Automatic Synthesis of Fast, Compact Self-Timed Control[1]

Kenneth S. Stevens

Department of Computer Science
University Of Calgary
Calgary, Alberta, Canada T2N 1N4

Abstract

An automated synthesis tool, called the *Most Excellent Asynchronous Tool, or MEAT* is presented. This tool has been used to specify and synthesize self-timed circuits for a fully self-timed 300,000 transistor communication co-processor. The specification is done with stylized state diagrams. This is a very compact and intuitive means to specify communication, concurrency, and synchronization necessary for control structures. Of primary importance to this project was the efficiency and simplicity of the implementation. The tool generates from the state description provably correct self-timed CMOS implementations with outstanding performance and compactness.

1 Introduction

Three major constraints – speed of operation, size, and design time must be considered with any computer architecture, be it a commercial product or a laboratory prototype. As integrated circuit technology improves, the amount of logic that can be placed on a VLSI circuit increases quadratically. Without improvement in design methodology, the design time of circuits will also increase quadratically. Synchronous design techniques are facing critical design and performance difficulties because clock signals are required to drive a quadratically increasing number of components. This problem is evident both in clock skew problems and the increased difficulty in tuning the performance of all the logic blocks to match the clock period.

Asynchronous circuits, although a viable approach, have generally been regarded difficult to design due to the requirement that all hazards be eliminated for them to work properly. Removing the hazards is viewed as an expensive process in both design time and circuit component count. An additional problem is the difficulty in specifying and describing asynchronous circuits in the absence of commercially available tools

Asynchronous systems are built hierarchically, and consist of two types of cells – leaf elements, and systems made by interconnecting leaf elements and/or other cells. The design of the elemental

[1]Most of the work was done while at Hewlett Packard Laboratories in Palo Alto, California.

leaf components in asynchronous designs may be more difficult and require more devices than a synchronous version. However, system level design is simplified because correct operation consists of assuring proper sequence of operation, without regard to time[13]. This ability to decompose a complex system into a hierarchical network of interconnected functional blocks permits us to automate the production of large systems. This automation has a great potential for fast design turnaround.

The ability to isolate and localize communication can also result in faster operation, less area, and simplified design. Because the elemental components are small, the task of removing hazards and producing correctly operating asynchronous components is manageable. Hence the speed of operation of asynchronous circuits has been demonstrated to be on par with that of their synchronous counterparts[5,9].

This paper presents a synthesis tool, called the *Most Excellent Asynchronous Tool,* or *MEAT*, that can greatly reduce design and implementation time while also generating compact self-timed circuits with excellent speed of operation. MEAT allows the designer to specify logical operation of asynchronous leaf components in a commonly used format. This specification is then automatically compiled into a provably correct implementation, freeing the designer from the details of asynchronous circuit stipulations. The process is fast enough that alternative design options can be freely examined. The main advantages of MEAT may be summarizes as follows:

1. Asynchronous circuits are specified for MEAT by a variant of mealy state machine. This specification is familiar and natural for any hardware designer. These specifications are a powerful way to encapsulate concurrency, communication, and synchronization in an accurate, intuitive and easy to understand form.

2. The automated synthesis of provably correct circuits greatly reduces the design time. It also frees the designer from understanding the underlying transformations required to produce hazard-free asynchronous circuits.

3. To achieve the highest performance and smallest circuit size, MEAT compiles the specification into a set of complex CMOS gates. This eliminates the need for slow, inefficient "library" components required by many other synthesis strategies[1,3].

This tool has been used to develop the largest known fully self-timed control-based circuit to date called the *Post Office*. Performance and circuit size were critical, and particular attention was paid to efficient circuit synthesis. The Post Office is a fully self-timed communication co-processor for message passing distributed memory multiprocessors. The circuit contains 300,000 transistors and has an area of 11×8.3 mm, fabricated in a 1.2 micron CMOS process by MOSIS.

The performance of circuits produced by MEAT is reflected in the the Post Office implementation. The Post Office can buffer up to 25 packets for delivery and has a bandwidth of $1\frac{1}{2}$ GBit/second. This performance was achieved using MEAT for the control blocks and with little attention given to the floorplan and data paths[2].

[2]We estimate that with an improved floorplan, the performance could be increased by 20-40% with the same control cells and function units.

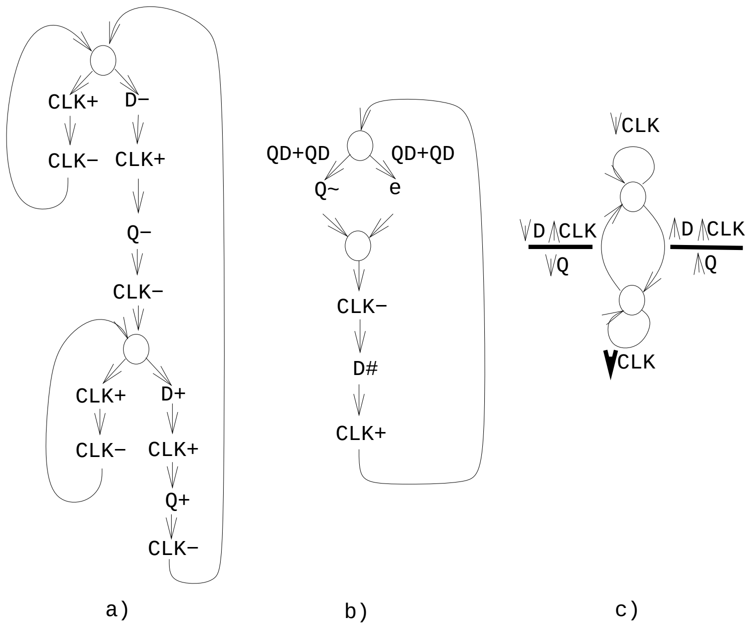


Figure 1: Sample Flip-Flop Specifications

In general there are two types of components used in an integrated circuit – control logic and data path logic[6]. MEAT can efficiently generate control circuitry, but was never intended to be used for data path logic design (such as registers, RAM cells, etc.). Hence in the Post Office, all data path circuits were designed by hand while this tool was used for the control path.

Section 2 describes the state machine specifications, rules and restrictions. Section 3 describes the synthesis techniques used to generate our circuits. Section 4 discusses the correctness of our synthesis and verification of these designs.

2 State Machine Specification

State diagrams are used to model control flow in state machines implemented using MEAT. They are an intuitive method of displaying control and actions, and are similar to the flow charts and state diagrams that are commonly taught in multiple disciplines today[4].

These state diagrams can easily represent parallelism and synchronization. The form is compact when compared to petri-nets, m-nets, STG's, and other graphical representations. The designer can also specify these state machines in a partially minimized form for clarity. These diagrams work well for transition (2 cycle) or level-mode (4 cycle) signalling protocols. Figure 1 shows an STG (a), enhanced STG (b), and state diagram (d) for an asynchronous flip flop.

2.1 State Diagram Specifics

Definition 1 *State diagrams used by MEAT are modeled by a directed graph $G = [V, E, \varphi]$. V is the set of nodes or vertices. The function φ maps from the set of edges, E, to the ordered pairs of nodes $\varphi : E \to V \times V$.*

Definition 2 *There is a set of signals, S, by which the state diagram communicates. These signals are partitioned into two groups, S_I, the input signals, and S_O, the output signals, where $S = S_I \cup S_O$ and $S_I \cap S_O = \epsilon$.*

Definition 3 *Signal transitions, S_T, are of the set $S_T = S \times \{\uparrow, \downarrow\}$. Transitions are partitioned into inputs and outputs, $S_{TI} = S_I \times \{\uparrow, \downarrow\}$ and $S_{TO} = S_O \times \{\uparrow, \downarrow\}$.*

Voltage levels are subsumed by the definition of signal transitions using positive logic. Hence $a\uparrow$ corresponds to signal a becoming asserted, at which time it will change from a low voltage to a high voltage.

Definition 4 *Steady state signals, S_{SI}, are of the set $S_{SI} = S_I \times \{\text{-}, \epsilon\}$.*

Signals that will not change value are represented as being asserted or unasserted. Placing a bar over the signal name indicates the signal is unasserted, e.g. $\overline{input}$.

2.2 Correct Graph Composition

The following rules are used to construct valid MEAT state graphs.

There are a finite number of vertices, V, in the state graph G. Each vertex represents a stable state and is drawn as a circle. Each state is assigned an arbitrary number unique number in the set $\{0 \ldots n-1\}$ where n is the number of states. The initial state is commonly labeled as state zero.

Rule 1 *The function φ consists of an* input burst IB *and an* output burst OB. *Every edge $e \in E$ is labeled with its input burst and output burst.*

Definition 5 *$IB = S_{TE} \cup S_{TDC} \cup S_{SIB}$ where $S_{TE} \subseteq T_I$, $S_{TDC} \subset T_I$, and $S_{SIB} \subset S_I$. Also, if $s \in S_I$, then $\forall s \in S_{TE} : s \notin S_{TDC} \wedge s \notin S_{SIB}$. When $s \in S_I$ then $\forall s \in S_{TDC} : s \notin S_{SIB}$. Finally, $S_{TE} \neq \epsilon$.*

All transitions $e \in E$ are labeled with an input burst and output burst. Each input burst must contain one or more essential transitions (S_{TE}) and may contain "don't care" transitions (S_{TDC}). The essential signal transition(s) will enable the state transition and output burst to fire. Steady state (S_{SIB}) inputs that will not change during the state transition need only be included to make unambiguous choice between two edges exiting a node. Additional steady state inputs may also be included for clarity.

Definition 6 $OB \subseteq T_O$

Rule 2 *A transition $e \in E$ will be satisfied and fire only when all S_{TE} signal transitions in the input burst have occurred. Any S_{TDC} signal transitions* may *occur.*

Rule 3 *Any signal transitions specified by the output burst OB of a transition $e \in E$ will not occur until the transition has been satisfied as specified by the input burst IB. At this point the output burst will fire and the state change will take place.*

Rule 4 *S_{TDC} signal transitions are placed in square brackets in the input burst to distinguish them from S_{TE} signal transitions.*

Rule 5 *There is no limit to the number of edges $e \in E$ exiting node $v \in V$. However, no edge e_i exiting a vertex v can be a subset of another edge e_j exiting from the same vertex.*

MEAT state graphs allow *multiple input change (MIC)* transitioning. When an input burst contains more than one transitioning signal, these signals may change in any order and at any time. This indicates a "synchronization" of two or more parallel functions.

Any unspecified signal transitions are considered illegal by the environment. At least one input change is required to generate a transition as there is no transparent clock!

Rule 6 *Let S_{TDCI} represent the union of all "don't care" S_{TDC} signal transitions for all edges* into *vertex v. If $S_{TDCI} \neq \epsilon$ then each edge* exiting *vertex v must satisfy the equation $\forall s_i \in S_{TDCI} : s_i \in S_{TDC} \vee (s_i \in S_{TE} \wedge (\exists s_j \in S_{TE} \wedge s_j \notin S_{TDCI}))$.*

S_{TDC} transitions are also called *long arcs.* These inputs may change at any time within a sequence of several states, with strict synchronization not required until the final state. All long arcs must end as an essential transition S_{TE} in an input burst, and any set of long arc transitions may not uniquely cause a vertex to fire.

Long arcs can be avoided in state machine implementation by using external hardware such as C-elements to prevent the signal from arriving at the state machine until the strict synchronization point. This may require an additional output from the state machine to "enable" the input when the state machine can accept it. State machines using long arcs may result in improved performance and use less hardware than implementations requiring external hardware and outputs.

The necessity to unambiguously mark transitions from the state of signals in the input set causes transitioning inputs and outputs to change an even number of times when there are loops in the state graph. The designer of a state machine must guarantee that no two transitions of the same polarity (assertions or deassertions) can occur consecutively.

Multiple arcs can exit from a node; a given node may also be the destination for any number of transitions. These state graphs can be drawn in a reduced form where a single state shares a number of compatible operations. When this is done transitions may return to the originating state. Some of the input bursts may then require steady state signals to uniquely distinguish edges exiting a given vertex as can be seen in Figure 2.

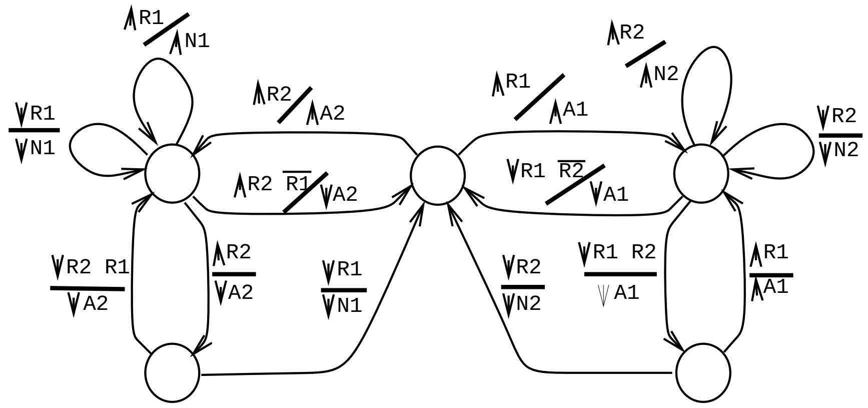


Figure 2: Naking Arbiter SIC State Machine Specification.

Rule 7 *When multiple edges exit a single state, there must be at least one pair of mutually exclusive signals for all pair of edges exiting the state[10]. If there is no pair of mutually exclusive signals for all pair of edges then the state machine can only operate in* single input change (SIC) *mode.*

This rule has been overlooked in some Post Office state machines which include "completion signals" to uniquely select between edges exiting a single vertex. The completion signal must arrive before it's corresponding transition which enables an edge to effectuate the state and output burst change. This places a partial timing order on the input burst, violating true asynchronous behavior[3].

The Naking Arbiter of Figure 2 is a SIC state machine. Since the environment permits the R1 and R2 signals to arrive concurrently these signals pass through a MIC to SIC filter.

Nondeterministic behavior inside a state graph is not allowed. However, the operation of a state machine may be nondeterministic if a mutual exclusion element (ME) is used to order the arrival of two or more inputs into the state machine. ME's are analog devices, and are the only external device that may be required to implement control functions using this methodology. They are easily fabricated in most VLSI technologies, requiring ten transistors in CMOS. ME's are used in the MIC to SIC filter.

2.3 Textual State Machine Description

The graphical state machine description is mapped to a textual description that can be processed by MEAT. Figure 3 is the textual description for the flip flop of Figure 1. The textual description contains two parts. The first section contains the general description and constraints of the state machine. These are specified by keywords followed by an identifier or list of values. The second

[3]Completion signals can be designed that do not violate our state graph rules. This is accomplished by generating two mutually exclusive completion signals, one indicating that the operation has accepted the current request but not terminated, and the second signal indicating that the current request has been processed and the operation has completed.

```
:fsm Asynch-Flip-Flop ;FSM that interfaces sends to the PE.
:in  (D Clk) ;list of input variables
:out (Q) ;list of output variables
:init-in  () ;value of inputs in initial state (optional), default is all zero.
:init-out () ;value of outputs in start state (optional), default is all zero.
:init-state 0 ;initial state (optional), default is 0.
:mex ()      ;sets of mutually exclusive inputs (optional)
;;;:mex (n1 n2) ;use as many :mex statements as required
:state 0 (Clk~) 0 ()
:state 0 (D * Clk) 1 (Q)
:state 1 (Clk~) 1 ()
:state 1 (D~ * Clk) 0 (Q~)
```

Figure 3: Textual specification of Naking Arbiter state machine

section contains the behavioral description of the state machine. Each arc in the state graph contains one :state keyword followed by the initial stable state number, the input burst, the final stable state number, and the output burst. Inputs are specified in a sum-of-products form, the output specified as a single AND term. Signals being asserted are represented by their name. When a signal is unasserted, it is postfixed by a tilde.

3 Synthesis

3.1 Performance Objectives

A key factor influencing the performance of asynchronous sequential circuits is the number of gate delays that are required to generate the output functions. Hence, a major objective of our design style is to minimize the number of gate delays from the inputs to the outputs.

Figure 4 shows a typical state machine, and a more detailed construction of MEAT generated state logic. The only signals accessible to the outside are the inputs, X, and the outputs, Z.

The filter box has two functions. First, high capacitance inputs or inputs with a slow rise time will be passed through an inverter or schmitt trigger. This will reduce the load on the input line and make a crisp local signal which can avoid any gate threshold variances which could result in hazards[17]. Secondly when an unasserted input signal is required by the state or output boxes, the filter box will invert that signal. Each input will have its inverted and uninverted signal shared among all function blocks in the state machine to eliminate hazards and create a smaller implementation. All components in a state machine are assumed to be physically close, so wire delays of the same signal between different components in a state machine is insignificant. The filter box will constitute zero or more gate delays for any MEAT implementation.

The output and state boxes generate the feedback (state) variables, Y, and output signals, Z, respectively. Post Office implementations usually consist of a single complex gate to realize the

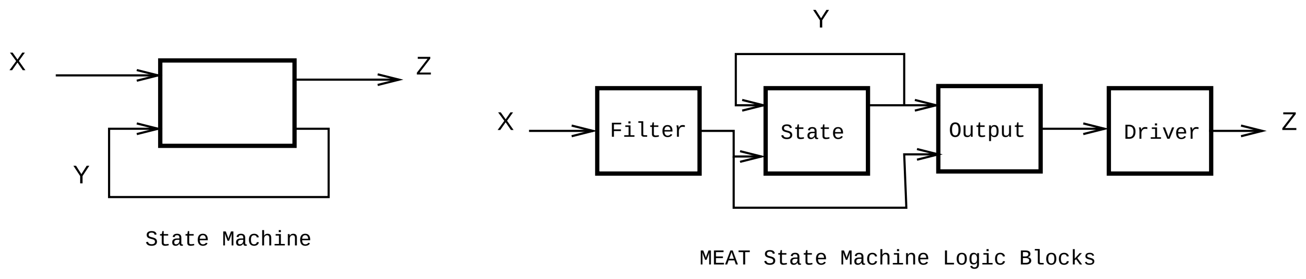


Figure 4: State Machine Generation

function as described in Section 3.5. This can reduce both the number of gate delays and devices in the implementation of a function block. Complex gates generally produce the unasserted signal, so an inverter may be required to produce the positive voltage levels. The feedback logic will contain two or more gate delays, and the output logic consists of one or more levels of logic.

The driver block is used to generate positive output voltage levels and to increase the signal strength when the output is heavily loaded. Hence it will contain zero or more levels of gate delays.

A further method used to reduce the maximum number of gate transitions is to enforce *Single Transition Time (STT)* state changes. All state changes where only a single state variable is modified is an STT assignment. Transitions where two or more state variables are modified to arrive at the next state can result in multiple state transition times. With two or more state variables changing, races can be avoided by using one or more intermediate "bridging" states to move from the starting state to the final state by changing one state variable at a time. Bridging states are used to remove races, but the resulting assignment is not STT, as two or more sequential passes through the state logic are required to generate the final stable state.

STT state assignments which allow only one state variable transition per state change may always be generated simply by adding a sufficient number of state variables[16]. In general, this is undesirable; the speed gained by using the resulting STT assignment is more than offset by the increased complexity and circuit area required to generate the additional feedback variables.

MEAT minimizes the number of state variables for each machine while also generating STT assignments. This results in possible race conditions. Assuring that the outcome of the race is independent of the order in which the feedback variables change results in a *non-critical* race which exhibits correct asynchronous operation. Allowing state variable races is comparable to permitting MIC operation, which exhibits races between the signals in an input burst.

The performance benefits of STT state assignments may come at the cost of more hardware. However, STT assignments also allow us to simplify the construction of self-timed circuits (see Section 4).

An analysis of Figure 4 shows that the minimum delay from input to output of a MEAT state machine is 2 gate delays (one for the output logic, and one for the invert logic, or two delays in the output logic). This ignores the degenerate case where an output is equivalent to an input.

State machine implementations in the Post Office show a minimum gate delay of two and a maximum gate delay of seven for any input burst. In the case where there are seven gate delays, four of the delays are simple inverters, which offer the minimum transit time for the technology.

3.2 Specifying A Flow Table

The first automated task of MEAT is to generate a flow table from the textual specification of the state machine. Each row of this table represents a node in the state diagram. Each column represents a unique combination of the input variables. For each possible combination of inputs and state variables, this table will specify the asserted outputs (if any) and next-state values. If a next-state value is the same as that of the current row, the state machine is said to be in a *stable state*. If the next-state value specifies a different row, the table entry represents an *unstable state*.

All states will have a stable entry point, where the input burst begins. The input burst is satisfied when it reaches an unstable state that will transition directly to the next state specified and fire the output burst. When multiple inputs change, the subcube spanned by the entry and exit points corresponds to a single product term of the function. The values of the outputs and current state are identical to the entry point for all stable states in each of these cubes. Meat will automatically cover all transitioning terms of each MIC cube with a single term. In some cases these cubes can have overlapping coverage. Any entry in the flow table not reached by input bursts is labeled as “don’t care” and can take on any value for the outputs or state values.

These don’t care entries are disallowed input sequences for the current state. As these entries cannot occur, any value can be assigned to them in the state reduction and circuit specification steps. The inclusion of don’t cares can significantly simplify the state reduction and lead to much simpler circuits. As it is not evident at problem specification time which values will lead to the simplest circuit, value assignment is deferred to a later time.

When an input burst has been satisfied, an unstable state will be reached effectuating the state change. The associated output burst may occur concurrently with the state change, or can be restricted to change *after* the state change has occurred. All signals in the output burst are labeled as don’t cares in the unstable exit state of the flow table. Since all state transitions are STT, the monotonicity of output voltage changes is guaranteed, regardless of whether the unstable entry is mapped to a zero or one. Putting off the value assignment for the outputs can result in simpler circuits. This also may result in the output transition being delayed until the state variable transition is complete.

3.3 Selecting The Minimized State Machine

The next step in the design process is to attempt to reduce the number of rows in the flow table by merging selected sets of two or more rows into one while retaining the specified behavior. The procedure for selecting these row sets is not complicated, but for brevity will not be described in this paper. It cannot be shown that minimizing states in a specification will simplify the hardware or increase the performance of a state machine (it’s an NP hard problem). However, a reduced state machine can result in fewer state variables which will likely result in a smaller, faster implementation.

After specifying the flow table, MEAT calculates the set of *maximal compatible* states. The set of maximal compatibles consists of the largest sets of state rows which can be merged, which are not subsets of any other such set. There may be various valid combinations of the maximal compatibles that can be chosen to produce a reduced table with the same behavior, but the combination that leads to the simplest implementation is not always evident. We have found that the simplest solutions tend to be either a) the solution with the fewest number of states, or b) a solution where only a single state variable needs to change for all state transitions in the implementation.

The final choice of minimized states must be chosen by the designer from the the set of original state markings. There are three constraints on this choice. First, only compatible states as returned by MEAT may be combined into a single reduced state (states must be *compatible*). Second, each state in the original design must be contained in at least one of the reduced states (the covering must be *complete*). Third, selecting certain sets of states to be merged may imply that other states must also be merged (the covering must be *closed*). For example, merging two states states 0 and 1 may require that states 2 and 3 must also be merged. If a covering is chosen which is not closed, MEAT will inform the user that the pairs are not valid. Note that all states in a maximal compatible need not be combined into a single state in the final implementation.

3.4 Implementation

A set of state variables is then assigned to uniquely identify each of the new rows resulting from the reduction step. In contrast to synchronous control logic design, state variable values may not be randomly assigned to states, but must be carefully chosen to prevent races. The MEAT state assignment algorithm is based on a method developed by Tracey[15]. Several valid assignments may be produced, and each will be passed to the next stage for evaluation. This will result in unique implementations for each state assignment.

After state codes are assigned, the next synthesis stage computes a canonical sum of products boolean expression for each output and state variable. A modified Quine-McCluskey minimization algorithm is used. The resulting expression includes all essential prime implicants, and possibly other prime implicants and additional terms necessary to produce a covering free of logic hazards. It may be possible for each output or state variable to be specified using several alternate minimal equations. The large number of don't cares entries typically present in the flow table increase the likelihood that more than one minimal expression will be found.

Each equation is given a heuristic "weight" that indicates the difficulty of building the function in CMOS. The equations of minimal weight are usually chosen for each output[4]. When multiple state assignments produced, the total weight of each unique SOP specification is used to choose between various instantiations.

[4]Equations of larger weight are chosen when they reduce the number of inputs and their complements. This can simplify the final circuit.

3.5 CMOS Optimization Of SOP Equations

Finally there is a back-end to this tool set that will generate minimized complex gates and schematics. This interfaces with the Electric[12] design system for schematic layout. The complementary nature of CMOS n-type and p-type devices is exploited to generate a single, complex, static gate through simple function preserving transformations. These transformations can increase performance while reducing the area and device count. As an SOP equation is folded into a single gate, the number of logic levels required to generate the output can drop from 2 to 1. If the function is complex, it can easily be broken up into a tree of complex gates with 2 or more logic levels, but better overall performance[14].

The single complex gate generates the output in negative logic (low voltage levels for asserted signals). A convention of positive logic levels is assumed for all signals external to the state machine, requiring that the outputs be inverted. This is a feature for performance reasons as the gain of the inverter can be used as a driver to increase signal strength and reduce rise and fall times. When outputs need to drive an extreme load, a buffer tree will be used.

All state machines also require a reset signal to place the storage logic into the correct initial state. Storage in these state machines is implemented via the state variables. If a single complex gate is used to generate the output, the state storage is reset by NOR-ing the output with the reset line. For complex gate trees, a resetable NAND gate is used. Although the performance of the NOR gate is not optimal, the load on the feedback lines is local to the state machine and typically small so a large gain is not required.

4 Implementation Specifics

4.1 Device Constraints

The class of entirely delay-insensitive circuits has been shown to be very limited[8]. When implementing MIC circuits directly from flow tables, if there is more than one essential prime implicant it may not be possible to generate hazard-free implementations. Our mechanism can produce multiple prime implicants, and hence we may have hazards inherent in the specification.

These hazards have been circumvented in other work by using large inertial delays or making certain timing assumptions[16,11,7]. Most of these techniques require extra logic or result in decreased performance due to the delay elements.

Inherent hazards are eliminated in MEAT by placing bounds on delays based on realistic properties of the gates and wires. This is done without any penalty in logic simplicity or performance! The following property must hold for the implementation of hazard-free circuits:

$$d_{fmin} + d_{Lmin} > d_{Lmax} \tag{1}$$

The minimum delay through a state variable feedback path, or y_i, is represented by d_{fmin}. The minimum and maximum logic delays from any input (x_i) or feedback (y_i) input to output are represented by d_{Lmin} and d_{Lmax} respectively. This timing assumption intuitively states that all logic will see the input burst (IB) and state change (SC) as two discreet events $\forall x_i \in IB \prec y_i \in SC$. In an actual circuit, the IB and SC may in fact be "pipelined", but the SC cannot interfere with the IB operation. The simplicity of Equation 1 is partially due to the single transition time nature of the implementation.

Any timing assumptions require knowledge of the implementation mechanics and the implementation media. Equation 1 can be guaranteed to hold for our CMOS implementations generated through MEAT without adding any additional delays or gates. This timing constraint may be applicable to other technologies.

The following is a summary of the properties of our state machines, mainly condensed from Section 2. Due to the construction steps taken in Section 3.4, all feedback and output logic is hazard-free. We can combine this invariant and the timing Equation 1 with the following state machine specification properties to show that our implementations are race free:

1. Inputs are monotonic, but can be MIC, allowing sets of input changes to be grouped in "bursts".
2. The completion of all input bursts are deterministic and unambiguous.
3. No outputs or state variables can change until the input burst is complete.
4. All state transitions are STT.
5. Long arcs cannot effectuate a state change.
6. The state machine operates in *fundamental mode*, where no new inputs may arrive until the feedback variables are stable with the exception of long arcs.

Theorem 1 *If each output and state function can be shown to contain no logic hazards when adhering to the above conditions, then by Equation 1 the entire state machine is hazard free.*

Proof: All logic hazards are removed from combinational logic so, the state machines will be hazard free for input changes without state changes, or state changes without input changes. Equation 1 guarantees that the input change and state change are non-interfering events to the logic, appearing as sequenced input and state changes. Hence the logic will behave as combinational logic, which is hazard free.

To show in more detail how each state and output function is designed to be hazard free under the above assumptions, we will examine the possible transitions that may occur for any input change. For each input burst, each output will remain at 0 or 1, or make a transition. Since the state change is STT, the MIC conditions must hold for both input bursts and state changes.

There can be no 0 hazard by correct construction. No combination of valid input values from the entry point to the end of the input burst may produce an output of 1. Also, no combination of state values from the unstable entry point marking the beginning of a state change to its termination in a stable state may produce a 1.

One hazards cannot occur in combinational logic for SIC state machines as long as all adjacent 1's are covered by a single term in the final implementation. MIC and STT logic requires that a single term covers the entire "cube" of transitioning values for the input burst or state change burst. The union of all input bursts and state change bursts where an output remains asserted must be covered by a single term where any given long arc may change.

Outputs may not be asserted until the input burst is complete. Hence an output that will transition from a 0 to a 1 will remain 0 until the input burst is complete, and the 0 hazard analysis must hold to that point. At the completion of the input burst the output may become asserted. If so, the output must remain asserted during the state change, and the 1 analysis must hold. Alternatively, the output may not change until the state change is complete. In this case the 0 analysis must hold during the input burst and output change, at which time the output will become asserted.

An output unasserting from 1 to 0 may be held high by several terms. The output may not change until the burst is complete. All of the terms will unassert in an arbitrary order, with at least one term holding the output valid until the input burst has completed. We must guarantee that there is no "interfering" term that can become asserted during the input burst when there is a 1 to 0 transition. Assume that the input burst has completed. Also assume that the interfering term has been enabled. If all the terms asserting the function become unasserted before the interfering term asserts, a hazard can occur. The interfering term may produce a dynamic hazard or runt pulse on the output. This hazard cannot occur with SIC combinational logic. As shown by Nowick[11], these hazards can be eliminated by not allowing certain states to be compatible in the state minimization process.

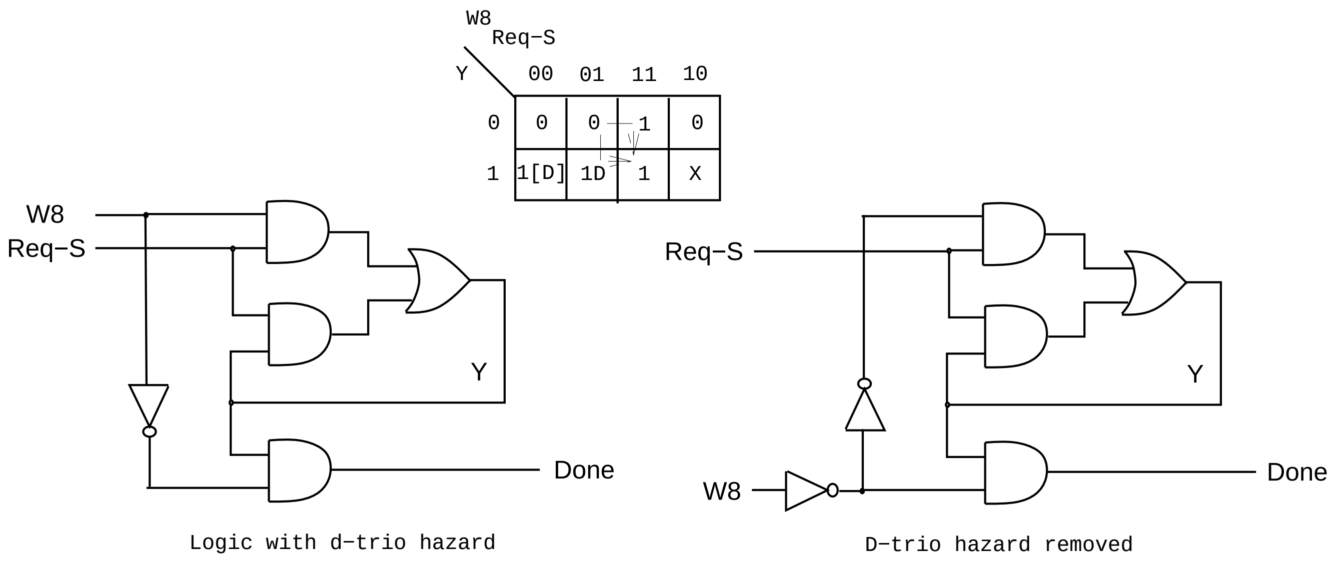
All of these combinational hazards can be avoided in the synthesis steps of MEAT, producing hazard-free state machines.

4.2 Verification

Pure delay-insensitive analysis of circuits generated by MEAT will show that there are a number of hazards and races that can occur. Dill's verifier[2] has been used to evaluate our designs, showing the hazards that are present in these circuits. This allows us to investigate concrete examples where the timing Equation 1 must hold.

Figure 5 shows a typical hazards that is present in designs synthesized by MEAT. D-trio hazards can generate 1 or 0 hazards. Examining the physical performance of the devices and wires in these implementations containing d-trio hazards show that they will not occur. Dill's verifier will also point out critical races. The cause of these is similar in nature to the d-trio races as it is caused by the state change burst being evaluated by a logic component before the input burst.

Transformations do exist that will convert many MEAT circuits into delay insensitive implementations. Figure 5 shows the transformation for a d-trio hazard. This changes the order in which logic blocks see the output change by adding inverters to some of the inputs. For this example, the



Y \ W8 Req−S	00	01	11	10
0	0	0	1	0
1	1[D]	1D	1	X

Figure 5: Hazard removal from "Sendr-Done" state machine

performance is not effected by the change, but the circuit is larger. Complex gates can also result in speed-independent implementations as races between different logic elements can be eliminated. These transformations are not necessary for the circuit to perform in a speed-independent manner. Applying transforms to make a circuit truely delay-insensitive can result in a larger and slower implementation.

Implementations that only use single complex gates result in a less strict timing assumption. Here the weakened constraint of $d_{Lmin} > d_{inv}$ is sufficient to guarantee correct circuit operation, where d_{inv} is the delay of a basic inverter.

5 Summary

There is a need for good tools to be developed in the asynchronous logic community that can mask the implementation complexity of these devices and show excellent performance. MEAT is a tool that was developed for this purpose and was used to build a very large self-timed communication processor. All state machines designed with this tool have worked flawlessly and the performance is outstanding.

One goal in the development of this tool was to generate fast, compact, efficient circuits. Showing the excellent performance that can be achieved with asynchronous designs is an important part of forwarding this technology to the hardware community at large. Many of the Post Office state machines have been offered to the design and tool community as samples of of a real design. The community can use these state machines as test cases for tools as well as compare performance results between different approaches. The state machine specification described in this paper is simple to map to most other specifications. We are also interested in specifications others have used to develop circuits. We would also like to challenge others to produce hardware displaying better performance to our tool set!

Building a large, fully self-timed circuit has resulted in many insights. First, this tool set must be completed. The back end only produces schematics. To significantly cut design time of high performance self-timed implementations requires that the back end also produces layout. Initial investigation leads us to believe that it would be fairly straight forward to produce layout from our schematics. Secondly, there are a number of performance factors that should be included in the tool set. As a circuit is passed down through the different stages of the tool, some information is lost. The complexity of the algorithms and simplicity of the circuits could be enhanced by passing some of this information down. Third, we need a front end to these tools to prove liveness, safeness, and deadlock free properties for single graphs as well as an entire circuit of concurrently communicating state machines.

6 Acknowledgments

This work could not have been completed without the help of a large set of people. In particular, key contributions have been made by Al Davis, Bill Coates, Steve Nowick, and Bic Schediwy. Dick Lampman has also facilitated this work by supporting its progress at Hewlett Packard.

References

[1] E. Brunvand and R. F. Sproull. Translating Concurrent Communicating Programs Into Delay-Insensitive Circuits. In *International Conference on Computer-Aided Design, ICCAD-89*, Randall Bryant, editor. IEEE Computer Science Press, 1989.

[2] David L. Dill. *Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* MIT Press, 1989.

[3] Jo C. Ebergen. A Formal Approach To Designing Delay-Insensitive Circuits. Technical Report Computing Science Note 88/10, Eindhoven University of Technology, May 1988.

[4] W. I. Fletcher. *An Engineering Approach to Digital Design.* Prentice-Hall, Englewood Cliffs NJ, 1980.

[5] A. B. Hayes. Self-Timed IC Design With PPL's. In *Third Caltech Conference on Very Large Scale Integration*, R. E. Bryant, editor, pages 257–274, Rockville, Maryland, 1983. Computer Science Press, Inc.

[6] John P. Hayes. *Computer Architecture and Organization.* Computer Science. McGraw Hill, 1978.

[7] L. Lavagno, K. Keutzer, A. Sangiovanni-Vincentelli. Synthesis of Verifiably Hazard-Free Asynchronous Control Circuits. Technical Report UCB/ERL M90/99, University of Berkeley, November 1990.

[8] Alain J. Martin. The Limitations to Delay-Insensitivity in Asynchronous Circuits. In *Sixth MIT Conference on Advanced Research in VLSI*, W.J. Dally, editor. MIT Press, 1990.

[9] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, P.J. Hazewindus. The Design of an Asynchronous Microprocessor. In *Decennial Caltech Conference on VLSI*, C.L. Seitz, editor, pages 251–273. MIT Press, 1989.

[10] R. E. Miller. *Switching Theory*, volume 2. Wiley, New York, New York, 1965. Chapter 10 is a review of Muller's work on speed independent circuits.

[11] Steven M. Nowick. Synthesis of Self-Timed State Machines Using a Local Clock. In *International Conference on Computer Design*, 1991.

[12] Steven M. Rubin. *Computer Aids for VLSI Design.* VLSI Systems. Addison-Wesley, 1987.

[13] Charles L. Seitz. *Introduction to VLSI Systems*, chapter 7, "System Timing". Addison Wesley, 1979.

[14] Ivan E. Sutherland and Robert F. Sproull. Logical Effort: Designing for Speed on the Back of an Envelope. In *Proceedings of the 13th Conference on Advanced Research in VLSI*, Carlo H. Sequin, editor, pages 1–16. UC Santa Cruz, March 1991.

[15] J.H. Tracey. Internal State Assignments for Asynchronous Sequential Machines. *IEEE Trans. Electronic Computers*, EC(15):551–560, August 1966.

[16] S. H. Unger. *Asynchronous Sequential Switching Circuits.* Wiley-Interscience, New York, New York, 1969.

[17] C. H. van Berkel. Beware the Isochronic Fork. Technical Report Nat. Lab Rep. UR 003/91, Philips Research Laboratories, January 1991.