Timing Verification Using HDTV

Alan R. Martello, Steven P. Levitan, and Donald M. Chiarulli

University of Pittsburgh
Pittsburgh, PA 15261

Abstract

In this paper, we provide an overview of a system designed for verifying the consistency of *timing specifications* for digital circuits. The utility of the system comes from the need to verify that existing digital components will interact correctly when placed together in a system. The system can also be used in the case of verifying specifications of unimplemented components.

1 Introduction

As digital design becomes more complicated, more engineers are looking for an off-the-shelf solution to their design problems. Although the design, integration, and testing of a variety of off-the-shelf components may not be trivial, it is usually easier than a full custom design.

Since we see the future of design continuing in this direction, we have been looking at ways to reduce the time from concept to working design of off-the-shelf components. We have been experimenting with describing interface timing in order to perform reasoning about the interconnection of such components. Our view of verification is based on the fact that the designer is *not* free to specify arbitrary components or specifications. Rather, the designer is limited by the available commercial technology. Since the components have been predetermined, we are attempting to verify proper interfacing according to their timing specifications.

We believe our approach is attractive for the following reasons:

- The interface description is concerned only with interface timings and constraints and is not tied to the behavior or function of the components.
- The component interface is completely generic and does not presuppose implementation technology or scale; it can be equally applied to digital logic gates, subsystem interconnections, and bus interfaces.
- We are concerned with verifying component intercorrection and are not considering synthesis. We are attempting to answer the question "can I interconnect these components" not "what circuit can I use to interconnect these components."
- The type of circuit does not matter; our approach works equally well for synchronous or asynchronous sequential circuits.

Our system is designed to be used for verifying the consistency of *timing specifications* for digital circuits. Verification in this case is defined to be a test that the *timing requirements* of each of the components of the system are met by the interacting *timing causalities* of all other components of the system. The utility of the system comes from the need to verify that existing digital components will interact correctly when placed together in a system. The system can also be used for verifying specifications of unimplemented components.

To perform this interface verification, we have defined two operators which allow us to perform useful reasoning concerning the interface. One operator deals with causality and another with timing constraints; they are discussed in greater detail in section 3. We believe the simplicity of our current operator set is an asset in describing circuits since it tends to de-mystify the interface specification. We have implemented a Hardware Design Timing Verification (HDTV) program to test our ideas and substantiate our work.

This paper is organized as follows. Section 2 provides a brief overview of related work. Section 3 describes our set of operators, the type of reasoning performed with these operators, and the algorithms which we use. Section 4 provides a substantial example which illustrates the verification which can be performed with our operators. Finally section 5 contains a conclusion and summarizes future research.

2 Related Work

Early work in verification of timing constraints in large synchronous systems were limited by the need for user input describing system clocks and state [1,2]. Other work concentrated on describing interfaces with hardware I/O interface languages [3,4]. This approach stressed formal definition of the interface and could be viewed as the verification of the synchronization of independent processes [5].

Modifications to formal logic reasoning have also been explored in the hope that if we can model a circuit or system in a formal way, then we should be

able to perform intelligent reasoning about it [6–11]. Dill extended some of the concepts from [10] to create a version of trace theory for verification of speed-independent circuits [12]. In contrast to formal mathematical logic, heuristics have also been suggested for timing verification [13].

A related area has been research into synthesizing the behavior of circuits and the interface between those circuits [14–17]. This approach creates interfaces which are "correct by construction" and therefore no verification is needed. In the synthesis process, behavior and timing have been considered as two separate concerns.

For non-trivial synthesis there are indications that both timing and behavior must be considered in tandem during the synthesis process [15,18–20]. The type of "verification" performed during synthesis is unnecessary if the synthesis process guarantees that the constraints specified are met or if the synthesis process iteratively modifies the circuit generated until the constraints are met.

On the other hand, our approach of interface timing verification of arbitrary hardware is a worst case analysis of the interfaces which have been pre-determined. Work in a similar vein has been presented in [21,22]. Another approach which has been taken in timing verification is one of an abstract timing verifier [23–25]. This system performed timing analysis by viewing time represented as arcs spanning a graph.

Although digital design has been occurring for many years, the interfacing of components has chiefly been left up to the human to perform. This is one reason that the verification of proper hardware interaction from interface descriptions is an ongoing research topic [26]. Timing interface verification has typically been considered from one of two limiting approaches: either the domain is limited by the nature of the input specification or by the type of synthesis algorithms applied. **We present our approach which we feel is not** limited to any particular domain or implementation paradigm.

3 Interface Timing Verification

3.1 Event Types and Operators

The basic timing paradigm we use views time as a series of discrete events on signals. These events are separated by periods of quiescence. In our analysis, we have defined four events: rising edge ('/'), falling edge ('\ '), signal going stable ('+'), and signal going unstable ('- '). We have found these four events sufficient for non-trivial analysis, but our system is not constrained to these event types.

We have defined two operators which relate signal events. One operator signifies causality and is represented as ' $\rightarrow$ ' while the other operator indicates a timing constraint relationship which must be satisfied and is represented as ' | '.

An expression with either operator must have a time range associated with it. This time specification is in the format (min_time, max_time) where *min_time* or *max_time* may be negative or positive. The only restriction on the time range is that the *min_time* must not be greater than the *max_time*. Infinity and negative infinity are also representable in the time range specification as '*' and '-*' respectively. The units for the time ranges are arbitrary time units; the only stipulation on units is that they must be the same for all time specifications.

The causality operator (' $\rightarrow$ ') is used to signify that one event *conceptually* causes another event. The ' $\rightarrow$ ' operator does not necessarily express true causality since we do not know if there is any causal relationship between the two events within the electronic device. The causality operator can also be used with a stable / unstable event on the *rhs* (right hand side) of the expression. In this case, perhaps *validates* and *invalidates* are more appropriate terms to use instead of *causality*.

The constraint operator (' | ') specifies a timing relationship between two events which must be satisfied for proper operation of a component. The specification of these constraints are typically obtained from specifications like "setup and hold" times. The specification of *all* desired constraints for *all* components is critical for the interface timing verification process.

For both operations, the time range which is associated with the operation expresses the relationship between the lhs event and the rhs event of the expression. Example constraint and causality expressions are shown in Table 1.

3.2 Interface Conflict Detection

The ability to define known relationships between events (causalities) and the ability to define necessary conditions for proper operation (constraints) allows us to reason about an interface by verifying that no constraints are violated by any possible sequence of causal events. A sequence of events which leads to a constraint violation is termed a conflict. There are two kinds of conflicts: direct conflicts and indirect conflicts.

A direct conflict is a conflict which can be determined by comparing the causalities, or chain of causalities to a constraint. For example, given $a_1 \rightarrow a_2\ (s_1, e_1)$ and $a_1 \mid a_2\ (S_1, E_1)$, if $S_1 > s_1$ or $E_1 < e_1$ then we have a direct conflict. This is the degenerate case for a direct conflict and can be found by direct comparisons of causalities and constraints. The general direct conflict case is expressed by a chain of

Table 1: HDTV Expression Examples

HDTV Expression	Meaning
`ld1 \ -> q1 + (34,45)`	data output (q1) is stable somewhere between 34 and 45 time units after load 1 (ld1) falls
`un1 / -> q1 - (0,0)`	data output (q1) is unstable immediately when unload (un1) rises
`d1 + | ld1 \ (50,*)`	load 1 (ld1) must fall more than 50 time units after data input (d1) is stable
`d1 - | ld1 \ (-*,-1)`	load 1 (ld1) must fall at least one time unit before data 1 (d1) goes unstable

causalities. Given a chain of causalities:

$$a_1 \rightarrow a_2\ (s_1, e_1)$$
$$a_2 \rightarrow a_3\ (s_2, e_2)$$
$$\vdots$$
$$a_n \rightarrow a_{n+1}\ (s_n, e_n)$$

we can express this chain as a collapsed causality $a_I \rightarrow a_{II}\ (s_I, e_I)$ where $a_I \equiv a_1$, $a_{II} \equiv a_{n+1}$, $s_I = \sum_{i=1}^{n} s_i$, and $e_I = \sum_{i=1}^{n} e_i$. Thus, a chain or sequence of causalities is equivalent to a single causality with its minimum the sum of the minimums on the chain and its maximum the sum on the maximums. We are defining this equivalent causality as the "worst case" causality of the chain. Given this equivalent causality, it can be compared to the constraint $a_I \mid a_{II}\ (S_I, E_I)$ (if it exists) and if $S_1 > s_1$ or $E_1 < e_1$ then we have a conflict.

The second type of conflict is an indirect conflict and its presence must be inferred. The steps to infer the conflict are as follows.

- given a constraint $B \mid C\ (S_{BC}, E_{BC})$
- If there exist two chains $A \rightarrow b_1\ (s_{b_1}, e_{b_1})$, $b_1 \rightarrow b_2\ (s_{b_2}, e_{b_2})$,$\cdots$, $b_{n-1} \rightarrow B\ (s_{b_n}, e_{b_n})$ and $A \rightarrow c_1\ (s_{c_1}, e_{c_1})$, $c_1 \rightarrow c_2\ (s_{c_2}, e_{c_2})$,$\cdots$, $c_{m-1} \rightarrow C\ (s_{c_m}, e_{c_m})$
- determine the collapsed causality of each of the causality chains $A \rightarrow B\ (s_{AB}, e_{AB})$ and $A \rightarrow C\ (s_{AC}, e_{AC})$ where $S_{AB} = \sum_{i=1}^{n} s_{b_i}$ and $S_{AC} = \sum_{i=1}^{m} s_{c_i}$
- define event B as occurring at relative $time = 0$
- back project the event chain $A \rightarrow B\ (s_{AB}, e_{AB})$ and find s_A and e_A which are the starting and ending times at which event A would have occurred to trigger event B ($s_A = -e_{AB}$ and $e_A = -s_{AB}$)
- forward project the causality $A \rightarrow C\ (s_{AC}, e_{AC})$ to see if the time at which event C occurred meets the constraint; i.e. verify that $(s_A + s_{AC}) > s_{BC}$ and $(e_A + e_{AC}) < e_{BC}$
- if these inequalities hold true, then the constraint is satisfied, otherwise the constraint is violated and we have an indirect conflict

3.3 Reasoning About Cycles

This situation becomes more complicated if we know that some signal on the chain A $\rightarrow$ B or A $\rightarrow$ C is cyclic in nature. This often occurs if we have a clock specification or a handshake protocol.

If one of the signals on either of the chains is cyclic, then the indirect verification test described above must take into account the period of the cycle. For example, if we have a signal on one of the chains, σ which is periodic of period t_σ, then the verification equations become $(s_A + s_{AC} + N * t_\sigma) > s_{BC}$ and similarly $(e_A + e_{AC} + N * t_\sigma) < e_{BC}$. N is the number of cycles which we need to add (or subtract if $N < 0$) to the verification equations to see if the comparison is valid.

The fact that $N \neq 0$ implies we have "period jumping". Period jumping comes about from the fact that when periodic signals are involved in an interface, the "starting time" of the periodic signal can be randomly chosen. This means that for proper verification, the correct period for comparison must determined.

Therefore, given the periodic signal σ on one of the chains, the problem is how to determine N. This can be done by finding N such that

(I) $(s_A + s_{AC} + N * t_\sigma) < 0$ and
$(e_A + e_{AC} + N * t_\sigma) > 0$ or
(II) $(s_A + s_{AC} + N * t_\sigma) > 0$ and
$(e_A + e_{AC} + (N-1) * t_\sigma) < 0$.

If (I) is satisfied, then N is the number of period jumps to perform and the comparison can be made directly. If (II) is satisfied, then correct cycle number needs to be determined (N or $N-1$).

The following determines whether to use period N or $N-1$ for case II:

- if the event associated with C is stable or unstable, then use the period value for N which results in N being closer to zero, that is if

$(s_A + s_{AC} + N * t_\sigma) > -(e_A + e_{AC} + (N-1) * t_\sigma)$ use $N-1$ for the number of periods, otherwise use N

- if C is rising or falling, then choose the number of cycles (N or $N-1$) such that the number of events (i.e. edges) crossed from $t = 0$ to either $(s_A + s_{AC} + N * t_\sigma)$ or to $(e_A + e_{AC} + (N-1) * t_\sigma)$ is zero (see Figure 1).

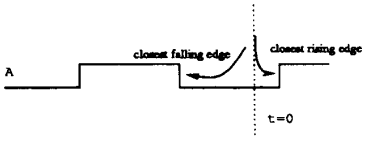

Figure 1: Closest Edge Determination

Once N has been determined, then the comparison can be made as shown above and thus the verification is performed.

4 Examples

We have implemented a system which can accept interface constraints and causalities, associate logical connection among the constraints and causalities, and perform the inference reasoning as described above. A causality graph is built within the system to determine cycles and causality chains which are needed for direct and indirect conflict detection.

This example shows how HDTV can be used for verifying asynchronous circuits in a handshake protocol. Figure 2 shows an asynchronous circuit made by connecting two SN74LS2222 FIFO circuits together. Each FIFO has two clocks (load and unload), two ready signals (input ready and output ready), and I/O data lines (D0-D3 and Q0-Q3). The FIFO's are designed to be cascaded by connecting the first FIFO's Q outputs to the second FIFO's D inputs. Handshake control comes from connecting the first FIFO's output ready to the second FIFO's load clock, and the input ready of the second FIFO to the unload clock of the first FIFO. The causality and requirements information in this example come from the Texas Instruments data book [27].

Two different interconnections are used. In both, the unload signal of the first FIFO is connected to the input ready of the second FIFO, and the data lines are connected. The first (ORIGINAL) shows the direct connection of the output ready line of the first FIFO with the load signal of the second FIFO. In the second (DELAYED) version a 10ns delay is inserted into this signal path.

Figures 3a and 3b show part of the output generated from two runs of the HDTV system for this example.

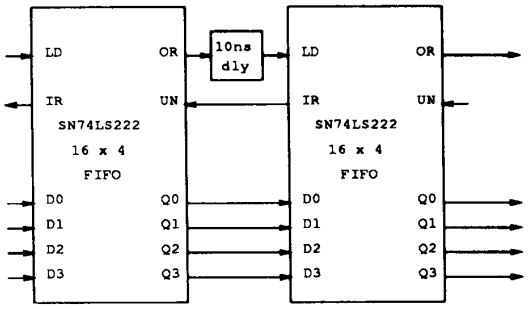

Figure 2: FIFO Interface

```
Original Test -- (a)
Cycles:
cycle time of ld2 \ = (110,110)   rise to fall = (55,55)
cycle time of un1 / = (110,110)   rise to fall = (59,59)

indirect conflict with rule 22: r21 d2 + | ld2 \ (50,*)
     real time [no cycle]: (-73,-71)
     real time [w/ cycle]: (37,39)

(trace back)
  • rule 9:   un1 / -> d2 + (46,48)
  • rule 13:  ld2 \ -> un1 / (25,25)
  • rule 6:   un1 \ -> ld2 \ (26,26)
  • rule 14:  ld2 / -> un1 \ (31,31)
  • rule 4:   ld1 \ -> ld2 / (45,45)

  • rule 6:   un1 \ -> ld2 \ (26,26)
  • rule 14:  ld2 / -> un1 \ (31,31)
  • rule 4:   ld1 \ -> ld2 / (45,45)
Number of rule conflicts: 8
------------------------------------------------------------
Delayed Test -- (b)
Cycles:
   cycle time of ld2 \ = (130,130) rise to fall = (65,65)
   cycle time of un1 / = (130,130) rise to fall = (69,69)
Number of rule conflicts: None
```

Figure 3: SN74LS222 Test
Original Test (a) and Delayed Test (b)

Note that the signal names reflect the name substitution implied by the hookup information. In figure 3a we used the original hookup of clock signals and in figure 3b we used the delayed version. Figure 3a shows how HDTV discovers a timing conflict between a 50ns setup time requirement on input data and the input clock for the second FIFO. As explained above, HDTV first discovers a cycle on the clock signals and then uses this information to identify the appropriate pairs of events to compare for conflicts. The second falling edge on ld2 does not meet the 50ns setup time on the second piece of data coming out of the first FIFO, and therefore an error is reported. In figure 3b the addition of a 10ns delay element resolves this problem. Figures 4a and 4b show hand generated timing diagrams which clarify this situation. The circled

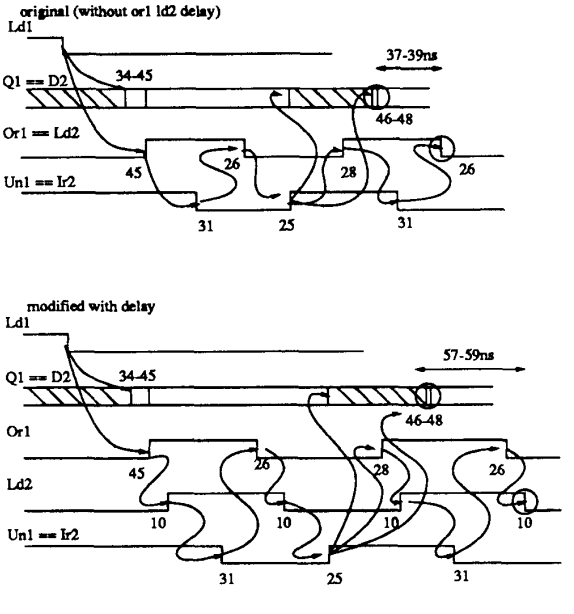

Figure 4: FIFO Interface Timing

events on the data lines (Q1 D2) and the Ld2 lines reflect the critical behavior of the system. It should be noted that the Texas Instruments data-book does in fact show a 10ns delay element in its example configuration diagram, with no explanation.

5 Summary, Conclusions, and Future Research

We have presented a system for verifying the consistency of *timing specifications* of digital circuits. Verification in this case is defined to be a test that the *timing requirements* of each of the components of the system are met by the interacting *timing causalities* of all other components of the system. The utility of the system comes from the need to verify that existing digital components will interact correctly when placed together in a system. The system can also be used in the case of verifying specifications of unimplemented components as well.

The system uses a simple model of events on signals, and two kinds of relationships between those events: causality and requirements. These are analogous to "recommended conditions" and "switching characteristics" in published data sheets. They are also similar to Seitz's *functional relations* and *domain relations* [28].

In this work, we have abstracted out the notion of timing from the more general notion of behavior by considering only the control information of the circuits. Data values are not considered. The only "values" used are stable and unstable. By doing this we have simplified the domain that both the user and the system need to reason about. We believe this abstraction will become more and more useful as the complexity of systems and their temporal behavior continues to grow.

Clearly, there is a place for data values used to enable or disable particular events in the system. While we have shown that the system is already useful without that capability, it is an essential extension to the work. We plan to incorporate the notion of "system state" which will be a vector of control values that can be used to limit the set of relations that are active at any given time. This will be useful for such cases as modeling the difference between read cycle and write cycle timing for a bus. As signal events cause transitions the state vector will change, thus enabling or disabling subsets of the causality expressions.

Even with this extension HDTV will not be performing a simulation but rather a verification. No input stimulation, test vector, or user supplied script will be necessary to verify the correct operation of the system.

Another extension of the system will be to provide the user with a graphical interface which will allow for both input and output of timing waveforms as an alternate method of specifying the relations and reporting the results of the analysis.

It should be noted that the "hand verification" of timing conflicts discovered by the system becomes time consuming even for the non-trivial yet fairly simple example shown above. Specifying the interfaces initially consists of referring to manufacturer's specifications for proper operation (the constraints) and the specifications for the interface's response (the causality). This process of specifying the interface is much easier than hand verifying the interface. This fact leads us to believe that our approach is fundamentally useful and worth pursuing for interface timing verification at a variety of levels.

References

[1] Thomas M. McWilliams, "Verification of Timing Constraints on Large Digital Systems," *Proc. 17th Design Automation Conference* (June, 1980).

[2] Robert B. Hitchcock, Sr., "Timing Verification and the Timing Analysis Program," *Proc. 19th Design Automation Conference* (June, 1982).

[3] Alice C. Parker & John J. Wallace, "SLIDE: An I/O Hardware Descriptive Language," *IEEE Trans. on Computers* C-30 (June, 1981), 423–439.

[4] Alice C. Parker & Nohbyung Park, "Interface and I/O Protocol Descriptions," in *Hardware Description Languages*, Advances in CAD for VLSI #7, Elsevier, Amsterdam–New York, 111–136.

[5] John J. Wallace, "On Automatic Verification of SLIDE Descriptions," Design Research Center, Carnegie-Mellon University, DRC-01-2-80, Pittsburgh, PA, Aug., 1979.

[6] James F. Allen, "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM* 26 (Nov., 1983), 832–843.

[7] Gregor V. Bochmann, "Hardware Specification with Temporal Logic: An Example," *IEEE Trans. on Computers* C-31 (Mar., 1982), 223–231.

[8] Ben Moszkowski, "A Temporal Logic for Multilevel Reasoning about Hardware," *IEEE Computer* (Feb., 1985).

[9] Philip A. Wilsey, *Computer Architecture Specification with Interval Temporal Logic*, University of Cincinnati, Dept. of Elec. and Comp. Engineering, May 10, 1989.

[10] M. Browne, E. Clarke, D. Dill & B. Mishra, "Automatic Verification of Sequential Circuits Using Temporal Logic," Dept. of Computer Science, Carnegie-Mellon University, CMU-CS-85-100, Pittsburgh, PA, Dec., 1984.

[11] E. M. Clarke, D. E. Long & K. L. McMillan, "A Language for Compositional Specification and Verification of Finite State Hardware Controllers," Carnegie-Mellon University, Computer Science Dept., CMU-CS-89-110, Jan., 1989.

[12] David L. Dill, "Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits," Dept. of Computer Science, Carnegie-Mellon University, CMU-CS-88-119, Pittsburgh, PA, Feb., 1988.

[13] Atsushi Kara, Ravi Rastogi & Kazuhiko Kawamura, "An Expert System to Automate Timing Design," *IEEE Design & Test of Computers* (Oct., 1988).

[14] J. A. Nestor & D. E. Thomas, "Behavioral Synthesis with Interfaces," *Proc. 1986 Inter. Conf. for Computer-Aided Design* (Nov., 1986).

[15] Sally Hayati & Alice Parker, "Automatic Production of Controller Specifications From Control and Timing Behavioral Descriptions," *Proc. 26th Design Automation Conference* (June, 1989).

[16] Gaetano Borriello & Randy H. Katz, "Synthesis and Optimization of Interface Transducer Logic," *Proc. 1987 Inter. Conf. for Computer-Aided Design* (Nov., 1987).

[17] Gaetano Borriello, "A New Interface Specification Methodology and its Application to Transducer Synthesis," Computer Science Division, Univ. of Calif. at Berkeley, UCB/CSD 88/430, Berkeley, CA, May 26, 1988.

[18] Gaetano Borriello, "Combining Event and Data-Flow Graphs in Behavioral Syhtnesis," *Proc. 1988 Inter. Conf. for Computer-Aided Design* (Nov., 1988).

[19] Tod Amon, Gaetano Borriello, Wayne Wilder & Carlo Séquin, "A Unified Behavioral / Structural Representation for Simulation and Synthesis," Northwest Lab. for Integrated Systems, University of Washington, LIS TR 89-30-03, Nov. 1, 1989.

[20] Teresa H.-Y. Meng, Robert W. Brodersen & David G. Messerschmitt, "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications," *IEEE Trans. on Comp.-Aided Design* 8 (Nov., 1989), 1185–1205.

[21] Steven K. Sherman, "Algorithms for Timing Requirement Analysis and Generation," *Proc. 25th Design Automation Conference* (June, 1988).

[22] Tomohiro Yoneda, Kazutoshi Nakade & Yoshihiro Tohma, "A Fast Timing Verification Method Based on the Independence of Units," *Proc. Fault Tolerant Computing Systems* (1989).

[23] David E. Wallace, "Abstract Timing Verification for Synchronous Digital Systems," Computer Science Division, Univ. of Calif. at Berkeley, UCB/CSD 88/425, Berkeley, CA, June 27, 1988.

[24] David E. Wallace & Carlo Séquin, "ATV: An Abstract Timing Verifier," *Proc. 25th Design Automation Conference* (June, 1988).

[25] David E. Wallace & Carlo Séquin, "Plug-In Timing Models for an Abstract Timing Verifier," *Proc. 23rd Design Automation Conference* (June, 1986).

[26] A. LaPaugh & D. Doukas, "Timing Verification," in *Princeton Project on Digital Design*, Andrea S. LaPaugh & Richard J. Lipton, eds., Computer Science Dept., Princeton University, Oct. 31, 1989, 2–3.

[27] Texas Instruments, *The Bipolar Microcomputer Components Data Book for Design Engineers*, 1981.

[28] Charles L. Seitz, "System Timing," in *Introduction to VLSI Systems*, Carver Mead & Lynn Conway, eds., Addison Wesley, Reading, MA, 1980, 245.