Automatic Clock Abstraction from Sequential Circuits

Samir Jain
Digital Equipment Corporation
Hudson, MA 01749

Randal E. Bryant
Carnegie Mellon University
Pittsburgh, PA 15213

Alok Jain
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Our goal is to transform a low-level circuit design into a more abstract representation. A pre-existing tool, Tranalyze [4], takes a switch-level circuit and generates a functionally equivalent gate-level representation. This work focuses on taking that gate-level sequential circuit and performing a temporal analysis which abstracts the clocks from the circuit. The analysis generates a cycle-level gate model with the detailed timing abstracted from the original circuit. Unlike other possible approaches, our analysis does not require the user to identify state elements or give the timings of internal state signals. The temporal analysis process has applications in simulation, formal verification, and reverse engineering of existing circuits. Experimental results show a 40%-70% reduction in the size of the circuit and a 3X-150X speedup in simulation time.

1 Introduction

As the digital design industry continues to grow, so does the importance of tools to aid in the analysis of large designs. Specifically, there exists a need for analysis tools that transform detailed circuit models into more abstract models. A more abstract circuit design may provide advantages in the areas of simulation, formal hardware verification, and reverse engineering of existing circuits. As designs grow larger, it becomes impractical to simulate an entire low-level design. Transformations are performed to remove details from the circuit design, thus producing a smaller, more abstract circuit and improving the performance of a simulator. Our formal verification strategy [2] attempts to bridge the wide gap between a detailed circuit and the abstract specification. Abstraction of the circuit model enables us to reduce the gap between the circuit implementation and the abstract specification. Finally, the abstracted model can be used to reverse engineer the functionality of some pre-existing circuit designs.

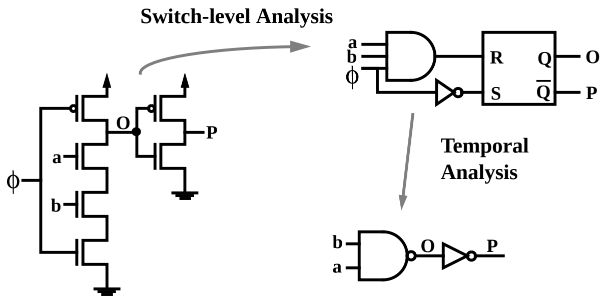

Figure 1 Domino Example

This research partially funded by Semiconductor Research Corporation contract number 94-DC-068.

A tool called Tranalyze [4] has been created that transforms a switch-level circuit into a functionally equivalent gate-level representation. Tranalyze is able to capture aspects of switch-level circuits such as bi-directional transistors, precharged logic, stored charge, and multiple signal strengths. As an example, consider the switch-level domino circuit shown in Figure 1.

The abstraction process of Tranalyze can be broken down into a few steps. The first step performs a switch-level analysis and generates a low-level gate circuit. For this domino circuit, the switch-level analysis generates a functionally correct but very complicated gate-level representation. The RS circuit in Figure 1 represents most aspects of the circuit generated by the switch-level analysis. The behavior of the circuit is similar to an RS latch with the set activated on a low clock and the reset on the **`and`** of the clock and data inputs. Note the complexity of this representation.

The normal operation of domino circuits is as follow: when ϕ is low the circuit is precharging and net **O** is low. When ϕ is high, the circuit conditionally discharges, and net **O** is essentially the **`nand`** of nets **a** and **b**. Net **P** is easily recognized to be the **`invert`** of net **O**. The circuit details of the clocking have been carefully included in the gate network, making it a complex representation.

It is possible to obtain a much simpler representation if the user specifies the clocking behavior and input/output timing for the circuit. For example, given the RS circuit in Figure 1 and details about the timing information for the circuit, our temporal analysis tool can extract out a much simpler circuit, as shown with the 2-input **`nand`** in Figure 1. This model is very simple to read since the clock has been abstracted out. By taking advantage of the temporal information specified by the user, a simple static logic gate is produced.

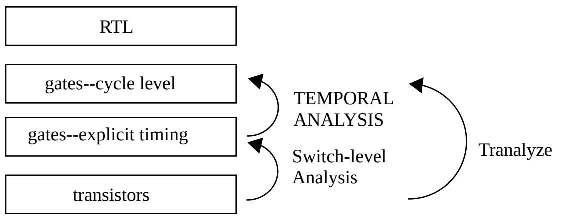

Figure 2 Abstraction process in circuit design

Like the example above, many sequential circuits may have complicated clocking and timing patterns, making them difficult to understand. If the clocking methodology is known, the clocks can be abstracted through a “temporal analysis” that would generate a higher-level circuit. Figure 2 describes this clock abstraction process. The lowest level is a transistor-level circuit. After a switch-level analysis has been performed, a functionally equivalent gate-level representation is obtained. This circuit has detailed timing information built into it, and thus a temporal analysis is performed that abstracts out all of the timing information from the circuit. This temporal analysis stage will be the focus of this paper. Unlike other possible approaches, our approach does not require the user to manually identify state nodes. In addition, our tool automatically identifies internal state nodes and performs the temporal analysis based on these nodes.

1.1 Previous Work

Most of the previous work done in the area of clock abstraction is based on the idea of clock suppression. This method attempts to reduce the activity in networks by suppressing the clocks during simulation. The limitation of clock suppression is that it must be implemented as algorithms in special purpose simulators. The original concept of clock suppression was devised by Ulrich [9]. The suggested method is to temporarily disconnect the sequential circuit from the clock source and reconnect it when a data input is received. Both Weber [10] and Takamine [8] introduce new signal values that represent periodic signals. Weber's model introduces a periodic state with *signal wave information*, which encompasses the period, the rise times, and the fall times. Takamine introduces 4 separate states to describe periodic signals that are currently high or currently low, in both positive and negative logic. While both methods produce good results, neither guarantee a full clock suppression. Although the number of periodic signals has been reduced, there may still be some signals that cannot be suppressed by these methods and are thus evaluated during simulation. Another disadvantage of these methods is that with the introduction of new states, new truth tables must be developed for each gate primitive.

A new general approach is called "Static Clock Suppression" [7]. Razdan performs his analysis on a phase-level model. A *phase* is defined as a period of time when all clock inputs are held constant. Razdan only allows inputs to change at the beginning of a phase. Like other work done in this area, his method produces very impressive results in simulation. However, it has certain restrictions. First off, inputs can only be set at phase boundaries. Secondly, nets must stabilize before they are reported. Thus the user is unable to view nets during an oscillatory period.

Kam [5] generates a finite state machine from a transistor netlist given information relating to clock signals and clock modules. The method involves performing a fixed point computation of the steady state response. The FSM generated is described as a BDD. The size of the BDD could become very large, making it difficult to represent large, complex circuits. Another problem with this method is that it is unable to handle oscillating circuits. Finally, inputs can only be changed on phase boundaries, similar to the restrictions placed by the clock suppression techniques.

1.2 Our Approach

All of the previously mentioned methods deal with circuits at a phase-level, implying data inputs can only be set and outputs viewed at phase boundaries, i.e. when a clock is changed. We remove this limitation by working with a discrete time model. This allows for inputs to be changed and outputs to be sampled at arbitrary points in time. As opposed to previous approaches, our approach also allows input nets to take on multiple values in one phase. Similarly, any net can be sampled at multiple points in the same phase. In effect, these input and output nets are multiplexed into and out of the circuit over a period of time. Another limitation of earlier approaches is the lack of a way to deal with oscillating nets. Most of the previous approaches either could not deal with oscillating nets or merely set the net to be a logic X. We can display the true value to the user for a given discrete time.

All of the clock suppression approaches are implemented in a logic simulator. However, we will generate a new abstracted gate-level circuit that can be simulated using any gate-level simulator. A form of symbolic simulation is used to perform the temporal analysis.

The timing model used to describe sequential circuits is explained in section 2. Our temporal analysis algorithm is presented in section 3. Section 4 describes an example to illustrate the methodology used for our analysis. Results are presented in section 5, and section 6 offers concluding remarks.

2 Discrete Timing Model

2.1 Basic Model

It is worthwhile to define the terminology used to describe sequential circuits. Figure 3 shows the model for a sequential circuit. *Primary inputs* (PI), or external inputs, are made up of *data inputs* (DI) and *clocks* (ϕ). The *combinational* portion of the circuit (C) uses the primary inputs and *present states* (PS) to generate the *outputs* (O) and *next states* (NS). The states are held during a zero-delay evaluation of the circuit. The next states are updated to present states as each next state passes through a *unit delay* (δ). This delay represents the smallest increment of a time delay. Thus a state is held for one time unit. The switch-level analysis in Tranalyze generates a model like the one on the left in Figure 3.

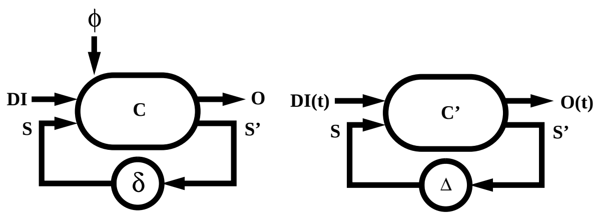

Figure 3 Sequential Circuit model, before and after abstraction

After performing a temporal analysis on the circuit, a model such as the one shown on the right in Figure 3 is obtained. In the new model, unit delays and clocks have been replaced by *cycle delay*s (Δ). States are now held for a full clock period, as opposed to a single time unit with a unit delay model. Note also that the combinational portion of the new circuit is not necessarily the same as the combinational portion in the earlier model. Extra logic may be added to make sure that all user-specified visible nets are present in the new circuit. Also, some logic may be deleted if none of the visible nets are dependent on this logic.

2.2 Timing Specifications

In order to temporally analyze a sequential circuit, the user must explicitly provide the temporal details of the circuit in a separate file. Unlike other work which used a phase-level timing model, our approach uses a discrete timing model. Under our timing scheme, there exists little difference between clocks and data inputs. Clocks must be set to a constant logic value, while data input values may be represented by symbolic variables. Variable names must explicitly be given by the user for every interval in which an input may take on a different value. The user must specify the output nets or visible nets in the circuit and the discrete points in time which the user wishes to sample each output net. Note that whereas data inputs and clocks are specified over an interval of time, outputs are sampled at discrete points in time. This allows for nets that oscillate over time to be reported.

2.3 Detailed Model

The temporal analysis is performed on a sequential circuit over one complete cycle. Any time a data input or clock changes, the circuit must be re-analyzed. After the circuit has been analyzed for a complete cycle, the individual results obtained are combined. Figure 4 shows a basic diagram of how this is accomplished. The shaded box in Figure 4 represents the combinational portion of the circuit on the right in Figure 3. The data input, DI(t), and output, O(t), have now been replaced by individual vectors for each discrete time.

Since primary inputs are set over ranges of time, every input change introduces a new input vector and a new combinational circuit. Associated with each new combinational circuit is a new output vector, as indicated in Figure 4. Note that while the inputs are set over ranges of time, the outputs are sampled at discrete points in time in the range [0, Δ-1], with Δ representing the clock period.

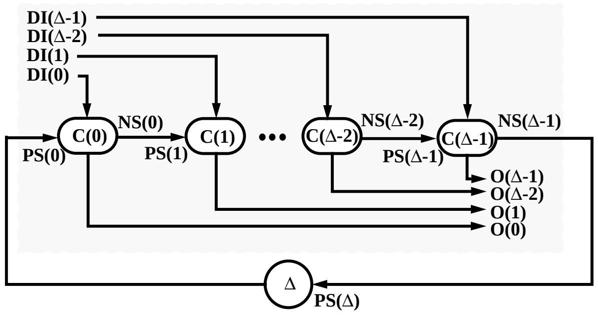

Figure 4 Detailed Model of a Temporally Analyzed Circuit

In a unit delay model, next states are updated to present states by traversing through unit delays. Thus the value for a next state net at time *t* is equivalent to the corresponding present state at time *t+1*, as shown in Figure 4. For instance, nets NS(0) and PS(1) can be represented by the same net in the gate-level description. The same is true for nets NS(Δ-1) and PS(Δ). Since we are only representing values in the range [0, Δ-1], the value of net PS(Δ) is the same as the value of net PS(0), separated by a delay of Δ. The net PS(Δ) effectively represents the initial value of the present state for the next cycle. Heuristics are used that allow a minimal amount of circuitry at each step.

By generating new circuits until the nets have stabilized, we are able to handle transient and even oscillating circuits. Figure 5 shows the analysis for a circuit that incurs a change in a primary input change at time t_0.

Assume that the circuit stabilizes at time t_s and the next primary input changes at time t. If $t > t_s$, the circuit is stable and the analysis uses net values from the C(t_s) circuit. If $t \le t_s$, the circuit has not stabilized. However, the analysis can still proceed, using the values from the C(t-1) circuit. In fact, this can be generalized to handle oscillating nets. An oscillating circuit is represented when t_s is infinity. Inputs can still be changed without the circuit first stabilizing.

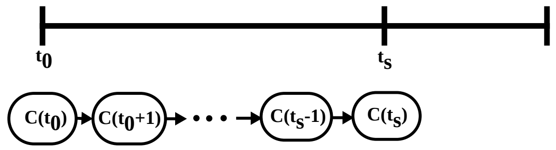

Figure 5 Transient Circuit

3 Temporal Analysis Algorithm

Once all of the data input and present state variables have been introduced, the temporal analysis algorithm is invoked. An algorithm for the analysis is presented below. The routine `SymbolicSimulate` performs a zero-delay symbolic simulation of the circuit. The algorithm takes in a circuit called **ckt** and generates a new temporally analyzed circuit called **ckt'**, while leaving **ckt** unchanged.

```
TemporalAnalysis(ckt)
    time = 0
    apply inputs and clocks at time 0
    introduce variables for initial present states
    SymbolicSimulate(ckt)
    for each unique net in ckt, create new net in ckt'
        sample outputs at time 0
        while (time < cycle_time)
            apply any new data inputs and/or clocks
            SymbolicSimulate(ckt)
            for each net in ckt
                if there is no logically equivalent net in ckt'
                    create new net
                sample outputs at current time
            update time to point of next change in DI or ϕ
    return ckt'
```

The procedure analyzes the circuit for one complete clock cycle. It begins by applying values to constant data inputs and clocks at time 0. Non-constant data inputs and present state nets are assigned symbolic variables. Then a zero-delay symbolic simulation of the circuit is performed until the circuit stabilizes. After each symbolic simulation call, nets are added to **ckt'**. Logical equivalence is checked by building up BDDs for each net. The routine `SymbolicSimulate` represents a gate-level symbolic simulator that has been developed by our group. It uses binary decision diagrams [3] as a platform for logic manipulation. The simulator uses a nominal transport delay model and an interpretive implementation method, as described in [1].

To initialize values, present states for the first stage were represented by introducing new variables. These were created for temporary purposes, as they actually represented states from the previous cycle. Upon completion of the algorithm, these variables must be removed. The initial present state nets now become cycle delays to the next state nets in the last stage. Referring back to Figure 4, the initial present state nets are PS(0). The next state nets in the last stage are NS(Δ-1). Note that this net is equivalent to the value of the artificial net PS(Δ). During this step, the temporary variables introduced for the PS(0) nets are replaced by cycle delays to PS(Δ) nets. Since the cycle delays are dependent on the values of the next states in the last stage, we must ensure that each of these nets are represented in our circuit. Thus, our tool automatically identifies each of the state nets and marks each of these nets to be visible at time 'cycle_time - 1', which is the last discrete time in our cycle. The values at this time represent the values in the last stage of the cycle, and thus guarantees that the next states in the last stage will be represented in the temporally analyzed circuit.

4 Mead and Conway Stack Example

An example is now presented to help explain the algorithm. Figure 6 shows a block diagram for a pipelined stack, as described by Mead and Conway [6].

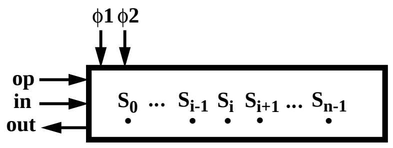

Figure 6 Block Diagram for Mead and Conway Stack

A **PUSH** operation will shift the data input (**in**) into the stack and shift stored data ($\mathbf{S_i}$) deeper into the stack (to the right). A **POP** operation does the reverse -- the data stored in state $\mathbf{S_0}$ is outputted and the rest of the data is shifted up one level (to the left). The third mode is a **HOLD**, which retains the states in the stack. The circuit uses a two-phase nonoverlapping clock. The mode of operation is selected by the sequence of values on the op signal. Figure 7 shows the timing for the circuit.

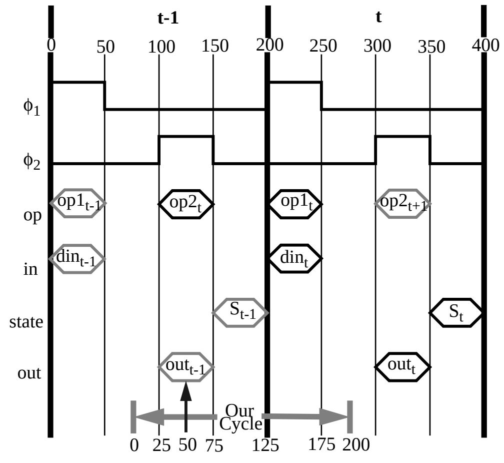

Figure 7 Timing Diagram for Stack

To represent the three modes, two states are multiplexed onto the control signal **op**. Note that **op2** is set during the previous cycle, therefore creating a pipelined design. Table 1 gives the encoding for these modes of operation.

Table 1 Modes of Operation for Stack

op2	op1	mode	stack operation
1	0	PUSH	$S_{i-1} \rightarrow S_i$
0	1	POP	$S_{i+1} \rightarrow S_i$
0	0	HOLD	$S_i \rightarrow S_i$

In order to incorporate all of the input information in one clock cycle, we have chosen to shift the cycle boundaries, as shown in Figure 7. As the user, we have chosen output net $\mathbf{out_t}$ to be visible at time 50 within our cycle. This net is a function of the data input from the previous cycle and the op-signal from both the previous and present cycle. The bold signals represent the information supplied to the temporal analyzer.

Figure 8 shows a generalized section of the circuit generated after the temporal analysis has been performed. The figure represents the circuitry required for the ith bit of the stack. Referring back to Table 1, the functionality of state $\mathbf{S_i}$ is correct for the three modes of operation. For example, when **op1**=**op2**=0, a **HOLD** operation is performed. From Figure 8, this means that states $\mathbf{S_{i+1}}$, $\mathbf{S_i}$, and $\mathbf{S_{i-1}}$ all retain their previous value. For a **POP** operation, the values of the states get updated as follows: $\mathbf{S_{i+2} \rightarrow S_{i+1}}$, $\mathbf{S_{i+1} \rightarrow S_i}$, and $\mathbf{S_i \rightarrow S_{i-1}}$. The values in each state is shifted up one level. The **PUSH** operation is the reverse. The fourth case, when both **op1** and **op2** are high, was designed to be a don't care. Our tool has correctly identified this case to implement a **HOLD** operation. The first and last bits of the stack represent special cases and added circuitry is produced to represent their functionality.

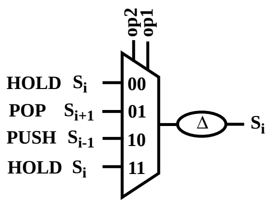

Figure 8 Temporally analyzed stack (generalized case)

5 Results

The temporal analysis was performed upon three classes of circuits. The first two are Manchester adders (AdderX) and counters (CounterX) and make heavy use of precharged logic. The last category consists of dynamic RAMS (RamX) made up of 1 bit words. Table 2 shows the results of the tests. All experiments were performed on a DEC 5000 workstation with 500 MB of RAM.

The columns labeled "SLA" correspond to the gate circuit generated after the switch-level analysis in Tranalyze, without any tem-

Table 2 The effect of temporal analysis

	Analysis (sec)		Memory (MB)		Gate Count		Simulation (sec)	
	SLA	TA	SLA	TA	SLA	TA	SLA	TA
Adder1	1.8	7.6	0.67	1.16	214	111	11.9	3.6
Adder4	4.1	11.2	0.74	1.41	298	142	64.4	11.4
Adder16	34.0	114.3	1.06	8.59	1186	550	116.7	4.5
Counter4	0.8	1.2	0.67	0.74	184	62	17.9	1.6
Counter16	7.7	8.0	0.90	1.06	816	235	33.8	1.0
Counter64	32.7	40.8	1.72	2.81	3320	917	82.4	0.5
Ram16	6.8	8.0	0.80	0.96	594	301	35.2	2.8
Ram64	40.1	42.0	1.12	1.75	1809	993	126.9	6.9
Ram256	240.4	265.8	2.81	5.50	9927	3511	591.0	48.0

poral analysis. The columns labeled "TA" correspond to the circuit generated after the temporal analysis

The temporal analysis reduces the gate count by 40%-70%. The circuit was simulated using Cadence Verilog-XL 1.7 and exhibited a speedup of 3X-150X. The simulation speed comes at a price, that being increased CPU time. However, this is a one-time cost and the circuit can then be resimulated without first performing the analysis.

The temporal analysis displayed a greater speedup for larger circuits. Thus the benefits of temporal analysis may be further realized on even larger circuits.

6 Conclusions and Future Work

We have developed a tool that performs a temporal analysis on gate-level circuits. The net effect of this temporal analysis is that the clocks are abstracted from the circuit, and a new gate-level circuit is produced. This new circuit has applications in simulation, formal hardware verification, and reverse engineering of existing circuits. We have observed a significant reduction in the size of the circuit after a temporal analysis is performed. The speedup of the simulation ranges from 3X-150X, with speedup increasing as the size of the circuit increases.

One major limitation we discovered is the memory needed to perform the analysis. Our tool uses BDDs, which can easily become very large if a non-optimal variable ordering scheme is used. Therefore, we should focus future efforts on ensuring a good variable ordering to control the size of the BDD.

Currently, our temporal analysis tool samples outputs at discrete points in time. The formal verification strategy used by our group requires outputs to be valid over a range of time, so it would be advantageous for us to extend our temporal analysis so that outputs are sampled over a range of time. This extension would also allow the analysis to generate an even more abstract circuit. For instance, the next logical step would be to extract a finite state machine from the temporally analyzed circuit.

References

[1] Z. Barzilai, J. L. Carter, B. K. Rosen, and J. D. Rutledge. "HSS--A High-Speed Simulator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 6, No. 4, July 1987, pp. 601-617.

[2] D. L. Beatty and R. E. Bryant. "Formally Verifying a Microprocessor Using a Simulation Methodology," *31st ACM/IEEE Design Automation Conference Proceedings*, 1994, pp. 703-709.

[3] R. E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, August 1986, pp. 677-691.

[4] R. E. Bryant. "Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis," *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, 1991, pp. 350-353.

[5] Timothy Kam and P.A. Subrahmanyam. "Comparing Layouts with HDL Models: A Formal Verification Technique," *Proceedings of the International Conference on Computer Design*, 1992, pp. 588-591.

[6] C. A. Mead and L. A. Conway. *Introduction to VLSI Systems*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1980.

[7] R. Razdan and G. Bischoff. "Clock Suppression Techniques for Synchronous Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.12, No.10, October 1993, pp.1547-1556.

[8] Y. Takamine, S. Miyamoto, S. Nagashima, M. Miyoshi, and S. Kawabe. "Clock Event Suppression Algorithm of VELVET And Its Application to S-820 Development," *25th ACM/IEEE Design Automation Conference Proceedings*, 1988, pp. 716-719.

[9] E. Ulrich. "A Design Verification Methodology based on Concurrent Simulation and Clock Suppression," *20th ACM/IEEE Design Automation Conference Proceedings*, 1983, pp. 709-712.

[10] T. Weber and F. Somenzi. "Periodic Signal Suppression in a Concurrent Fault Simulator," *Proceedings of The European Conference on Design Automation*, 1991, pp. 565-569.