# Automatic Synthesis of Fast, Compact Self-Timed State Machines[1]

Kenneth S. Stevens

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

**Abstract**

**An automated synthesis tool, called the *Most Excellent Asynchronous Tool, or MEAT* is presented. This tool has been used to specify and synthesize asynchronous finite state machines (AFSMs) for a fully self-timed 300,000 transistor communication co-processor called the Post Office. The specification is done with stylized state diagrams with a restricted form of multiple input change constraints called *burst-mode*. 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 self-timed CMOS implementations with outstanding performance and compactness. When burst-mode is coupled with a timing inequality, the circuits can be verified as hazard free.**

## 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

[1]This work is supported by Hewlett Packard Company, Palo Alto, CA.

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 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 [17]. 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. The speed of operation of asynchronous circuits has been demonstrated to be on par with that of their synchronous counterparts [7,10]. Because the elemental components are small, the task of removing hazards and producing correctly operating asynchronous components is manageable.

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 an 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 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 [1,5].

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 $\times$ 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 asynchronous finite state machine

(AFSM) synthesis with little attention given to the floor plan and data paths which adversely affected the overall performance[2].

In general there are two types of components used in an integrated circuit – control logic and data path logic [8]. 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 4 describes the synthesis techniques used to generate our circuits. Section 5 discusses the correctness of our synthesis and verification of these designs.

## 2 State Machine Specification

Asynchronous circuits are specified for MEAT as a burst-mode Mealy state machine. The input specification is compiled into a set of CMOS complex gate. The result is an implementation which is efficient both in terms of speed and area.

### 2.1 Burst-Mode Specifications

In order to achieve the required hazard free AFSM implementation, it is necessary to place constraints on how inputs are allowed to change. The most common is the *single input change* or SIC constraint [20]. SIC circuits may require state transitions after each input variable transition. SIC circuit response will be artificially slow in cases where the next output does not occur until several inputs have changed, either due to too many state transitions or due to the external arbiters required to sequence the multiple inputs. *Multiple input change* or MIC circuit design methods have been developed [20,3] but either required input restrictions or delays which were unsuitable for performance-oriented implementations. As a result a design style called *burst-mode* was developed and used in the Post Office implementation which permits a certain style of multiple input change. The burst-mode implementation method does not require performance inhibiting local clock generation or flip-flops.

Burst-mode state diagrams allow a constrained form of MIC and multiple output change (MOC) operation. When a state change or output is triggered by a conjunction of input signal transitions (an input burst), these signals are allowed to change in any order and at any time. Allowing MIC operation simplifies the definition of synchronization operations and tends to more closely match the designer's mental model of the hardware.

Burst-mode transitions can be defined in terms of flow table specifications. A flow table [20] is a two-dimentional array structure which captures the internal and external states of a circuit. The rows of the table correspond to the internal state of the circuit, and the columns to the state of the inputs. Table entries are ordered pairs containing the next state and current output information.

[2]With an improved floor plan, the performance could be increased by 20-40% with the same control cells and function units.

When the next state in an entry corresponds the the current state, flow table is in a *stable* state, otherwise the current state is unstable and an internal *state transition* will occur. A simple way of understanding the flow table is to note that horizontal movement within a row represents changes in the values of input signals, while vertical movement within a column represents a state transition.

**Rule 1** *Burst-mode transitions always begin in a stable* initial state *of the flow table.*

**Definition 1** *Burst mode allows MIC and MOC operation. All inputs required to effectuate the transition are called the* input burst, *and the outputs generated by the transition are called the* output burst.

**Rule 2** *Each input burst must contain at least one input signal transition. Output bursts may be empty.*

**Definition 2** *The input burst will move the current flow table state horizontally in the same row until all inputs in the burst have changed. This final state is called the burst-mode* transition *state.*

**Rule 3** *All intermediate states reachable by an input burst between the* initial state *and the final* transition *state must contain the same next state and output entries as the* initial state.

**Rule 4** *The* transition *state entries in the flow table will lead to the next stable* initial state *and will change the output entries corresponding to the output burst.*

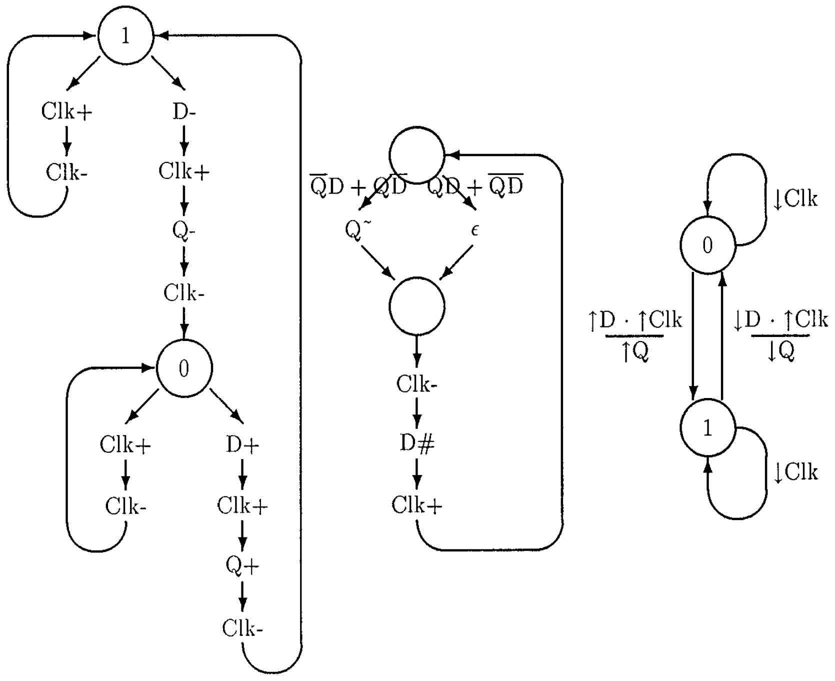
**Rule 5** *No input change may occur until all outputs have been asserted and the next stable* initial state *in the flow table has been reached. All circuit elements must also be allowed to stabilize before the next inputs arrive. This is the burst-mode* stability *requirement.*

Burst-mode is similar in nature to the *fundamental mode* of operation. However, each allowed input burst will result in a particular path through the FSM state space, starting at the stable entry where the burst begins. To correctly implement MIC behavior the circuit must remain stable in the initial row until all inputs in the input burst been accepted, at which point the transition state will be reached. Correct implementation of the state transition requires that no new inputs to the AFSM arrive until the state transition has occurred. Further, correct implementation of MOC behavior requires that all outputs in the output burst are generated before any subsequent inputs arrive.

It is not necessary in a physical implementation that a state change occur in each transition state. Output bursts and state changes can also be generated in parallel.

## 2.2 State Diagram Specifics

Because flow tables are tedious to specify and understand, the MEAT interface uses state graph specifications. The mealy state diagrams are similar to the flow charts and state diagrams that are commonly taught in multiple disciplines today [6]. The format is compact when compared to petri nets, m-nets, STGs, and other graphical representations. The designer can also specify 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) and enhanced STG (b) for an asynchronous flip flop taken from Moon [13], and the analogous state diagram (c).


1
Clk+
Clk-
D-
Clk+
Q-
Clk-
0
Clk+
Clk-
D+
Clk+
Q+
Clk-
$\overline{Q}D + Q\overline{D}$
$QD + \overline{QD}$
Q~
$\epsilon$
Clk-
D#
Clk+
↓Clk
0
$\frac{\uparrow D \cdot \uparrow Clk}{\uparrow Q}$
$\frac{\downarrow D \cdot \uparrow Clk}{\downarrow Q}$
1
↓Clk

(a) STG (b) Enhanced STG (c) State Diagram

Figure 1: Sample Flip-Flop Specifications

The state diagram specifications described below implement the burst-mode assumptions.

**Definition 3** *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 $\varphi$ maps from the set of edges, $E$, to the ordered pairs of nodes $\varphi : E \rightarrow V \times V$.*

**Definition 4** *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 5** *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 $S_{TO} = S_O \times \{\uparrow, \downarrow\}$. All signal transitions are assumed to be monotonic.*

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 6** *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. $\overline{input}$.

## 2.3 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 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 6** *All inputs and outputs must strictly alternate between rising and falling transitions for any valid path in the directed graph $G$.*

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.

**Definition 7** *The function $\varphi$ 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 8** *$IB = S_{TE} \cup S_{TDC} \cup S_{SIB}$, 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 ($S_{TDC}$) also called *long arcs*. Long arcs may change value across a set of states, and are not constrained to a single state transition. 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. Long arc transitions are placed in square brackets in the input burst to distinguish them from essential signal transitions.

**Definition 9** *$OB \subseteq T_O$*

**Definition 10** *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.*

**Definition 11** *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 7** *No edge $e_i$ exiting a vertex $v$ can be a subset of another edge $e_j$ exiting from the same vertex. There is no limit to the number of edges $e \in E$ exiting node $v \in V$.*

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.

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

$S_{TDC}$ input transitions 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.

**Rule 9** *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 [12]. 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.*

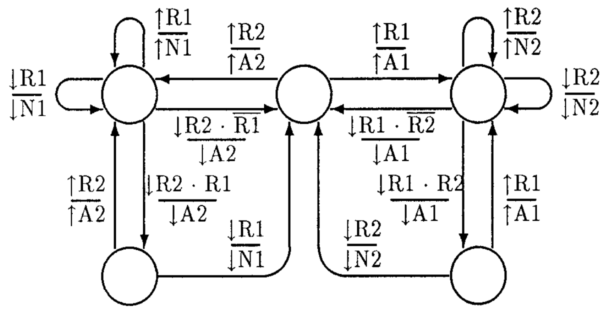


Figure 2: Naking Arbiter SIC State Machine Specification.

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 *sequencer* which converts MIC signals into SIC signals.

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. MEs 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 12 transistors in CMOS. MEs are used in the sequencer.

### 2.4 Textual State Machine Description

Presently MEAT does not contain a state graph editor so a textual specification format is used. The more natural graphical state machine description may be trivially mapped to the textual version. 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 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 MEAT Design Style

Key factors influencing the performance of a circuit network are (a) the number of gate delays required to generate the output, (b) the number of series transistors between the power rails and outputs, and (c) the load and stray capacitance on the inputs and outputs. The design of MEAT

```
: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), use as many as needed.
: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

state machines attempts to address these issues as well as avoid potential circuit failures that arise from isochronous forks and other assumptions.

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, $Z$.

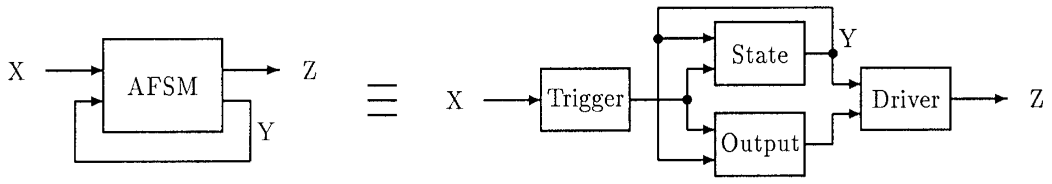


Figure 4: State Machine Generation

The trigger 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 [21]. Secondly when an unasserted input signal is required by the state or output boxes, the trigger 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 internal signal between different components in a state machine is insignificant. The trigger 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 function as described in Section 4.4. 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 state 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. This block is also used to remove isochronous forks from AFSM outputs. 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 [20]. 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 multiple inputs to change simultaneously. These state variable races must obey our burst-mode constraints.

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 4.1) and assure the burst-mode assumptions are not violated (see Section 5). Burst-mode assumptions may be violated if there is a large delay required to stabilize the circuit after an input burst has arrived and outputs are generated. Without STT transitions, outputs probably should be constrained to occur *after* the final stable state has been reached to conform to burst-mode requirements.

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.

The AFSMs in the Post Office show a minimum input to output 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 and load for the technology.

## 4 Synthesis

### 4.1 Specifying A Flow Table

The first automated task of MEAT is to generate a burst-mode 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.

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. When the output change is made in the unstable entry of the flow table, performance of the AFSM is improved as the outputs and state variables are changed concurrently. This performance is gained but the burst-mode stability requirement must be verified. When the output change is delayed until the next stable state is reached, the state is changed which then causes the output to change in serial, costing some performance. MEAT places priority on producing the simplest circuit, so an assignment is chosen which yields the simplest circuit. This can result in faster average operation as the logic will be smaller, and more states may be compatible which results in fewer state variables and state changes.

When multiple inputs change, the subcube spanned by the entry and exit points of a state in the flow table must correspond 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.

### 4.2 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 (its 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, and the combination that leads to the simplest implementation is not always evident. Experience indicates 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.

## 4.3 Generation of Implementations

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 [19]. 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 (SOP) boolean expression for each output and state variable. An algorithm developed by Nowick [15] is used which produces guaranteed hazard free two level combinational logic implementations. There may be multiple unique and minimal equations for each output or state variable. The large number of don't care entries typically present in MEAT flow tables increase the likelihood that more than one minimal expression will be found.

Each equation is given a heuristic "weight" that calculates the estimated performance and difficulty of building the function in CMOS. The equations of minimal weight are usually chosen for each output[3]. When multiple state assignments are produced, the total weight for each entire

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

implementation is calculated by a summation of the solution of minimum weight for each state variable and output for each implementation.

### 4.4 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 [16] design system for automatic 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 and requires more than three or four transistors between the power rails, it will be broken up into a tree of complex gates with 2 or more logic levels. This tree will have better overall performance because there are fewer gates between the power rails and the intermediate node capacitances are reduced [18].

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 while placing minimal output load on the complex gate. 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 state variable, the storage loop 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 usually not required. When loads are large, such as where a state variable also serves as on output, signal amplification will be accomplished in the driver block of Figure 4.

## 5 Implementation Specifics

### 5.1 Device Constraints for Hazard Removal

The class of delay-insensitive and even speed-independent circuits has been shown to be very limited [2,11]. When there is more than one essential prime implicant in a sequential MIC circuit created directly from flow tables it may not be possible to generate hazard-free implementations. Flow tables can produce multiple prime implicants, and hence hazards may inherently exist in the specification.

These function hazards will only occur using an unbounded delay model. These hazards have been circumvented in other work by using bounded delay models coupled with large inertial delays or certain timing assumptions [20,14,9]. Most of these techniques require extra logic or result in decreased performance due to the delay elements or extra logic.

Function hazards can be eliminated in MEAT circuits as well by using a bounded delay model based on the physical properties of the circuit realization. A simple timing inequality can be used to verify that the circuit will operate correctly under the bounded delay model:

$$dT_{min} + dS_{min} + dO_{min} > dT_{max} + dO_{max} \tag{1}$$

The minimum and maximum delay of any input ($x_i$) through the trigger box is represented as $dT_{min}$ and $dT_{max}$ respectively. The minimum delay through a state variable feedback path, or $y_i$, is represented by $dS_{min}$. The minimum and maximum delay through the output logic is $dO_{min}$ and $dO_{max}$. 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 actually be "pipelined" through the gates, but the SC cannot interfere with the IB operation. Equation 1 is simplified due to the single transition time nature of MEAT implementations.

Any timing assumption requires knowledge of the implementation mechanics and the implementation media. Equation 1 can be checked and guaranteed to hold for our CMOS implementations generated through MEAT without adding any performance inhibiting 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 4.3, all state and output combinational logic is hazard free. Combining this invariant with the assumption that the timing inequality of Equation 1 holds, with the following state machine specification properties, it can show that MEAT implementations are hazard 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 burst-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.*

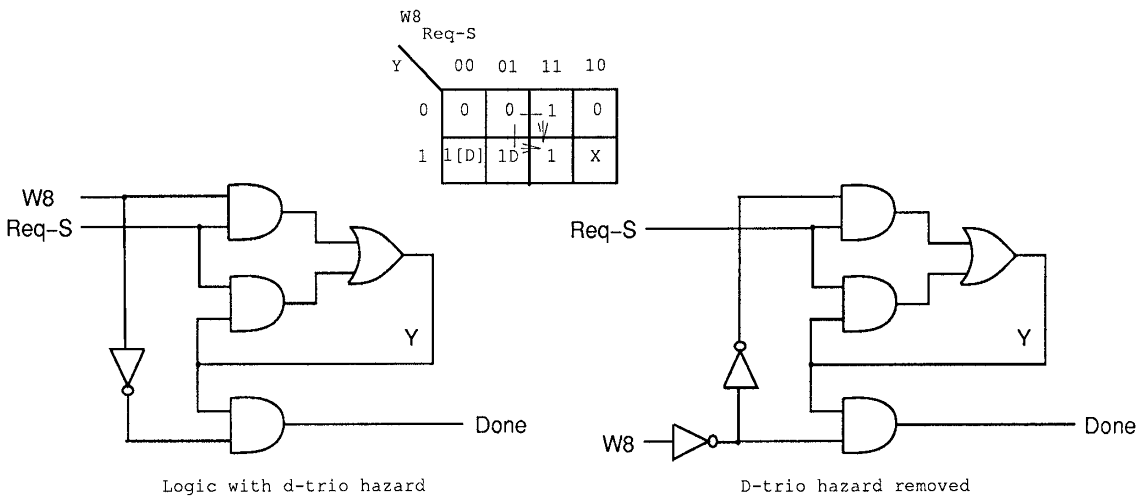


Figure 5: Hazard removal from "Sendr-Done" state machine

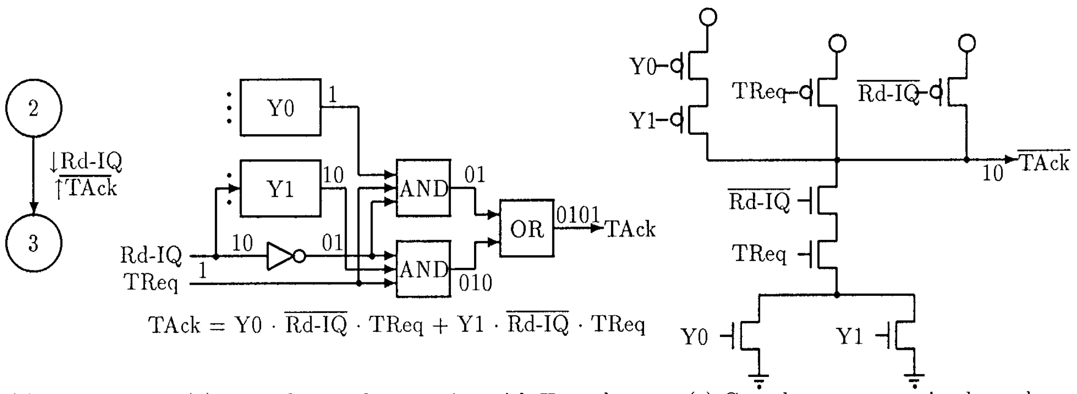
**Proof:** All logic hazards are removed from combinational logic. Therefore the state machines will be hazard free for input changes without state changes, or state changes without input changes. Equation 1 guarantees that the input change and state change are non-interfering events to the logic, appearing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

## 5.2 Verification

Speed-independent and delay-insensitive analysis of MEAT AFSMs will show that there are a number of hazards and races that can occur. Dill's verifier [4] 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 inequality of Equation 1 must hold.

Figure 5 shows a typical function hazard that is present in designs synthesized by MEAT, called a *d-trio* hazard. D-trio hazards can generate static 1 or 0 hazards. In this example, the two inputs change from "01" to "11". For example, in the circuit on the left in Figure 5, if the *Done* AND gate sees a change in *Y before* it sees the change from the inverted *W8* signal, a static 1 hazard will occur on the output. Examining the physical performance of the devices and wires in implementations such as this which contain d-trio hazards will show that the timing inequality holds and that these hazards will not occur in practice. 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 remove hazards thus convert many MEAT circuits into speed independent gate-level implementations. The second circuit in Figure 5 shows the transformation which has removed the d-trio hazard. The extra inverter changes the order in which logic blocks are forced to evaluate input changes. In this example, the performance is not effected by the change as no extra gate delays are required to generate any output. However, the circuit is larger.



2
↓Rd-IQ
↑TAck
3
Y0
Y1
AND
AND
OR
Rd-IQ
TReq
0101
TAck
TAck = Y0 · $\overline{\text{Rd-IQ}}$ · TReq + Y1 · $\overline{\text{Rd-IQ}}$ · TReq
Y0
Y1
TReq
$\overline{\text{Rd-IQ}}$
$\overline{\text{TAck}}$

(a) SG Segment (b) AND-OR Implementation with Hazard (c) Complex gate removing hazard.

Figure 6: PE-Send-Ifc Hazard Removal with Complex Gate

Complex gates can also result in speed-independent implementations as races between different logic elements can be eliminated. Figure 6 shows a piece of the state graph for the *PE-Send-Ifc* AFSM from the Post Office. Two of the seven AND terms in the implementation of *TAck* generated by MEAT are shown in (b) with an AND-OR gate implementation. This circuit as shown has a dynamic 1 hazard. If the top AND gate is significantly slower than the bottom AND gate, then the bottom AND gate can turn on then off before the top AND gate ever fires, producing the hazard as the output can bounce 0–1–0–1 before stabilizing. The *TAck* function is designed as a single complex gate in (c) as generated by MEAT. This complex gate removes the hazard entirely.

If the timing inequality from Equation 1 holds, then the dynamic 1 hazard in Figure 6 (b) will not occur in practice. This can be verified in a circuit realization by examining the physical devices and their loads. The complex gate transformations may not be necessary for the circuit to perform in a speed-independent manner. Implementations that only use single complex gates result in a less strict timing assumption. Here the weakened constraint of $dS_{min} > dT_{max}$ is sufficient to guarantee correct circuit operation, where $dT_{max}$ will commonly be the maximum delay of a simple inverter.

## 6 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. It uses a state graph interface which is familiar to both synchronous and asynchronous design engineers, and automatically produces a transistor schematic suitable for implementation.

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, state graphs lack the formalisms required to analyze compositions of these circuits for safety, liveness, deadlock, and other properties. A process calculus is currently being investigated as a means of specifying and generating MEAT state graphs as well as proving correct operation and construction. Lastly, the timing inequality is dependent on device parameters and circuit technology. A tool which can automatically analyze circuits for worst-case conformance to this inequality and to the stability assumption will greatly increase the confidence of implementations where all hazards cannot be removed with complex gates and input orderings.

## 7 Acknowledgments

This work could not have been possible without the help of a large set of people, and many of the results are not solely due to the author. In particular, Al Davis and Bill Coates at Hewlett Packard Laboratories in Palo Alto were instrumental in the development of MEAT, including much of the coding of many of its functions. Steve Nowick at Stanford University has also contributed significantly to this project through direct contributions and insightful discussions. He ported Dill's verifier to our toolset and modified it to accommodate the burst-mode timing model. He also discovered and corrected some errors in our synthesis algorithms. I would also like to thank Dick Lampman for supporting this project at Hewlett Packard.

## References

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

[2] J. A. Brzozowski and J. E. Ebergen. On the Delay-Sensitivity of Gate Networks. Technical Report 90/5, Eindhoven University of Technology, July 1990.

[3] Henry Y. H. Chuang and Santanu Das. Synthesis of multiple-input change asynchronous machines using controlled excitation and flip-flops. *IEEE Transactions on Computers*, C-22(12):1103–1109, December 1973.

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

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

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

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

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

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

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

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

[12] 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.

[13] Cho W. Moon, Paul R. Stephan, and Robert K. Brayton. Specification, Synthesis and Verification of Hazard-Free Asynchronous Circuits. Technical Report UCB/ERL M91/67, Univ. of California at Berkeley, August 1991.

[14] S. M. Nowick and D. L. Dill. Synthesis of Asynchronous State Machines Using a Local Clock. In *1991 IEEE International Conference on Computer Design: VLSI in Computers and Processors.* IEEE Computer Society, 1991.

[15] Steven M. Nowick and David L. Dill. Exact Two-Level Minimization of Hazard-Free Logic with Multiple-Input Changes. In *1992 International Conference on Computer-Aided Design (ICCAD-92).* IEEE Computer Society, 1992.

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

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

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

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

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

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