AND of the characteristic functions corresponding to those two subsets. Using binary decision diagrams [6] to represent the boolean functions, these operations can be performed fairly efficiently.

For the purpose of this work, we compute the set of all reachable states in the state transition graph corresponding to a certain finite state machine. A generic algorithm is shown in Figure 2. The next-state functions *f* are constructed from the netlist. The computation of the set of all reachable states reduces to a fixed point calculation of an iterative process. During each iteration, the set of currently reachable states is augmented with the states reachable from the states which where newly discovered in the previous iteration. The process is repeated until no more new states are discovered. The `image` operator gives the set of next-states given a transition function and a set of states.

```
S = {reset state}
newStates = S
while (newStates) {
      NS = image( f, newStates)
      newStates = NS - S;
      S = S U NS;
      }
```

Figure 2: Computation of reachable states

4.4 Methodology

1. *Specification of the circuit.* The specification of the circuit includes: a netlist, and input-output delays annotating each gate. Latches are characterized by their setup and hold time. Furthermore, additional high-level information is included: 1) a partition of the latches into either datapath or control, 2) identification of control signals of interest.
2. *Abstraction of the circuit.* This step involves computing the minimum and maximum combinational delays between a pair of latches. In case the drain latch depends on a control signal of interest, the condition delay model is applied as indicated in Section 4.1. In case the combinational delay between a pair of latches exceeds the proposed clock cycle, multi-cycle operation is assumed, and the minimum cycle budget is inferred.
3. *Analysis of the network generating the gated clocks*. The functional and topological requirements for gated clocks are checked as described in Section 3.3. Constraints originating from the side-inputs along the path between a secondary clock and the primary clock it is derived from are generated.
4. *FSM analysis*. A reachability analysis on the finite state machines of interest is performed. First gated clocks are removed from the machine using the mux transform. The next-state functions are constructed. This is followed by symbolic state transition graph traversal.
5. *Verification of the properties governing multi-cycle behavior.* The conditions on the control signals defining the gated clocks, and other control signals are verified. These conditions guarantee that there are indeed as many clock cycles available for multi-cycle operation as inferred in step 2.
6. *Static timing verification using constraint relaxation*.The SMO constraints are adjusted for multi-cycle operation with the cycle budget verified in step 5, and the SMO constraints are augmented with the constraints generated in step 3. The system of constraints is solved using constraint relaxation.

5 Example

In this section the methodology outlined above is illustrated with an example. The circuit under consideration is shown in Figure 3. For simplicity transport delays are assumed. Unlabeled combinational circuit elements have a unit delay. The datapath fragment contains two functional units with very different delays. The longest delay through the multiplier is 23 time units. The adder executes in 4 time units. Both functional units share the same destination latch through a multiplexer. Assuming multiplication is relatively infrequent, it might be appropriate not to pipeline that unit, thus saving the cost of the extra latches. Instead, the multiplier is operated in a multi-cycle mode. Whenever a multiplication is needed, the data input latches are not clocked with new values, until the present data has had time to propagate through the multiplier. Similarly, the destination register is not clocked until the result is stable. If the destination register were to be clocked unconditionally, it is likely that it would contain unpredictable values in some cycles.

The select signal of the multiplexer and the signals gating the clocks are generated by a small finite state machine. This controller ensures that when the multiplier is selected, the input data latches do not change for two cycles such that the signals have time to propagate through the multiplier. The state machine has one primary input, which indicates whether the next operation to be executed is a multiplication. The state diagram of the controller is shown in Figure 4. Note that in our method we never build the state diagram explicitly.

We now apply our methodology to the circuit.

1. *Specification of the circuit.* The latches are partitioned into datapath and control latches. Latches $\{LA, LB, LZ\}$ belong to the datapath partition; latches $\{LQ11, LQ12, LQ21, LQ22\}$ are part of the control partition. The signal M is marked as a control signal of interest.
2. *Abstraction of the circuit.* Using a delay calculator based on conditional arithmetic [8], the minimum and maximum delays through the combinational blocks are computed. The delays are expressed as a function of the control signals of interest, i.e. M. Note that when $M = 1$, the longest path through the combinational logic is 24, whereas it is only 5 when $M = 0$. Hence multi-cycle operation can be inferred. Since the clock cycle is 10 time units, at least 2 extra cycles are required. The abstraction of the datapath using the conditional delay model is shown in Figure 5. For clarity, only maximum delays are indicated.
3. *Analysis of the network generating the gated clocks*. A functional and a topological analysis of the combinational circuits generating the gated clocks is performed. In the example, those networks are single AND-gates. They do satisfy the criteria for well-formed gated clocks. Also, a temporal analysis is performed. Constraints on the arrival times of the side-inputs are generated: S has to be stable from the rising edge of ϕ_2 until the next falling edge of ϕ_2, and L, has to be stable from the rising edge of ϕ_1 until the falling edge of ϕ_1.
4. *FSM analysis.* Any gated clocks present in the controller are removed by performing the mux-transform. Subsequently, the next state functions of the FSM of the controller are extracted. The set of reachable states is computed. Expressions of the gated clocks and the control signals *(M, L, S),* as a function of the state variables of the finite state machine are constructed.

   Next state functions:

   $$Q_1 = \overline{q_1} \cdot \overline{q_2} + q_1 \cdot \overline{OP}$$
   $$Q_2 = q_1$$

   Output functions:

   $$L = q_2 @ 0.5$$
   $$M = \overline{q_2}$$
   $$S = q_1$$

   where $@\,0.5$ indicates that the signal is shifted over one phase to the future with respect to the frame of reference.
5. *Verification of the properties governing multi-cycle behavior.* The property that needs verification is that when the multiplier is selected, the input data latches do not change for two cycles such that the signals have time to propagate through the multiplier. Formally this can be stated as:

   $$\forall s \in \Re, \forall\ t \in \Re : \exists\ OP \in \{0, 1\} \text{ such that } Q(t, OP) = s : \quad M(s) \cdot S(s) \Rightarrow \overline{L(s)} \cdot \overline{L(t)}$$

   where $\Re$ is the set of all reachable states.

   Proof: Let $s = (s_1, s_2)$, and $t = (t_1, t_2)$

   Using the next state equations:

   $$\exists\ OP \in \{0, 1\} : s_1 = \overline{t_1} \cdot \overline{t_2} + t_1 \cdot \overline{OP}, \text{ and } s_2 = t_1$$

   Now, expressing both sides of the implication as a function of t:

   $$M(s) \cdot S(s) = \overline{t_1} \cdot (\overline{t_1} \cdot \overline{t_2} + t_1 \cdot \overline{OP}) \text{ , and}$$
   $$\overline{L(s)} = \overline{t_1} \text{ , and } \overline{L(t)} = \overline{t_2}$$

Thus: $\forall\ OP \in \{0, 1\}\ ,\forall\ t \in \Re :\ \overline{t_1} \cdot (\overline{t_1} \cdot \overline{t_2} + t_1 \cdot \overline{OP}) \Rightarrow \overline{t_1} \cdot \overline{t_2}$ ❑

All these expression are computed using OBDDs which allows easy application of the quantification operators. Now we have verified that at least two cycles are available. Consequently the paths in the graph can be annotated as multi-cycle paths, taking two cycles. The abstraction of the complete circuit for static timing analysis is shown in Figure 6. Note that arcs corresponding to multi-cycle paths are labeled with a # followed by the number of cycles available.

6. *Verification using constraint relaxation.* Constraints corresponding to the network shown in Figure 6 are generated, and the network is verified using constraint relaxation [7].

6 Conclusion

We proposed a methodology for verifying the temporal behavior of synchronous circuits containing gated clocks. The circuit is partitioned in a datapath and controller sections. The combinational logic in the datapath is abstracted using the conditional delay model and high level design information about control signals of interest. Secondary clocks are identified and a well-formedness property is checked. When the combinational delay connecting two latches exceeds the given clock cycle, that path is assumed to be a multi-cycle path. This hypothesis is checked by examining relationships between the (secondary) clocks enabling the latch pair. State transition graph traversal techniques are employed on a restricted finite state machine. The multi-cycle information is incorporated in the subsequent static timing analysis.

References

[1] K. Sakallah, T. Mudge, O. Olukotun, “checkTc and minTc: Timing verification and optimal clocking of synchronous digital circuits,” in *ICCAD-90 Digest of Technical Papers*, pp. 552-555, Nov. 1990.

[2] T. Szymanski, “Computing optimal clock schedules,” in *Proc. of the 29th Design Automation Conference,* pp. 399-404, 1992.

[3] A. Ishii, “Retiming gated-clocks and precharged circuit structures,” in *ICCAD-93 Digest of Technical Papers,* pp. 300-307, 1993

[4] M. Kawarabayashi, N. Shenoy and A. Sangiovanni-Vincentelli, “A verification technique for gated clock,” in *Proc. of the 30th Design Automation Conference,* pp. 123-127, 1993.

[5] A. Gupta and D. Siewiorek, “Automated multi-cycle symbolic timing verification of microprocessor-based designs,” in *Proc. of the 31st Design Automation Conference,* pp. 113-119, 1994.

[6] R. Bryant, “Graph-based Algorithms for Boolean Function Manipulation,” *IEEE Trans. Computer*, Vol. C-35, No. 8, August 1986, pp. 677-691.

[7] T. Burks, K. Sakallah and T. Mudge, “Critical paths in circuits with level-sensitive latches,” *IEEE Trans. VLSI Systems*, Vol. 3, No. 2, pp. 273-291, June 1995.

[8] H. Yalcin, J. Hayes, “Hierarchical timing analysis using conditional delays,” in *ICCAD-95 Digest of Technical Papers,* to appear, 1995.

[9] P. McGeer, R. Brayton, *Integrating Functional and Temporal Domains in Logic Design*, Kluwer Academic, 1991.

[10] A. Ghosh, *Sequential Logic Testing and Verification*, Kluwer Academic 1992.

[11] J. Burch, et al., “Sequential circuit verification using symbolic model checking,” in *Proc. of the 27th Design Automation Conference*, pp. 46-51, 1990.

[12] H. Touati, et al., “Implicit state enumeration of finite state machines using BDD’s,” in *ICCAD-90 Digest of Technical Papers*, pp. 130-133, 1990.

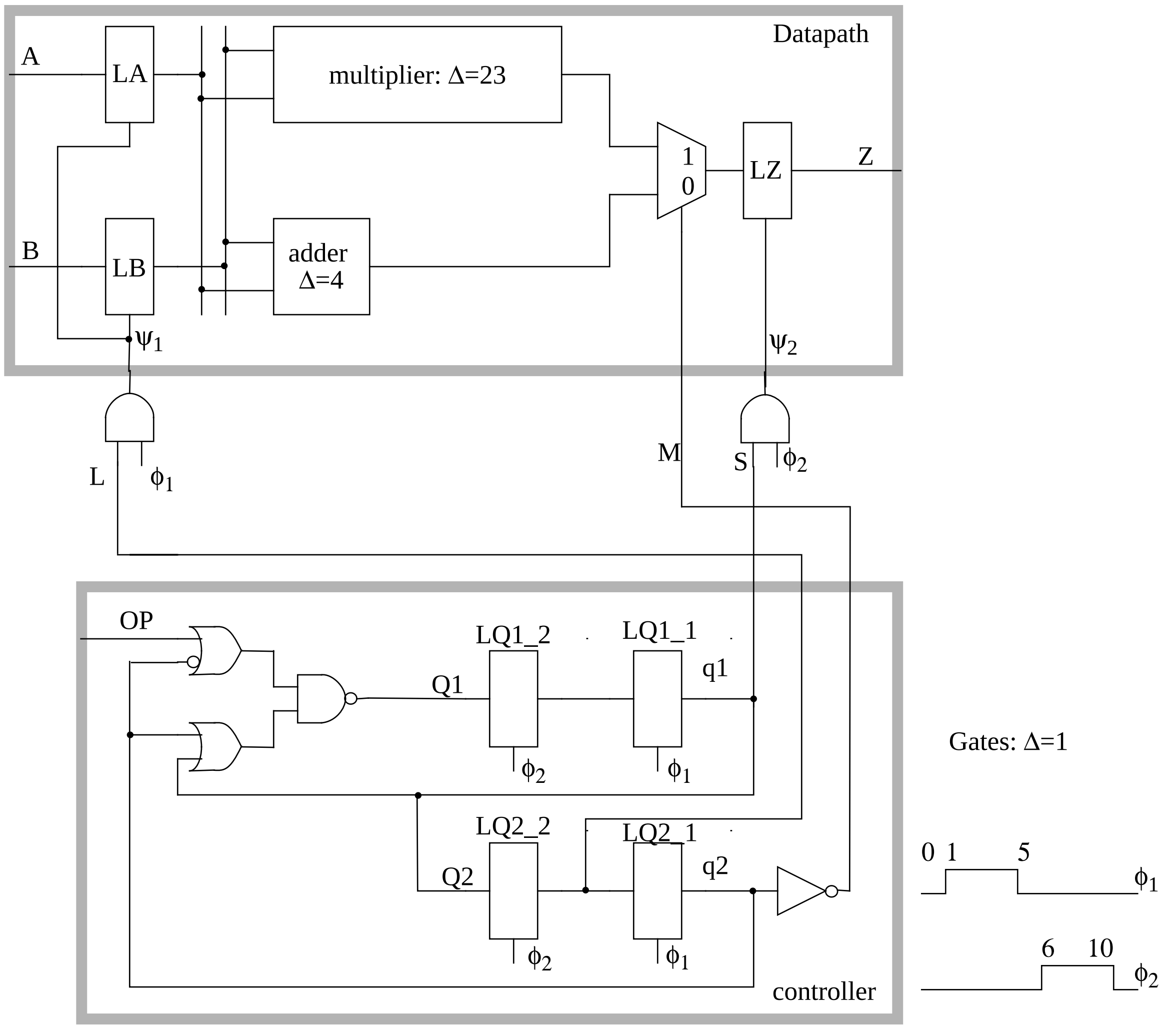
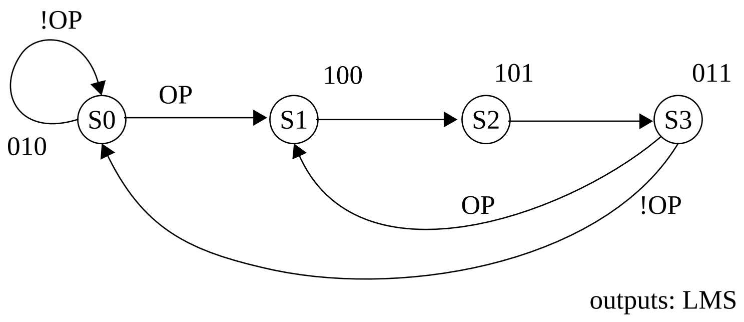


Figure 3: Example circuit



State encoding

State	q_1 q_2
S0	11
S1	01
S2	00
S3	10

Figure 4: State diagram

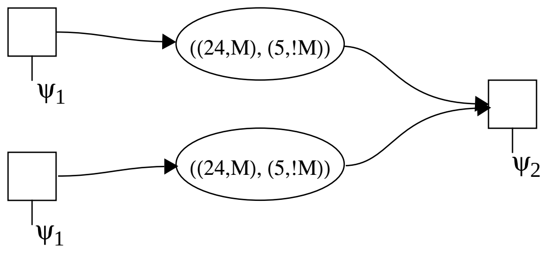


Figure 5: Conditional delay model of datapath

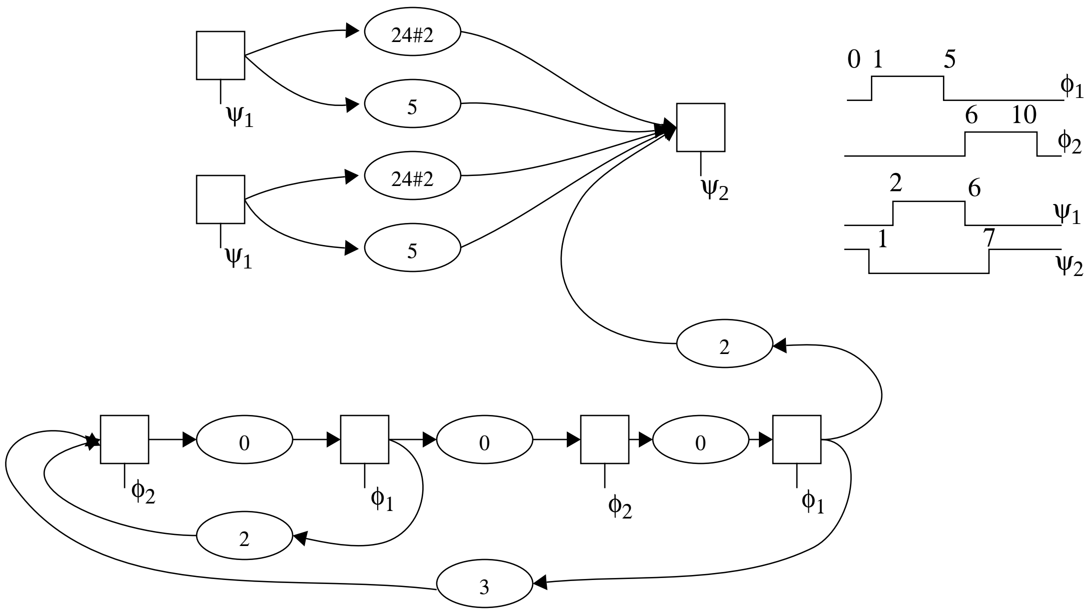


Figure 6: Bubble diagram