Symbolic-Timing-Equation Generation from a High-Level Specification of Interface Behavior*

Ajay J. Daga, William P. Birmingham

Abstract

We present a scheme for the generation of symbolic timing equations that establish the temporal relationship between arbitrary events on an interface transaction. We discuss a representation for the temporal and sequencing aspects of interface behavior, using causal event graphs and state-transition graphs, that is amenable to the equation-generation task. We present algorithms used to traverse these graphs to establish the temporal relation of events with respect to the start of an interface transaction. We also present an algorithm used to establish the temporal relation between any arbitrary event pair.

1. Introduction

The sequencing and temporal aspects of an interface transaction for a component\(^1\) are typically specified through timing diagrams and state diagrams [1, 2]. It is desirable to automatically generate the symbolic timing relation between arbitrary events on an interface transaction. This is useful to support manual and automated synthesis and timing-verification tasks. For example, high-level-synthesis tools need to consider component delays relative to a critical path to aid the part-selection task [3]. Symbolic equations, which are generated from a high-level specification of interface behavior, may be input to linear programming tools to aid such synthesis decisions. Likewise, timing verifiers need to consider path delays. Applications like these need to establish the temporal relation between arbitrary events that do not necessarily have a specified relation.

In this paper, we describe an algorithm that, given a timing diagram and state diagram, will generate a symbolic equation between arbitrary events. While a variety of timing-verification tools [4, 5, 6, 7] use timing equations internally, none of these tools provides equations to the user. We do not focus on the manipulation or solution of symbolic equations, or their transformation from one form to another, but instead on their automatic generation from a high-level specification of interface behavior.

The paper is organized as follows. Section 2 briefly characterizes interface behavior. Section 3 discusses our representation schemes for temporal and sequencing aspects of interface behavior. Section 4 presents the algorithms used to traverse a causal-event-graph. Section 5 outlines the algorithm used to traverse a state-transition graph. Section 6 discusses the technique used to generate timing equations between an arbitrary event pair and Section 7 summarizes this paper.

2. Interface Behavior

The interface behavior of a device is defined by a collection of bus cycles that describe how a device communicates with its environment. A bus cycle may be synchronous or asynchronous, master or slave [8]. A bus cycle is composed of a sequence of bus states. Bus states define subactivities of a bus cycle, i.e., signal activity on a set of interface ports for a portion of the bus cycle.

---

This research was supported by Digital Equipment Corporation and the National Science Foundation grant MIPS-905781. All views expressed here are those of the authors, and not necessarily those of the funding agencies.

\(^1\)Component is used in a general sense, indicating a VLSI device (e.g., a microprocessor), or a cell in an ASIC cell library.
A sequence of bus states that form a complete bus cycle is called a transaction path for the bus cycle. A read bus cycle for the Motorola MC68020, for example, consists of bus states $S0, S1, S2, S3, S4$ and $S5$ [9]. Taken in that order, these bus states represent a transaction path for the read cycle. A bus cycle may have multiple transaction paths. We assume that asynchronous cycles, such as a fully-interlocked handshake [8], have a single transaction path.

The temporal information associated with signal activity during a bus cycle is typically documented using timing diagrams. If a bus cycle has a single transaction path, then a timing diagram completely specifies the sequencing of events for a given cycle. These diagrams do not, however, convey other non-temporal information needed for a complete interface specification, such as conditional information associated with a bus cycle. Conditional information is required, in the event of multiple transaction paths, to specify transaction paths as a function of signal values. Thus, in addition to timing diagrams, state diagrams are required to specify sequencing aspects of interface behavior.

3. Representation of Temporal and Sequencing Information

3.1 Causal Event Graphs

A typical timing diagram, captured using the tool Xwave [10], specifying the read bus cycle for the Motorola MC68020 [10], is shown in Figure 3.1. The labels $S0$ through $S5$ associated with the phases of the $CLK$ signal, referred to as a sequencing signal, represent different bus states of the read cycle. A sequencing signal is one that sequences a cycle through its different bus states.

A representation of temporal information that supports the equation-generation task is a causal event graph (CEG). An event graph is a directed acyclic graph (DAG) whose nodes, $E = \{e_1, e_2, ..., e_n\}$ represent events, and arcs, $A = \{t_1, t_2, ..., t_m\}$, represent timing links between pairs of events. A timing link, $t_l$, is represented by a directed arc from $e_{from}$ to $e_{to}$. An event graph essentially encapsulates the timing behavior of a bus cycle with logic levels abstracted away.

While there are two types of timing links shown on a timing diagram, causal and constraint [10], for the purpose of equation generation they are both treated and represented identically (constraint links are represented as causal links). This assumes that input events will occur at times consistent with specified timing constraints. Hence we refer to this representation as a CEG. Note, in general, the time of occurrence of an input event cannot be anticipated, and therefore, the need to represent causal and constraint links differently [10]. The CEG for the timing diagram of Figure 3.1 is shown in Figure 3.2.

Events on a sequencing signal are referred to as sequencing events. The representation of a sequencing signal is such that each sequencing event $e_i$ is associated with a bus state $s_i$, the state entered by an interface when the event $e_i$ occurs. For example, $CLKR1$, a sequencing event, is associated with state $S0$ in Figure 3.1.

The undirected CEG is connected and has a single component. A CEG that has more than one component manifests an incompletely specified timing diagram. It is the connected property of a CEG that allows the temporal relation of an arbitrary event pair to be established.

A CEG has a set of events that serve as temporal reference points for other events. These events are referred to as reference events. Reference events are cut-vertices of the undirected CEG. There are two important properties with regard to the reference events on a CEG:

- If there is no state diagram associated with a bus cycle, then there is exactly one reference event, the start event $e_s$. The start event is the first event to occur in a bus cycle and is specified by a user.
• When a bus cycle has multiple transaction paths, and consequently a state diagram is specified, then the number of reference events is the number of sequencing events with an out-degree greater than one. The events rooted at a sequencing event \( e_i \) occur when the device enters the bus state \( s_j \) associated with \( e_i \).

Figure 3.1: An example timing diagram.

Figure 3.2: Example CEG.
3.2. State-Transition Graphs

State diagrams are captured using the tool Xstate [10]. Every state diagram has a single start state, $s_0$, one or more final states and transitions between states. Each state transition is labeled with the sequencing event at which the transition is made. In addition, if sequencing between states is dependent on the logic levels sensed at input signals then a conditional expression is used to specify this sequencing information. For the purpose of equation generation, conditional expressions may be omitted from a state diagram, and a non-deterministic state diagram specified (Figure 5.1).

A state diagram specifying sequencing aspects for the read cycle of the MC68020 is shown in Figure 3.3. The sequencing signal is $CLK$. A state diagram is represented as a state-transition graph, that essentially has the same structure as the diagram.

![State diagram for the read bus cycle of the MC68020](image)

Figure 3.3: State diagram for the read bus cycle of the MC68020.

4. Traversing Causal Event Graphs

Traversing a CEG refers to the propagation of timing values from a set of reference events $R = \{e_1, e_2, ..., e_k\}$ to events that are connected directly or indirectly to $R$ [10]. Event-graph traversal is necessary to define the temporal relation of a non-reference event, connected indirectly to a reference event, through a direct link that subsumes the information contained in the indirect links. Traversal occurs by following arcs leading from a reference event $e_r$ to all non-reference events $e_i \in E$ rooted at $e_r \in R$. The propagation of timing values gives a timing equation, $t(e_i)$, to an event $e_i$, relative to an event $e_r$.

A CEG is traversed using the traverse_graph algorithm, which calls propagate_value to establish timing equations for a non-reference event, and to establish state-transition times (if a state diagram is specified). The propagate_value algorithm calls resolve_reference_event to resolve the situation where an event has multiple reference events. These algorithms are discussed in the following sections.

4.1 The Traverse_Graph Algorithm

The traverse_graph algorithm passes reference events to the propagate_value algorithm. The algorithm consists of two steps that are repeated iteratively until all arcs in a CEG have been traversed. In Step 1, the start event $e_s$ and all sequencing events with an out-degree greater than one are passed to propagate_value. After all reference events have been passed to the propagate_value algorithm there may remain untraversed arcs; for example, the arc between $DSACK0_{f1}$ and $CLK_{f2}$, in Figure 3.2. In this situation (Step 2), those untraversed arcs, $t_f$, whose event $e_{to}$ has received a timing equation are examined. The timing equation associated with the event $e_{from}$ of such arcs is established in the following manner:
If $e_{lo} \in e_r$:
    * $t(e_{from}) = t_l$
    * $e_{r}(e_{from}) = e_{lo}$
Else:
    * $t(e_{from}) = t(e_{lo}) - t_l$
    * $e_{r}(e_{from}) = e_r(e_{lo})$

Step 1 is then repeated, with those events with an out-degree greater than one that received a value through Step 2 passed to propagate_value. The algorithm terminates when all arcs have been traversed.

4.2 The Propagate_Value Algorithm

The propagate_value algorithm is given a reference event $e_r$, and traverses the CEG affixing an equation for events rooted at $e_r$. An event $e_i$ has a set of parent events $P = \{e_{p1}, e_{p2}, ..., e_{pm}\}$ and children events $C = \{e_{c1}, e_{c2}, ..., e_{ck}\}$. The algorithm performs a breadth-first search of the CEG rooted at $e_r$ by traversing arcs connecting a parent event with its children. All the arcs connected directly or indirectly to $e_r$ are traversed, except as follows:

- If event $e_r$ is a sequencing event and is associated with bus state $s_r$, and is connected directly or indirectly to a sequencing event $e_i$ (associated with bus state $s_i$), then $e_i$ is not traversed during a traversal of the CEG rooted at $e_r$.

Furthermore:

- If $e_r$ and $e_{r+1}$ are consecutive sequencing events then the arc $t_l$ between them specifies the state transition time of state $s_r$ associated with $e_r$.

The propagate_value algorithm is shown in Figure 4.1 and an example execution of this algorithm is shown in Figure 4.2.

4.3 The Resolve_Reference_Event Algorithm

Consider the case, depicted in Figure 4.3, where an event $e_i$ has two reference events $e_{r1}$ and $e_{r2}$, and that $e_i$ has received a timing equation relative to $e_{r1}$. When the CEG relative to $e_{r2}$ is traversed $e_i$ will be reached. At this point a decision has to be made, if possible, on which of the two reference events, $e_{r1}$ or $e_{r2}$ to use when determining $e_i$’s timing equation. The resolve_reference_event algorithm performs this task using the following rule:

- If an event $e_i$ (an $e_{lo}$ event) is indirectly or directly connected to events $e_{r1}$ and $e_{r2}$ ($e_{from}$ events), and if it is known that $e_{r1}$ occurs before $e_{r2}$, then the link between $e_{r2}$ and $e_i$ is the true causal link. The link between $e_{r1}$ and $e_i$ is shown for documentation purposes.

Note that for an event to have multiple reference events, these reference events need to be sequencing events. Event $e_{r1}$ is associated with bus state $s1$ and $e_{r2}$ with bus state $s2$. If it can be shown, by traversing the state diagram, that state $s1$ ($s2$) always occurs before state $s2$ ($s1$), then $e_{r2}$ ($e_{r1}$) is the reference for event $e_i$. The traversal of a state diagram to determine if state $s2$ occurs after state $s1$ requires two conditions to be satisfied: (1) it must be shown that state $s1$ has a transaction path to state $s2$, and (2) that state $s2$ does not have a transaction path to state $s1$. 

5
The search for a transaction path between two states is performed by a depth-first search of the state-transition graph. The start state for condition 1 is \( s1 \), and that for condition 2 is state \( s2 \). A state once visited is marked, and is not revisited. Arcs leading into the start state, \( s0 \), (\( s0 \) in Figure 3,3) are not traversed as they represent the start of a new bus cycle. The search terminates for condition 1 (2) when state \( s2 \) (\( s1 \)) is reached, or when there are no more arcs to traverse.

**Propagate_value** (\( e_r \))

\[
\begin{align*}
\text{e}_p & = e_r; \\
t_l & = \text{next}_bfs\_arc(); \\
\text{e}_bfs & = \text{e}_t o(t_l); \\
\text{while} \ (t_l <= \text{null}) \ {\{ \\
\text{change} & = 0; \\
\text{If} (\text{e}_bfs \ \text{not visited and} \ e_p = e_r) \ {/} \text{Illustrated as Case A in Figure 4.2} \ */ \\
\text{t(}e_bfs) & = t_l; \\
\text{change} & = 1; \\
\} \\
\text{Else If} (\text{e}_bfs \ \text{not visited and} \ e_p <> e_r) \ {/} \text{Illustrated as Case B in Figure 4.2} \ */ \\
\text{t(}e_bfs) & = t_l + t(e_p); \\
\text{change} & = 1; \\
\} \\
\text{Else If} (\text{e}_bfs \ \text{visited and} \ e_r(\text{e}_bfs) = e_r) \ {/} \text{e_r(}e_bfs) \ \text{refers to the reference event of} \ e_bfs \ */ \\
\text{t(new)} & = t_l + t(e_p); \ /* \text{Illustrated as Case C in Figure 4.2} */ \\
\text{t(}e_bfs) & = \text{tighter (}t_{\text{new}}, \ t(e_bfs)) \ /* \text{computed using the actual values associated with} \\
\text{If} (\text{t(}e_bfs) = t_{\text{new}}) \ \text{the timing symbols and comparing their lower and} \\
\text{change} & = 1; \ \text{upper bounds [10]} */ \\
\} \\
\text{Else If} (\text{e}_bfs \ \text{visited and} \ e_r(\text{e}_bfs) <> e_r)) \ {\} \\
\text{e_r(}e_bfs) & = \text{resolve_reference_event} (e_r(\text{e}_bfs), e_r, e_bfs); \\
\text{If} (\text{e_r(}e_bfs) <> e_r \ \text{and} \ e_p <> e_r) \ {\} \\
\text{t(}e_bfs) & = t_l + t(e_p); \\
\text{change} & = 1; \\
\} \\
\text{If} (\text{e_r(}e_bfs) <> e_r \ \text{and} \ e_p = e_r) \ {\} \\
\text{t(}e_bfs) & = t_l; \\
\text{change} & = 1; \\
\} \\
\text{If} (\text{change}) \\
\text{e_r(}e_bfs) & = e_r; \\
t_l & = \text{next}_bfs\_arc(); \ /* \text{The next arc to be traversed is chosen so that it satisfies Rule 5.1} */ \\
\text{If} (t_l <= \text{null}) \ {\{ \\
\text{e}_bfs & = \text{e}_t o(t_l); \\
\text{e}_p & = \text{e}_f r o(t_l); \\
\text{If} (\text{e}_p \ \text{and} \ e_bfs \ \text{in consecutive sequencing events}) \\
\text{t}_{\text{trans}} (\text{e}_p(s_p)) & = t_l; \\
\} \\
\} 
\]

**Figure 4.1:** The propagate_value algorithm

If condition 1 is not satisfied, then nothing can be said about the temporal relation between the two states \( s1 \) and \( s2 \). If condition 2 is not satisfied, then \( s1 \) may or may not occur after \( s2 \), and consequently no a priori decision can be made on the temporal relation between the two states. If both conditions are not satisfied both \( e_{r1} \) and \( e_{r2} \) are retained as reference events.
The resolve_reference_event algorithm is illustrated in Figure 4.3. The tree representing the depth-first search of the transition diagram is also shown in Figure 4.3. From the tree shown in Figure 4.3, it is clear that $S0$ has a path to $S1$, and that $S1$ does not have a path to $S0$. Therefore, bus state $S1$ is always reached after $S0$ and, consequently, the falling edge of the clock signal is the true reference event for the falling edge of $AS$.

![Figure 4.2: Illustration of propagate_value](image)

![Figure 4.3: Illustration of resolve_reference_event](image)

The CEG of Figure 3.2, after traversal is shown in Figure 4.4. Note that all non-reference events are directly connected to reference (sequencing) events and that the temporal relation between a non-reference event and a reference event is contained in this direct link. Also, note that the temporal relation of sequencing events with regard to the start event is not yet established. To do so it is necessary to traverse the state-transition graph.

5. **Traversing State-Transition Graphs**

In the presence of sequencing events, it is necessary to traverse the state-transition graph associated with a cycle to determine the timing equations that relate the occurrence of a state relative to the start of a cycle. It is necessary to identify the transaction paths to a state, and loops in a state diagram. Once this is done, the timing equations relating the occurrence of a bus state relative to the start of a cycle may be expressed as a function of: the paths to a state, the time associated with each state-transition in a path (determined by the propagate_value algorithm), and the possible loops that may be encountered (the number of times a loop is executed is expressed as a constant).
$t_{trans}(S0) = T_3$; $t_{trans}(S1) = T_2$; $t_{trans}(S2) = T_3$; $t_{trans}(S3) = T_2$; $t_{trans}(S4) = T_3$; $t_{trans}(S5) = T_2$.

**Figure 4.4:** Example CEG after traversal.

A depth-first search of the state-transition graph is performed starting from $s_0$. Arcs leading into $s_0$ are not traversed, nor are the arcs that signify completion of a bus cycle. For example the arc from $S5$ to $S0$ on the state diagram, shown in Figure 3.3 is not traversed as it leads into $s_0$. Also, the arc from $S5$ to the Read box is not traversed as it signifies the completion of the read cycle. The path being traversed relative to $s_0$ is maintained in current_path. When a state, $s_i$, is visited, and it does not exist in current_path, then current_path is added to the list of paths to $s_i$ from $s_0$. If the state $s_i$ is a member of current_path then a loop in the state-transition graph has been traversed. An index, loop_counter, of the loops in a state diagram is maintained; it initially has a value of 0. Loop_counter is incremented when a loop is identified and with it is associated those states in current_path that form the loop. All paths to a state are enumerated and so are all the loops in a state-transition graph. If, after the traversal of the state-transition graph, there are states that do not have a transaction path from $s_0$, then these state are unreachable. Such a condition manifests an incorrect state-diagram specification.

Figure 5.1 shows an example state-transition graph and the result of traversing this graph.

**Figure 5.1:** Illustration of state-transition-graph-traversal
6. Generating Timing Equations

The timing equation relating an arbitrary event pair, \( e_1 \) and \( e_2 \), may be determined by: finding the closest common ancestor, \( e_a \), to the reference events of \( e_1 \) and \( e_2 \), and generating timing equations that define the temporal relation of \( e_1 \) and \( e_2 \) with regard to \( e_a \).

It is necessary to find the closest common ancestor to a pair of reference events to avoid examining false transaction paths. When a bus cycle has a single transaction path, there is only a single reference event, \( e_s \). The event \( e_s \) is, therefore, also the closest common ancestor, \( e_a \), for any event pair. In the presence of multiple transaction paths, all reference events are sequencing events. Given two reference events \( e_i \) and \( e_j \) associated with bus states \( s_i \) and \( s_j \), if \( s_i = s_j \) then \( e_a = e_i = e_j \). If \( s_i \neq s_j \) then the transaction paths to both states, established by the state-transition-graph-traversal algorithm, are examined. If a transaction path to \( s_i (s_j) \) includes \( s_j (s_i) \) then \( e_j (e_i) \) is the ancestor event for \( e_i (e_j) \).

Note that, in general, either \( e_i \) or \( e_j \), or both, are ancestor events when given a pair of dissimilar reference events \( e_i \) and \( e_j \). This is important because it eliminates the possibility of considering a transaction path that does not include both \( e_i \) and \( e_j \) when determining the temporal relation between events rooted at these reference events. For example, consider the the state diagram shown in Figure 5.1. Assume that \( s_i \) is \( S2 \) and \( s_j \) is \( S3 \), \( e_a \) is \( S2 \). If instead \( e_a \) was assumed to be \( S0 \) then a transaction path that does not include \( S2, S0-S3 \) will be considered when generating equations that establish the relationship between \( S2 \) and \( S3 \).

Given an ancestor event \( e_a \) and a pair of events \( e_1 \) and \( e_2 \) whose reference events are \( e_i \) and \( e_j \), associated with bus states \( s_i \) and \( s_j \), respectively, the timing equations that define the temporal relation of \( e_1 \) with regard to \( e_2 \) are determined as follows.

The equation \( t(e_1/e_2) \) is to be generated. The symbol \( t(e_1/e_2) \) refers to the separation of \( e_1 \) relative to \( e_2 \); \( t(e_1/e_2) = -t(e_2/e_1) \). Without loss of generality let \( e_a = e_j \), then \( t(e_1/e_2) = t(e_1/e_i) + t(e_i/e_j) - t(e_2/e_j) \).

The equation \( t(e_1/e_j) \) is established as follows. There are as many timing equations defining the relation between \( e_i \) and \( e_j \) as paths from \( s_j \) to \( s_i \). The paths from \( s_j \) to \( s_i \) are determined by examining the transaction paths to \( s_i \). These transaction paths have the form \( s_0 \alpha s_j \beta s_i \). There are as many paths from \( s_j \) to \( s_i \) as unique state sequences \( \beta \). Each of these paths yields a timing equation from \( s_j \) to \( s_i \). Each timing equation is composed of the state-transition times associated with the sequence of states that form \( s_j \beta \). In addition, if any of the states in the sequence \( s_j \beta \) are part of a loop in the state-transition graph (recognized using the state-transition-graph-traversal algorithm), then the timing equation is also composed of the transition time of the loop. The transition time of a loop is the sum of the transition times of the states in a loop, multiplied by a unique constant associated with each loop. Each loop is factored into a timing equation only once. The transition time of loops that include the state \( s_i \) are not added to the equation. This is because the timing equation establishes the temporal relation between the last occurrence of \( e_j \) and the first occurrence of \( e_i \).

The equation between a non-reference event and a reference event \( (t(e_1/e_i) \) and \( t(e_2/e_j)) \) is established by examining the arc in the traversed-causal-event-graph connecting a non-reference event with a reference event.

Consider, for example, the CEG shown in Figure 4.4 and the state diagram shown in Figure 3.3. The timing equation between \( DATA_{S1} (e_1) \) and \( ADDR_{S1} (e_2) \) is established as follows. First note that \( e_i \),

\(^2\alpha \) and \( \beta \) denote a concatenation of a sequence of states.
the reference event for $DATA_{41}$ is $CLK_{73}$, and is associated with bus state $S4$. The reference event for $ADDR_{31}$, $e_j$, is $CLK_{71}$, and is associated with bus state $S0$. The transaction path from $S0$ to $S4$ is $S0$-$S1$-$S2$-$S3$ with a loop at $S3$. The equation $t(e_j/e_i)$ is therefore $2 \times T3 + (2 + n_1) \times T2$. The equation $t(e_j/e_i)$ is $-T27$ and the equation $-t(e_j/e_i)$ is $-T6$. Therefore, the equation $t(e_j/e_i)$ is $2 \times T3 + (2 + n_1) \times T2 = T27 - T6$.

7. Summary

In this paper, we have discussed a technique for generating symbolic timing equations from a graphical specification of interface behavior. This technique employs a scheme for the representation of temporal and sequencing information using causal-event-graphs and state-transition graphs. Some of the important characteristics of a CEG are that it has a single component, and either a single reference event, or as many reference events as sequencing events with out-degree greater than one.

We have outlined algorithms used to traverse a CEG to establish timing equations of non-reference events relative to a reference event, and to determine state-transition times. The resolve_reference_event algorithm handles the situation wherein a non-reference event has multiple reference events. The state-transition-graph-traversal algorithm determines all paths to a state and loops in a state diagram. It, in effect, establishes the temporal relation of reference events with regard to the start of a bus cycle. Finally, we have discussed an algorithm that determines the timing equation between arbitrary events. The symbolic-timing-equation generation tool is functional and uses components of the SpecITP [10] suite of tools (Xwave and Xstate) for the specification of interface behavior.

References