

# Automatic Synthesis of Gate-Level Timed Circuits with Choice\*

Chris J. Myers, Tomas G. Rokicki<sup>†</sup>, Teresa H.-Y. Meng  
Stanford University  
Stanford, CA 94305

## Abstract

*This paper presents a CAD tool for the automatic synthesis of gate-level timed circuits from general specifications to basic gates such as AND gates, OR gates, and C-elements. Timed circuits are a class of asynchronous circuits that incorporate explicit timing information in the specification which is used throughout the synthesis procedure to optimize the design. Our procedure begins with a textual specification capable of specifying conditional operation, or choice. This specification is systematically transformed to a graphical representation which can be analyzed using an exact and efficient timing analysis algorithm to find the reachable state space. From this state space, a timed circuit that is hazard-free at the gate-level is derived, facilitating the use of semi-custom components, such as standard-cells and gate-arrays. Because timing information is used to guide the synthesis to reduce circuit complexity while guaranteeing correct operation, the resulting timed circuit implementations are up to 40 percent smaller and 50 percent faster than those produced using other design methodologies.*

## 1: Introduction

In recent years, there has been a resurgence of interest in the design of *asynchronous circuits* due to their ability to eliminate clock skew problems, achieve average case performance, adapt to processing and environmental variations, and provide component modularity. Asynchronous circuits can also lower system power requirements by reducing synchronization power, automatically powering down unused components, removing spurious transitions, and easily adjusting to a dynamic power supply. While asynchronous designs have long been used in interface circuits, they are now being considered for the design of low-power embedded controllers and portable devices due to their low-power advantages.

Traditional academic asynchronous design methodologies use unbounded delay assumptions, resulting in circuits that are verifiably correct, but sacrifice timing for simplicity, leading to unnecessarily conservative designs. In industry, however, timing is critical to reduce both chip area and circuit delay. Due to the lack of formal methods to handle timing information correctly, circuits with timing constraints usually require extensive simulation to gain confidence in the design. Our research bridges this gap by introducing *timed circuits* in which explicit timing information is incorporated into the specification and utilized throughout the design procedure to optimize the implementation. Timed circuits can be significantly smaller and faster than those produced using traditional methods, and they

---

\*This research is supported by an NSF Fellowship and a research grant by ARPA.

<sup>†</sup>Currently at Hewlett-Packard Laboratories, Palo Alto, CA 94306.

are more reliable than those produced using ad hoc methods. The specification of timing constraints also facilitates a natural interaction between synchronous and asynchronous circuits.

Timing considerations, however, often introduce an additional exponential factor of complexity into the design procedure. As a result, timing analysis has hitherto either been avoided [13, 7, 14], simplified [3, 20], or considered only after synthesis [12]. This paper develops a synthesis procedure that utilizes timing analysis throughout to produce efficient gate-level timed circuits.

In our previous work, an efficient timing analysis algorithm is developed and applied to incorporate timing considerations into the synthesis of timed circuits [16]. This procedure, however, is not without its limitations. First, since the timing analysis is limited to only deterministic specifications, our synthesis procedure could only be applied to a limited class of circuits. Second, our timed circuit implementations require complex-gates, making it difficult to use semi-custom components, such as standard-cells and gate-arrays.

This paper presents a more general and widely applicable synthesis procedure. First, we expand the class of specifications to allow conditional operation, or choice, by applying systematic transformations to the specification to obtain a representation which can be analyzed by an exact and efficient timing analysis algorithm. Second, we facilitate the mapping of our implementations to semi-custom components by adding constraints to our synthesis procedure, thereby, producing hazard-free timed circuits using only basic gates such as AND gates, OR gates, and C-elements. This procedure has been fully automated in a CAD tool and applied to several examples, resulting in gate-level timed circuit implementations which are up to 40 percent smaller and 50 percent faster than designs using other methodologies.

In section 2, we describe the initial textual specification language and the graphical representations which are used for timing analysis and synthesis. Section 3 motivates the technique which is used by our synthesis procedure to obtain a gate-level implementation and describes it at a theoretical level. Section 4 explains the complete synthesis procedure in detail from a textual specification to a gate-level timed circuit implementation. Section 5 gives our experimental results, and conclusions are given in section 6.

## 2: Timed specifications

Many approaches could be taken to specify timed circuits including using languages such as *communicating sequential processes* (CSP) [13] or graphs such as *signal transition graphs* (STG) [7]. Graphs are conducive to automated timing analysis and synthesis algorithms, but they are cumbersome for specifying a large system. Languages, however, allow large designs to be specified clearly and concisely. For these reasons, we chose to use a language, *timed handshaking expansions* (THSE), as the initial specification which is then compiled to a graphical representation, an *orbital net* [18], for timing analysis. The timing analysis algorithm produces another graphical representation, a *state graph* (SG), used during circuit synthesis.

### 2.1: Timed handshaking expansions

THSE are based on Martin's *handshaking expansions* and are easily derivable from a CSP specification using a method similar to Martin's [13]. A specification using THSE is composed of two parts: a set of signal declarations and a set of *processes* executing in parallel. Each declaration consists of a type (either *input* or *output*), a signal name, an

initial value (either *true* or *false*), and delays associated with transitions on the signal. A delay is given in the form:  $\langle l_r, u_r; l_f, u_f \rangle$  where  $l_r$  and  $u_r$  are the lower and upper bounds on a rising transition and  $l_f$  and  $u_f$  are the lower and upper bounds on a falling transition. If the fall times are not specified, they are assumed to be equal to the rise times. Each process is a set of commands repeated forever (denoted  $*[ C ]$ ). These commands can be executed either in sequence (denoted  $C_1 ; C_2$ ) or in parallel (denoted  $C_1 \parallel C_2$ ). Each basic command is either an *event* or a *wait*. An event specifies when the process *causes* the occurrence of either a rising transition (denoted  $s \uparrow$ ) or a falling transition (denoted  $s \downarrow$ ) on a signal  $s$ . A wait, on the other hand, specifies when the process must *stall* for a certain set of events (denoted  $[ event\_list ]$ ) caused by some other process(es). The events in the set can be composed *conjunctively* (denoted  $e_1 \wedge \dots \wedge e_n$ ) to specify that the process waits until it has seen all the events in the set. The events can also be composed *disjunctively* (denoted  $e_1 \vee \dots \vee e_n$ ) to specify that the process waits until it has seen exactly one event in the set. Since the semantics of the orbital net representation used in our timing analysis is event-based, the semantics of our waits are based on events on signals rather than on values of signals. This change is made explicit in the language by using lists of events rather than predicates on signal values in wait commands.

Within a process, a choice of behavior is specified with a set of *guarded commands* (denoted  $[ G_1 \rightarrow C_1 \mid \dots \mid G_n \rightarrow C_n ]$ ). Each guarded command is composed of two parts: a guard  $G_i$ , which is either an event or a wait, and a set of commands  $C_i$ . If the guards in a set of guarded commands are waits, the commands associated with the first wait to be satisfied are executed. If the guards are events, one event is nondeterministically chosen resulting in the occurrence of the event followed by the execution of the commands that it guards. Since we allow choice but do not allow arbitration, an event as a guard must be on a signal of type *input*. If an arbiter is needed it can be added as a special environment process. A guarded command may also loop (denoted  $G_i \rightarrow C_i; *$ ) [6]. If a guarded command that loops is selected, then after the set of commands is executed, control is returned to the beginning of the guarded command. This looping continues until a guarded command that does not loop is selected.

The CSP specification for a port selector (SEL) is shown in Figure 1(a). The CSP specification dictates the ordering of communications on channels, but many different THSE using the signal wires shown in Figure 1(b) could implement the communications. Part of one possible THSE is shown in Figure 2 including a few declarations, the process for the SEL being designed, and the environment process which makes the choice of which output port to use. The basic operation of the SEL is as follows. First, the SEL waits until it gets a request for a data transfer (i.e.,  $xfer_i \uparrow$ ), then concurrently issues requests for the selection of an output port (i.e.,  $sel_o \uparrow$ ) and for the data to be transferred (i.e.,  $data_o \uparrow$ ). After the SEL receives the port selection (i.e.,  $sel1_i \uparrow$  or  $sel2_i \uparrow$ ) and acknowledgement that the data is ready (i.e.,  $data_i \uparrow$ ), it initiates the transfer of the data onto the selected output port (i.e.,  $out1_o \uparrow$  or  $out2_o \uparrow$ ). After the SEL receives acknowledgement that the selected port received the data (i.e.,  $out1_i \uparrow$  or  $out2_i \uparrow$ ), it acknowledges the completion of the data transfer (i.e.,  $xfer_o \uparrow$ ). The rest of the signals deal with resetting the four-phase handshake used to implement the communications.

## 2.2: Orbital nets

The THSE is transformed to an orbital net representation which is essentially a labeled safe Petri net extended to allow simultaneous actions and timing requirements [18]. These nets can accurately model general timed circuit behavior including choice while still being analyzable with an exact and efficient timing analysis algorithm.

**Figure 1. (a) CSP specification and (b) block diagram for a port selector (SEL).**

```

input  sel1i = {false, ⟨40, 260; 2, 40⟩};
output selo = {false, ⟨0, 20⟩};
etc.
* [ [xferi ↑]; (([datai ↓]; datao ↑) || ([sel1i ↓ ∨ sel2i ↓]; selo ↑)); [datai ↑];
  [ [sel1i ↑ ∧ out1i ↓] → out1o ↑; selo ↓; [out1i ↑]; (xfero ↑ || datao ↓); out1o ↓; [xferi ↓];
  xfero ↓
  | [sel2i ↑ ∧ out2i ↓] → out2o ↑; selo ↓; [out2i ↑]; (xfero ↑ || datao ↓); out2o ↓; [xferi ↓];
  xfero ↓
  ] ]
||
* [ [selo ↑]; [sel1i ↑ → [selo ↓]; sel1i ↓ | sel2i ↑ → [selo ↓]; sel2i ↓ ] ]
|| etc.

```

**Figure 2. Part of a THSE for the SEL.**

An orbital net is modeled by the tuple  $\langle A, P, T, F, M_0, R, L \rangle$  where  $A$  is the set of atomic actions,  $P$  is the set of places,  $T$  is the set of transitions,  $F \subseteq (P \times T) \cup (T \times P)$  is the set of edges,  $M_0 \subseteq P$  is the initial marking,  $R$  is an assignment of timing requirements to places, and  $L$  is a function which labels transitions with sets of simultaneous actions. A *marking* is a subset of the places. For a place  $p \in P$ , the *preset* of  $p$  (denoted  $\bullet p$ ) is the set of transitions connected to  $p$  (i.e.,  $\bullet p = \{t \in T \mid (t, p) \in F\}$ ), and the *postset* of  $p$  (denoted  $p\bullet$ ) is the set of transitions to which  $p$  is connected (i.e.,  $p\bullet = \{t \in T \mid (p, t) \in F\}$ ). For a transition  $t \in T$ , the presets and postsets are similarly defined (i.e.,  $\bullet t = \{p \in P \mid (p, t) \in F\}$  and  $t\bullet = \{p \in P \mid (t, p) \in F\}$ ).

**Timing requirements:** Timing in an orbital net is associated with a place as a timing requirement consisting of a lower bound, an upper bound, and a type (denoted  $\langle l, u \rangle$  type). There are two types of timing requirements: *behavior* ( $b$ ) and *constraint* ( $c$ ). Behavior timing requirements are used to specify guaranteed timing behavior. Constraint places, on the other hand, are used to specify desired timing behavior, and they do not affect the actual timing behavior. If the timing requirement on a place is omitted, it is assumed to be  $\langle 0, \infty \rangle c$ .

When there is a single behavior place  $p$  in the preset of a transition, regardless of the interpretation, the time of occurrence of the transition in the postset of  $p$  (denoted  $t(p\bullet)$ ) is always greater than the time of occurrence of any transition in the preset of the place (denoted  $t(\bullet p)$ ) by at least the lower bound of the timing requirement on  $p$ , and the difference is always less than the upper bound. If, on the other hand, there are multiple behavior places in the preset of a transition, there are four different ways the specified behavior can be interpreted [20]. The first, or *type 1*, says that for all behavior places  $p$ ,  $t(p\bullet) - t(\bullet p)$  must

exceed the lower bound but must not exceed the upper bound (this is the type used by our constraint places). If no possible timing behavior satisfy these constraints, the specification is inconsistent. The second, or *type 2*, says that for all behavior places,  $t(p\bullet) - t(\bullet p)$  must exceed the lower bound and for at least one behavior place,  $t(p\bullet) - t(\bullet p)$  must not exceed the upper bound. This is the type usually associated with circuit behavior, so it is the type we associate with our behavior places. *Types 3* and *4* are duals in which only a single lower bound needs to be reached (i.e., an OR relationship). These two types are not considered as they do not correspond with the conjunctive nature of the Petri-net model.

Our timing analysis algorithm relies on the fact that each behavior place represents a single nondeterministic choice of delay that cannot be affected by external state or other behavior places. When there are multiple behavior places in the preset of a transition, the type 2 semantics allows the delay between the transitions in the preset and postset of a behavior place to exceed its upper bound if the transition in the postset is being constrained by another behavior place. Therefore, our algorithm requires specifications to include at most a single behavior place in the preset of each transition. Fortunately, our original orbital net specification can always be transformed, as described later, into one which satisfies this single behavior place requirement.

**Operational semantics:** The behavior specified by an orbital net that satisfies the single behavior place requirement is defined with an operational semantics composed of two types of operations: advancement of time and firing of transitions. In an orbital net, an *untimed state* is a marking of the net. A *timed state* is an untimed state with a time-valued *clock* associated with each marked place. Each clock advances with time and denotes how long the place has been marked. Time is advanced by uniformly increasing these clocks by an amount  $\tau$  which is less than or equal to *max-advance* for a given marking. The function *max-advance* is defined as the minimum difference over all marked behavior places between the upper bound of the timing requirement on the place and its clock, or  $\infty$  if there are no marked behavior places. This upper limit on time advancement maintains the clocks for all behavior places below the maximum allowed by their range.

In an orbital net, a transition is *untimed-enabled* if all places in its preset are marked. A transition is *timed-enabled* when it is untimed-enabled and if there is a behavior place in its preset, this place's clock is greater than the lower bound of the timing requirement on the place. Any timed-enabled transition can be fired instantaneously, and any number of transitions can be fired without time advancing. A transition is fired by removing the marking in the places in its preset and discarding the clocks. The places in the postset of the fired transition are then marked, and all newly marked places are assigned a clock initialized to zero.

### 2.3: State graphs

Our timing analysis algorithm operates on the orbital net representation to find the reachable state space represented as a state graph. A state graph (SG) is a graph in which its vertices are untimed states and its edges are possible *state transitions*.

A state graph is modeled by the tuple  $\langle I, O, \Phi, \Gamma \rangle$  where  $I$  is the set of input signals,  $O$  is the set of output signals,  $\Phi$  is the set of states, and  $\Gamma \subseteq \Phi \times \Phi$  is the set of edges. For each untimed state  $s$ , there is a corresponding labeling function  $s : I \cup O \rightarrow \{0, R, 1, F\}$

which returns the value of each signal and whether it is untimed-enabled, i.e.,

$$s(u) \equiv \begin{cases} 0 & \text{if } u \text{ is stable low in } s \\ R & \text{if } u \text{ is untimed-enabled to rise in } s \\ 1 & \text{if } u \text{ is stable high in } s \\ F & \text{if } u \text{ is untimed-enabled to fall in } s. \end{cases}$$

It is useful to also define a function *val* which strips the excitation information, i.e.,

$$val(x) \equiv \begin{cases} 0 & \text{if } x = 0 \text{ or } x = R \\ 1 & \text{if } x = 1 \text{ or } x = F. \end{cases}$$

Traditional definitions of state labeling functions have not included the enabling of signals as it can usually be inferred from the set of state transitions. In timed circuits, however, it is possible that a signal is untimed-enabled but not timed-enabled in a given state. In this case, there would be no state transition out of that state in which that signal fired, and thus, it would not be possible to infer from the state graph that the signal is untimed-enabled.

A state graph is defined to be well-formed if it is strongly connected and for any state transition  $(s, s')$  in  $\Gamma$ , the value of exactly one enabled signal in  $s$  changes to a new value in  $s'$ . The signal  $v$  that differs in value in the state transition  $(s, s')$  is denoted as follows:  $s \xrightarrow{v} s'$ . Our synthesis procedure also requires that the state graph be *complete state coded*, defined such that if for any two states in which all signals have the same value, any output signal untimed-enabled in one state is also untimed-enabled in the other.

### 3: Gate-level timed circuit implementations

After obtaining a state graph, there are several different approaches that could be used to obtain a gate-level timed circuit implementation. The first approach is to use a traditional Boolean minimization technique. We demonstrate, however, that when mapping the resulting implementation to basic gates, it may result in a hazardous implementation. Another approach is to split the design of the rising and falling transitions to obtain a *generalized C-element* implementation [13] and decompose it to basic gates. This technique alleviates some of the hazard problems, but we demonstrate that it may still be hazardous when mapped to basic gates. The approach that we take is to decompose the design to a set of rising and falling regions. Each region is implemented using a single AND gate, or *cube*, which must satisfy certain constraints. When all the regions are merged, the resulting implementation is guaranteed to be a hazard-free gate-level timed circuit.

#### 3.1: Boolean minimization technique

To apply a traditional Boolean minimization technique, the state space is partitioned into an *on-set*, an *off-set*, and a *don't-care-set*. Then, a Boolean minimization program, such as **espresso** [4] can be used to find the optimal sum-of-products representation. For our designs, a minimization problem would be setup for each output signal  $u$  with the on-set containing each state  $s$  in which the signal is enabled to rise or is stable high (i.e.,  $s(u) = R$  or  $s(u) = 1$ ), the off-set containing each state  $s$  in which the signal is enabled to fall or is stable low (i.e.,  $s(u) = F$  or  $s(u) = 0$ ), and the don't-care-set containing all unreachable states (i.e.,  $\beta^{I \cup O} - \Phi$ ).

Applying this technique to the signal  $out2_o$  from the SEL results in the Boolean equation:

$$out2_o = (data_i \wedge sel2_i \wedge \neg out2_i) \vee (data_o \wedge out2_o) \vee (\neg xfer_o \wedge out2_o).$$

In order to guarantee correctness, Chu [7] and Meng [14] assumed that the logic equation for each output signal could be implemented directly with a single complex *atomic* gate. In other words, each signal is built with an infinitely fast function block with all its delay concentrated at its output. Unfortunately, if the equation is mapped to basic gates and the delays of these gates are considered individually, the implementation may be hazardous. For example, the equation for  $out2_o$  could be implemented directly as a sum-of-products as shown in Figure 3. If the 3-input AND and OR gates (gates 1 and 4) are assumed to have a delay of  $\langle 2, 5 \rangle$  while the 2-input AND gates (gates 2 and 3) have a delay of  $\langle 2, 3 \rangle$ , this implementation is hazard-free. However, if the upper bound of the delay on the 2-input AND gates increases to 4 or more time units, this circuit is now hazardous. The segment of the state graph to the left illustrates a sequence of transitions which cause a hazard. Essentially, after gate 1 has caused  $out2_o$  to rise, it has the potential of being shut off again before gates 2 or 3 can come on to hold the state.

**Figure 3. A hazardous sum-of-products implementation of  $out2_o$  from the SEL.**

In order to solve this problem, Lavagno [12] first mapped the logic equations ignoring hazards using standard synchronous techniques, then added delay elements where necessary to remove any potential hazards. This technique, however, not only adds additional overhead in terms of area and delay, but the resulting circuits may not be very reliable due to the difficulty in designing delay elements with accurate timing.

**3.2: Generalized C-element technique**

Another implementation strategy originally proposed by Martin [13] is to use generalized C-elements as the basic building blocks. This is also the technique used in our earlier work [16]. In this technique, the implementation of the set and reset of a signal are decoupled. The basic structure is depicted in Figure 4(a) in which the upper sum-of-products represents the logic for the set, the lower sum-of-products represents the logic for the reset, and the result is merged with a C-element. This can be implemented directly in CMOS as a single compact gate with weak-feedback as shown in Figure 4(b) or as a fully-static gate as shown in Figure 4(c).

Using a procedure similar to the one described in [16], we obtain a generalized C-element implementation for the signal  $out2_o$  shown in Figure 5. While this could be implemented

**Figure 4. (a) The generalized C-element configuration, (b) a weak-feedback CMOS implementation, and (c) a fully-static CMOS implementation.**

with a single generalized C-element, a gate-level implementation would be composed of a 3-input AND gate, a 2-input AND gate, and a C-element. Although this implementation no longer has the hazard associated with the SG fragment in Figure 3, it still has a hazard illustrated with the state graph shown to the left in which the reset AND gate glitches while the output is stable low. For the specified delays, it can be shown that the hazard does not propagate to the output, but given appropriate delays this hazard may propagate [1]. To address this problem, after a generalized C-element implementation is produced and decomposed to basic gates, the design could be back-annotated with delays from the gate library, and the circuit could be verified. While this may often work, it is not clear what to do in the cases in which a hazard does exist. Also, a hazard is a spurious transition which wastes power and does no useful work. In a power efficient implementation, it is desirable to have logic which is hazard-free both internally and externally.

1

**Figure 5. A hazardous gate-level implementation of  $out2_o$  from the SEL.**

### 3.3: Standard C-implementation technique

To avoid the hazard concerns discussed above, our approach obtains a gate-level implementation by first decomposing the design into a set of rising and falling regions which are each implemented using a single cube. While the general structure is similar to the generalized C-element structure shown in Figure 4, each cube in the set or reset block must satisfy certain constraints to guarantee that the merged implementation is a gate-level hazard-free circuit. The approach is conservative in that timing analysis may show that the decomposed generalized C-element implementation is sufficient, but the overhead required tends to be small to get a safe implementation that is free of internal hazards.

**Excitation regions and quiescent states:** In order to obtain our gate-level implementation, the SG is decomposed for each output signal into a collection of *excitation regions*. An excitation region for the output signal  $u$  is a maximally connected set of states in which the signal is enabled to change to a given value (i.e.,  $s(u) = R$  or  $s(u) = F$ ). If the signal is rising in the region (i.e.,  $s(u) = R$ ), it is called a *set region*, otherwise the region is called a *reset region*. The excitation regions for a signal  $u$  are indexed with the variable  $k$  and the  $k^{\text{th}}$  excitation region of signal  $u$  is denoted  $ER(u, k)$ . Typically, different excitation regions correspond to different output signal transitions in a higher-level specification. For example, there are two set regions for the signal  $xfer_o$  in the SEL which correspond to the two instances of  $xfer_o \uparrow$  in the THSE in Figure 2.

For each excitation region, there is an associated set of stable, or *quiescent*, states  $QS(u, k)$ . For a set region  $ER(u, k)$ , the quiescent states are those states where the signal is stable high (i.e.,  $QS(u, k) = \{s \in \Phi \mid s(u) = 1\}$ ), and for a reset region, they are those states where the signal is stable low (i.e.,  $QS(u, k) = \{s \in \Phi \mid s(u) = 0\}$ ).

**Correct covers:** In our gate-level implementation, each excitation region is implemented with a single cube corresponding to a *correct cover* of the excitation region. The *cover* of an excitation region  $C(u, k)$  is a set of states for which the corresponding cube in the implementation evaluates to one. A cover is a correct cover if it satisfies two conditions. First, it must satisfy the *covering constraint* which says that the reachable states in the cover must include the entire excitation region but must not include any states outside of the union of the excitation region and associated quiescent states, i.e.,

$$ER(u, k) \subseteq [C(u, k) \cap \Phi] \subseteq [ER(u, k) \cup QS(u, k)].$$

Second, it must satisfy the *entrance constraint* which says that a correct cover must only be entered through excitation region states, i.e.,

$$[(s, s') \in \Gamma \wedge s \notin C(u, k) \wedge s' \in C(u, k)] \Rightarrow s' \in ER(u, k).$$

The definition of correct covers is based on the definition given for speed-independent circuits in [2]. This definition differs slightly from the one in [2] in that  $QS(u, k)$  does not need to be a maximally connected set of states, but it is easy to show this condition is made redundant by the entrance constraint. Extending the proof for the speed-independent case, it can be shown that these conditions ensure when the specified timing assumptions are met that the resulting gate-level timed circuit implementation is hazard-free.

## 4: Synthesis procedure

The synthesis procedure begins with a THSE which is compiled to an orbital net representation. This net is then systematically transformed to one which can be efficiently analyzed. A timing analysis algorithm based on *geometric regions* and *partial orders* finds the set of reachable untimed states. The resulting state graph is decomposed into excitation regions for each output signal represented using approximations of the state space. A covering problem is setup and solved to find a correct cover for each region to produce a gate-level timed circuit implementation using only basic gates such as AND gates, OR gates, and C-elements.

### 4.1: Translation from a timed handshaking expansion to an orbital net

The procedure to translate a THSE to an orbital net first transforms each concurrent process individually, then composes the resulting orbital nets to obtain the complete orbital net representation. While this process can be formalized, only an intuitive description is given here as it is beyond the scope of this paper.

The procedure to transform a process to an orbital net is initially described without choice. First, for each signal in the specification a rising and falling transition on the signal are added to the set of actions. Next, a transition is added to the net for each occurrence of an action in either an event or a wait, and the transition is labeled with the corresponding action. For each event in the specification, the procedure adds a behavior place and corresponding edges to the corresponding transition from each action in an event or wait that directly precedes it with a timing requirement taken from the declarations. The procedure also adds a constraint place with timing requirement  $\langle 0, \infty \rangle c$  and corresponding edges to the preset for any other event which is separated only by waits. To handle the outer loop, the procedure considers the last events as preceding the first events with the difference that each place that is added is initially marked.

To address choice, care must be taken to determine whether a new place is needed or if a previously added place is to be shared. For example, for the selection process from the SEL, the events  $sel1_i \uparrow$  and  $sel2_i \uparrow$  must share the place in the postset of the event  $sel_o \uparrow$ . The complete orbital net for the selection process is shown in Figure 6(a).

When this net is composed with the orbital net for the SEL process, different occurrences of actions in one process are matched with their corresponding occurrences in the other process by keeping track of their order. Part of the orbital net after composition with the SEL process is shown in Figure 6(b). Note that as an optimization a simple analysis of the graph determines that all the constraint places in Figure 6(a) can be removed since there are paths through behavior places that make them redundant.

### 4.2: Satisfying the single behavior place requirement

The next step of our synthesis procedure transforms the orbital net to one which satisfies the single behavior place requirement. To accomplish this, consider a fragment of an orbital net that has two behavior places in the preset of a transition shown in Figure 7(a). The desired timing behavior can be depicted graphically as shown in Figure 7(b). This net can be transformed to the one shown in Figure 8(a) which satisfies the single behavior place requirement. Basically, the idea behind this net transformation is that a path through the net is created for each possible ordering of the transitions in the preset. This has the effect that each transition in the preset is given the chance to be the last one preventing

**Figure 6. (a) Orbital net for the selection process from the SEL; (b) part of the orbital net after composition with the SEL process.**

the transitions in the postset from occurring. The timing behavior of  $c_0$  and  $c_1$  are shown graphically in Figure 8(b) and (c), respectively. The behavior of these two together is exactly the desired timing behavior of  $c$ . For  $n$  behavior places, the net is transformed to model the  $n!$  possible orderings of the  $n$  enabling events. While this transformation can lead to a substantial blowup in the net size, we have found that the value of  $n$  tends to be quite small in practical examples.

**Figure 7. (a) Fragment of the orbital net that violates the single behavior place requirement; (b) graphical representation of the desired timing behavior.**

The transformation is more complicated in the case that one of the behavior places in the preset has multiple transitions in its postset. Consider a fragment of the orbital net from the SEL shown in Figure 9(a). In this net, the behavior place in the postset of  $data_i \uparrow$  is shared by the transitions  $out1_o \uparrow$  and  $out2_o \uparrow$ . In other words, if  $out2_o \uparrow$  occurs, the marking is removed before it can contribute to the firing of  $out1_o \uparrow$ . In order to model this, the net is first transformed using the procedure described above for  $out1_o \uparrow$  and  $out2_o \uparrow$ . Then, transitions are added to the part associated with  $out1_o \uparrow$  on  $out2_o \uparrow$  that reset the marking, and similarly transitions are added to the part associated with  $out2_o \uparrow$  on  $out1_o \uparrow$ .

**Figure 8. (a) Orbital net that satisfies the single behavior place requirement; graphical representation of the timing behavior of  $c_0$  (b) and  $c_1$  (c).**

A portion of the transformed net illustrating this is shown in Figure 9(b).

**Figure 9. (a) Fragment of orbital net with a behavior place that has multiple transitions in its postset; (b) part of the transformed orbital net that satisfies the single behavior place requirement (note “a” and “b” are shorthands for connections).**

### 4.3: Finding a state graph

Once an orbital net representation has been obtained that satisfies the single behavior place requirement, a technique called *partial order timing* is used to find the set of reachable states. This technique is briefly described in this subsection, but for a more complete description see [19]. Partial order timing represents sets of timed states as geometric regions, and it improves upon standard geometric timing techniques by making use of partial orders to separate concurrency from causality in the generation of these regions.

Using geometric regions to represent the timed state space was originally proposed by Dill [8]. These regions of timed states are represented by lower and upper bounds on specific clock values and on the differences between pairs of specific clocks. The set of such constraints is usually represented by a matrix  $A$ , where the constraints on clocks  $\{clk_1, \dots, clk_n\}$  are of the form  $clk_i - clk_j \leq a_{ji}$ . A fictitious clock  $clk_0$  that is always exactly zero is introduced so that lower and upper limits on a particular clock can be represented in the same form [8]. For any convex region that can be represented by such

a matrix, there are many matrices that represent the same convex region, but a unique matrix can be obtained by *canonicalization* [8] using Floyd’s algorithm.

An *orbital net process* is an acyclic, choice-free net created from an orbital net and a sequence of transitions, and it is used to reflect a partial order on the transitions in the sequence. In other words, the process explicitly represents the causality and concurrency inherent in the sequence. A particular process corresponds to many different sequences that differ only in the interleavings of concurrent transitions; every such sequence fires the same set of transitions and leads to the same final untimed state.

In order to determine the reachable state space for our system, the partial order timing algorithm uses a depth-first search to incrementally construct for each sequence of transitions examined its corresponding orbital net process. For each process, the algorithm also incrementally calculates a constraint matrix for the geometric region corresponding to the process. This constraint matrix is constructed by first adding the lower and upper bounds of the clocks. Next, for each constraint place  $p$ , a constraint of the form  $t(\bullet p) \leq t(p\bullet)$  is introduced. For each behavior place  $p$  in the resulting process with a timing requirement of  $\langle l, u \rangle b$ , two constraints are introduced. The first reflects the minimum separation,  $t(\bullet p) - t(p\bullet) \leq -l$ . The second reflects the maximum separation,  $t(p\bullet) - t(\bullet p) \leq u$ . After canonicalizing this constraint matrix, the resulting geometric region represents the full set of reachable states for this process.

While obtaining a state graph, there may be violations of some constraint timing requirements. For our nets, this typically occurs if an ordering assumed at the specification level is not guaranteed by the specified behavior. If there are any constraint violations, the specification is automatically modified to solve the problem. If all transitions in the postset of the violating constraint place are output signals, our procedure transforms it to a behavior place with a default timing requirement. If any transition in the postset of the violating constraint place is an input, the procedure slows down an earlier transition by adding a behavior place between the transitions in the preset and the first output transition that precedes the transitions in the postset.

For the SEL, there are four constraint violations on the places between  $xfer_i \uparrow$  and  $data_o \uparrow$ ,  $xfer_i \uparrow$  and  $sel_o \uparrow$ ,  $data_i \uparrow$  and  $out1_o \uparrow$ , and  $data_i \uparrow$  and  $out2_o \uparrow$ . All of these violations have output signals in the postset, so they are simply changed to behavior places with a default timing requirement of  $\langle 0, 20 \rangle b$ . The final state graph obtained for the SEL contains 53 states. If a state graph is generated ignoring all the timing information, it contains 256 states.

#### 4.4: Finding enabled cubes and trigger cubes

Since each region is implemented with a single cube, to obtain a hazard-free implementation, all literals in the cube must correspond to signals that are *stable*, i.e., constant throughout the excitation region. Otherwise, the cube would not cover all the excitation region states. When a single-cube cover exists, an excitation region  $ER(u, k)$  can be sufficiently approximated using a cube called an *enabled cube*, denoted  $EC(u, k)$ , defined on each signal  $v$  as follows:

$$EC(u, k)(v) \equiv \begin{cases} 0 & \text{if } \forall s \in ER(u, k) [val(s(v)) = 0] \\ 1 & \text{if } \forall s \in ER(u, k) [val(s(v)) = 1] \\ X & \text{otherwise} \end{cases}$$

If a signal has a value of 0 or 1 in the enabled cube, it can be used in the cube implementing the region. A cube, such as the enabled cube, implicitly represents a set of states in the

obvious way. The set of states represented by the enabled cube is always a superset of the set of excitation region states (i.e.,  $EC(u, k) \supseteq ER(u, k)$ ).

Each cube in the implementation is composed of *trigger signals* and *context signals*. For a given excitation region, a trigger signal is a signal whose firing can cause the circuit to enter the excitation region while any non-trigger signal which is stable in the excitation region can potentially be a context signal. The set of trigger signals for an excitation region  $ER(u, k)$  can also be represented with a cube called a *trigger cube*  $TC(u, v)$  defined as follows for each signal  $v$ :

$$TC(u, k)(v) \equiv \begin{cases} val(s'(v)) & \text{If } \exists s, s' [(s \xrightarrow{v} s') \wedge (s \notin ER(u, k)) \wedge (s' \in ER(u, k))] \\ X & \text{otherwise} \end{cases}$$

In order for our synthesis procedure to generate a circuit, the cover of each excitation region must contain all its trigger signals (i.e.,  $C(u, k) \subseteq TC(u, k)$ ). Since only stable signals can be included, a necessary condition for our algorithm to produce an implementation is that all trigger signals be stable (i.e.,  $EC(u, k) \subseteq TC(u, k)$ ). If a trigger signal is not stable, then we must either constrain concurrency [14], add state variables [11], or use a more general algorithm [2].

The enabled cubes and trigger cubes are easily found with a single pass through the state graph. Table 1 shows the enabled cubes and trigger cubes corresponding to all the excitation regions in the SEL.

**Table 1. Enabled cubes and trigger cubes for the SEL.**

$u, k$	<i>set/reset</i>	$EC(u, k)$	$TC(u, k)$
$xfer_o, 0$	<i>set</i>	11X0100X010	XXXX1XXXXXX
$xfer_o, 1$	<i>set</i>	110X010X001	XXXXX1XXXXX
$xfer_o, 2$	<i>reset</i>	0X00XX10000	0XXXXXXXXXX
$data_o, 0$	<i>set</i>	10000000X00	1XXXXXXXXXX
$data_o, 1$	<i>reset</i>	11X010X1010	XXXX1XXXXXX
$data_o, 2$	<i>reset</i>	110X01X1001	XXXXX1XXXXX
$sel_o, 0$	<i>set</i>	1000000X000	1XXXXXXXXXX
$sel_o, 1$	<i>reset</i>	11100001110	XXXXXXXXXX1X
$sel_o, 2$	<i>reset</i>	11010001101	XXXXXXXXXX1
$out1_o, 0$	<i>set</i>	11100001100	X11XXXXXXXX
$out1_o, 1$	<i>reset</i>	1XX01010010	XXXXXX10XXX
$out2_o, 0$	<i>set</i>	11010001100	X1X1XXXXXXXX
$out2_o, 1$	<i>reset</i>	1X0X0110001	XXXXXX10XXX

$\langle xfer_i, data_i, sel1_i, sel2_i, out1_i, out2_i, xfer_o, data_o, sel_o, out1_o, out2_o \rangle$

#### 4.5: Finding an optimal correct cover

Our procedure to find an optimal correct cover begins with a cube consisting only of the trigger signals (i.e.,  $C(u, k) = TC(u, k)$ ). If this cover contains no *conflicts*, i.e., states that violate either the covering or entrance constraint, we are done. This, however, is often not the case, and context signals must be added to the cube to remove any conflicting states. For each conflict detected, the procedure determines the choices of context signals which would exclude the conflicting state. Finding the smallest set of context signals to resolve

all conflicts is a covering problem. Due to the implication in the entrance constraint, inclusion of certain context signals may introduce additional conflicts which must be resolved. Therefore, the covering problem is *binate*.

To solve our binate covering problem, we create a *covering and closure (CC) table* [9] for each region. While other techniques exist to find binate covers such as those described in [10, 5], the CC table is simple and facilitates presentation. There is a row in the CC table for each context signal, and there is a column for each conflict and each conflict that could potentially arise from a context rule choice. An entry in the table contains a cross ( $\times$ ) if the context signal resolves the conflict. An entry in the table contains a dot ( $\circ$ ) if the inclusion of the context signal would require the conflict to be resolved.

To construct the table for a given excitation region  $ER(u, k)$ , the procedure first finds all states in the initial cover (i.e.,  $TC(u, k)$ ) which conflict with the covering constraint. In other words, a conflict exists in a state  $s$  in  $TC(u, k)$  if the signal  $u$  has the same value but is not enabled (i.e.,  $s(u) = 0$  for a set region or  $s(u) = 1$  for a reset region), is enabled in the opposite direction (i.e.,  $s(u) = F$  for a set region or  $s(u) = R$  for a reset region), or is enabled in the same direction but the state is not in the current excitation region (i.e.,  $s(u) = R$  for a set region or  $s(u) = F$  for a reset region and  $s \notin EC(u, k)$ ). If a conflict exists, the procedure adds a new column to the table with a cross in each row corresponding to a context signal  $v$  that would exclude the conflicting state (i.e.,  $EC(u, k)(v) = \neg val(s(v))$ ).

The next step in the table construction is to find all state transitions which conflict with the entrance constraint in the initial cover or may conflict due to a context signal choice. For any state transition  $s \xrightarrow{v} s'$ , this is possible when  $s$  is not in the excitation region (i.e.,  $s \notin EC(u, k)$ ),  $s'$  is a quiescent state (i.e.,  $s'(v) = 1$  for a set region and  $s'(v) = 0$  for a reset region),  $s'$  is in the initial cover (i.e.,  $s' \in TC(u, k)$ ), and  $v$  excludes  $s$  (i.e.,  $EC(u, k)(v) = \neg val(s(v))$ ). For each entrance conflict or potential entrance conflict detected, the procedure adds a new column to the table again with a cross in each row corresponding to a context signal that would exclude the conflicting state. If the signal  $v$  in the state transition is a context signal, the state  $s'$  only needs to be excluded if  $v$  is included in the cover. This implication is represented with a dot being placed in the row corresponding to the signal  $v$ .

If a conflict is detected for which there is no context signal to resolve it, the CC table construction fails. In this case, as with non-stable trigger signals, it is necessary to constrain concurrency, add state variables, or use a more general algorithm.

In a single pass through the state graph, all the CC tables can be constructed. When implementing  $(out2_o, 1)$  from the SEL, no covering conflicts are detected. This is not surprising since our complex-gate implementation of this region only contained the trigger signals  $xfer_o$  and  $\neg data_o$ . There are, however, entrance conflicts which are shown in the CC table in Table 2.

The last step is to find the smallest set of context signals to implement each excitation region by solving the CC tables that are constructed. The CC tables are solved using standard reduction rules [9]. After applying these reduction rules, the CC table for  $(out2_o, 1)$  is reduced to one containing a single conflict which can be solved using either  $out2_i$  or  $out2_o$  as a context signal. In this case, they are equivalent, and  $out2_i$  is arbitrarily selected. While in practice these reduction rules are often sufficient to solve the table, some tables may be *cyclic*. To solve a cyclic table, we use a branch and bound method.

For the SEL, we derive a gate-level timed circuit implementation with 27 literals shown in Figure 10(a). If all the timing information is ignored, we obtain a gate-level speed-independent circuit implementation with 44 literals shown in Figure 10(b). Besides being nearly 40 percent smaller, the timed circuit has reduced latency since it requires gates with

**Table 2. The CC table for the region  $(out2_o, 1)$  from the SEL.**

Signal	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$xfer_i$		×		×	×								×		
$sel_i$	×	○	×		○	×	×	○		○		○	×		○
$out1_i$	×	×	○	○		×	×	×	×	×	○		○	×	×
$out2_i$	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
$xfer_o$															
$data_o$															
$sel_o$															
$out1_o$	○					×	○	×	○					×	×
$out2_o$	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×

at most 3-inputs while the speed-independent circuit requires many large gates including one with 6-inputs.

**Figure 10. Gate-level (a) timed and (b) speed-independent circuits for the SEL.**

## 5: Experimental results

The synthesis procedure described in this paper has been fully automated within the CAD tool ATACS. We have applied this procedure to several examples and compared our results with designs produced using other methodologies including Beerel’s speed-independent method (SYN), Lavagno’s method which adds delay elements to remove hazards (SIS), and Yun’s burst-mode method (3D). In addition to the SEL described above, another design (SEL2) is given in the table in which the selection of the output port is performed using a single conditional signal rather than dual-rail encoding and three signal wires. In [16], due to limitations on the allowed specifications, only one of the six possible cycles of an asynchronous memory management unit (MMU) originally specified in [15] could be designed, but now the complete specification can be synthesized directly. The last two examples of a DRAM controller (DRAM) and the target-send burst-mode portion of a SCSI controller (TSBM) were originally specified using burst-mode state machines in [17]. The 3D specifi-

cations and results are taken from [21] assuming a  $0.3ns$  inverter delay in a  $0.8\mu m$  CMOS process.

The results are tabulated in Table 3. First, we compared the literal counts (Lit) for the gate-level circuits derived using the generalized C-element (gC) technique and our standard C-implementation techniques. Our results show only about a 10 percent increase in literal count for generating a safe implementation that has no internal hazards. For the first three examples, the timed implementations are compared with those produced by SYN and SIS in terms of area represented by transistor count and delay represented as ratio of fanout of four inverter delays. The timed implementations are about 40 percent smaller and faster than the those produced by SYN. Compared with SIS, the area gains are about the same, but the improvement in delay is now about 50 percent. The table also gives the number of reachable states ( $|\Phi|$ ) for timed and other untimed methods showing up to two orders of magnitude less states in the timed case. In fact, due to the large size of the state space for the MMU, SIS runs out of memory during synthesis. The last two examples are compared with the 3D method showing about a 30 percent improvement in area (comparing literal count) and delay. Finally, all implementations produced by ATACS are verified to be hazard-free at the gate-level using `orbits` [19].

**Table 3. Experimental results.**

Ex.	$ \Phi $	Timed				Other Design Methodologies						
		gC Lit	ATACS			$ \Phi $	SYN		SIS		3D	
		Lit	Lit	Area	Del		Area	Del	Area	Del	Lit	Del
SEL	53	25	27	104	5	256	160	7	158	11	n/a	n/a
SEL2	36	19	21	76	5	128	108	6.5	130	11.5	n/a	n/a
MMU	187	56	62	210	4.5	23,296	412	10	out of memory		n/a	n/a
DRAM	79	38	38	110	5.5	n/a	n/a	n/a	n/a	n/a	46	7
TSBM	113	32	33	140	4.5	n/a	n/a	n/a	n/a	n/a	58	7.5

## 6: Conclusion

This paper describes a methodology for the automatic synthesis of gate-level timed circuits with choice. Our synthesis procedure begins with a general specification capable of specifying sequencing, concurrency, and choice. The specification is systematically transformed to an orbital net representation which can be analyzed with an efficient timing analysis algorithm based on geometric regions and partial orders to obtain the reachable state space. Using a method based on approximations of these states, our procedure obtains a hazard-free gate-level timed circuit implementation. We have demonstrated the effectiveness of this synthesis procedure on several practical examples, and our results indicate that our timed circuit implementations are significantly smaller and faster than those produced by other design methodologies.

## Acknowledgments

We are especially grateful to Professor Peter Beerel of the University of Southern California for his invaluable comments on this work. We would also like to thank Professor Steve

Burns, Professor Gaetano Borriello, and Henrik Hulgaard of the University of Washington for providing many stimulating discussions about timing analysis. Finally, we greatly appreciate the comments on this work which we received from Professor Alain Martin and his graduate students at Caltech.

## References

- [1] P. A. Beerel and T. H.-Y. Meng. Semi-modularity and testability of speed-independent circuits. *INTEGRATION, the VLSI journal*, 13(3):301–322, September 1992.
- [2] P. A. Beerel, C. J. Myers, and T. H.-Y. Meng. “Automatic synthesis of gate-level speed-independent circuits”, November 1994. Submitted for publication in *IEEE Transactions on Computer-Aided Design*.
- [3] G. Borriello. *A New Specification Methodology and its Applications to Transducer Synthesis*. PhD thesis, University of California, Berkeley, 1988.
- [4] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic, 1984.
- [5] R. K. Brayton and F. Somenzi. An exact minimizer for Boolean relations. In *Proceedings IEEE 1989 ICCAD Digest of Technical Papers*, pages 316–19, 1989.
- [6] S. M. Burns. Automated compilation of concurrent programs into self-timed circuits. Technical Report Caltech-CS-TR-88-2, California Institute of Technology, 1987.
- [7] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications*. PhD thesis, Massachusetts Institute of Technology, 1987.
- [8] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the Workshop on Automatic Verification Methods for Finite-State Systems*, June 1989.
- [9] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IEEE Transactions on Electronic Computers*, pages 350–359, June 1965.
- [10] S. Jeong and F. Somenzi. A new algorithm for the binate covering problem and its application to the minimization of boolean relations. In *IEEE ICCAD Digest of Technical Papers*, pages 417–420, 1992.
- [11] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proc. ACM/IEEE Design Automation Conference*, 1994.
- [12] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, 1991.
- [13] A. J. Martin. Programming in VLSI: from communicating processes to delay-insensitive VLSI circuits. In C.A.R. Hoare, editor, *UT Year of Programming Institute on Concurrent Programming*. Addison-Wesley, 1990.
- [14] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messersmitt. Automatic synthesis of asynchronous circuits from high-level specifications. *IEEE Transactions on Computer-Aided Design*, 8(11):1185–1205, November 1989.
- [15] C. J. Myers and A. J. Martin. The design of an asynchronous memory management. Technical Report CS-TR-93-30, California Institute of Technology, 1993.
- [16] C. J. Myers and T. H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.
- [17] S. M. Nowick, K. Y. Yun, and D. L. Dill. Practical asynchronous controller design. In *International Conference on Computer Design, ICCD-1992*. IEEE Computer Society Press, 1992.
- [18] T. G. Rokicki. *Representing and Modeling Circuits*. PhD thesis, Stanford University, 1993.
- [19] T. G. Rokicki and C. J. Myers. Automatic verification of timed circuits. In *International Conference on Computer-Aided Verification*, pages 468–480. Springer-Verlag, 1994.
- [20] P. Vanbekbergen. *Synthesis of Asynchronous Controllers from Graph-Theoretic Specifications*. PhD thesis, Katholieke Universiteit Leuven, September 1993.
- [21] K. Y. Yun. *Synthesis of Asynchronous Controllers for Heterogeneous Systems*. PhD thesis, Stanford University, 1994.