Automatic Synthesis of Fast Compact Asynchronous Control Circuits

Al Davis, Bill Coates, and Ken Stevens

Hewlett-Packard Laboratories, P.O. Box 10490, Palo Alto, CA 94303, USA

Abstract

A tool called MEAT is presented which automatically synthesizes transistor level, CMOS, asynchronous control circuits from finite state machine based specifications. MEAT has been used to synthesize the control circuits for an asynchronous 300,000 transistor communication coprocessor.

Keyword Codes: B.6.1; B.6.3; B.7.2
Keywords: Logic Design, Design Styles, Design Aids; Integrated Circuits, Design Aids

1 Introduction

Design time is heavily influenced by the quality of the design tools. Until recently little in the way of CAD support has been available to the asynchronous designer. MEAT is a tool which synthesizes high performance control circuits in the form of a CMOS transistor schematic from finite state machine based specifications. The finite state machine basis was chosen since it is a common and familiar method. The result is that an experienced synchronous circuit designer will notice a minor conceptual shift in order to use MEAT in the creation of an asynchronous design. MEAT is by no means complete or totally original. Prior to getting into the details, it is worthwhile to characterize the terminology and the design space.

Notational Comment: We use the terms asynchronous and self-timed synonymously. All asynchronous styles are fundamentally concerned with the synthesis of hazard free circuits under some timing model. *DI* (delay-insensitive) circuits exhibit hazard free behavior with arbitrary delays assigned to both the gates and the wires, and *SI* (speed-independent) circuits are hazard free with arbitrary gate delays but assume zero wire delays.

There are a large number of rather different design styles in today's asynchronous design community. One partition of design styles can be based on the type of asynchronous circuit target: locally clocked [13,8], delay-insensitive [3,21], or various forms of *single*- and *multiple*- input change circuits [19]. Yet another distinction can be made on the nature of the control specification: graph based [22,4], programming language based [3,21,1], or finite state machine based [13,8]. For the finite state machine based styles, there is a further distinction that can be made based on the method by which state variables are assigned [9,17]. The design style space is large and each design style has its own set of merits and demerits.

The methods which produce delay-insensitive circuits, while not perfect [2], are the most tolerant of variations in device and wire delays. We chose to slightly expand the domain of timing assumptions which must remain valid to retain hazard free implementation since this permits higher performance implementations at the expense of reduced operational tolerance. Our view is motivated by the reality that our designs have to meet certain performance requirements. For any given layout and fabrication process, we have models which predict the speeds of the wires and transistors for the desired operational window. We also know the percentage of error that can be tolerated in those predictions. We could not live with arbitrary delays for performance reasons and MEAT has therefore been designed to insure hazard free operation under sets of timing assumptions that can be verified as being within acceptable windows of fabrication and operational tolerance.

In order to achieve the necessary hazard free asynchronous finite state machine (*AFSM*) implementation, it is necessary to place constraints on how their inputs are allowed to change. The most common is the *single input change* or SIC constraint [19]. SIC circuits inherently require state transitions after each input variable transition. In cases where the next interesting behavior is in response to multiple input changes, the circuit response will be artificially slow, 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 [19,5] but either required input restrictions or implementation techniques that were unsuitable for our purposes. Consequently we developed the *burst-mode* design style which permits constrained MIC signalling, and our implementation method does not require performance inhibiting local clock generation or flip-flops.

Informally burst-mode state machines are Mealy machines which respond to an input burst by making a state change and producing an output burst. As with SIC AFSM's, there is an implicit fundamental mode assumption which requires that the AFSM be given sufficient time to stabilize prior to the arrival of another input burst. In practice, this is not a severe constraint, as often some variable in the output burst enables the subsequent input burst. More specifically an input burst is the transition of all of a set of input variables. These signals must always change in a known trajectory, going either high or low as indicated in the specification, but may change in any order and at any time. A specification is illegal, if from any given state there are two exit paths one of which is a proper subset of the other. It is also illegal to specify a state machine machine where a given variable is not required to strictly alternate trajectory directions. The work of Ken Yun and Steve Nowick [13,12] provides other burst-mode synthesis methods.

This paper presents the MEAT synthesis tool, which has proven its ability to greatly reduce design time while also generating compact, high-performance, asynchronous circuits. MEAT synthesizes a verifiably correct, hazard free implementation of the design to produce a *complex gate* CMOS transistor level schematic. A complex gate is a fully complementary CMOS function which implements the sum of products equations that describe the implementation. MEAT has been used to develop a control intensive multicomputer communication chip called the *Post Office* [15]. The Post Office contains 300,000 transistors and has an area of 11×8.3 mm in the 1.2 micron MOSIS CMOS process.

2 MEAT - a Tool for Control Circuit Synthesis

Asynchronous circuits are specified for MEAT as a burst-mode Mealy state machine. This style of specification provides a powerful way to encapsulate concurrency, communication, and synchronization in a natural manner. Burst-mode state diagrams are reasonably compact when compared to petri-nets, m-nets, STG's, and other graphical representations. These diagrams work well for transition (2 cycle) or level-mode (4 cycle) signalling protocols. The first automated task performed by MEAT is to generate a primitive flow table [19] from the textual AFSM specification. This is a two-dimensional array structure which captures, in a more detailed form, the behavior represented by the state diagram. Each row of this table represents a node in the state diagram; each column represents a unique combination of input signals. Each entry in the table thus represents a position in the possible state space of the FSM.

For each entry, the value of the output signals and the desired next state may be specified. 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*. 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*. All of our specifications are given in *normal form*, that is, each unstable entry in the table must lead directly to a stable state.

Each allowed input burst will result in a particular path through the FSM state-space, starting at the stable entry where the burst begins. Other entries in the same row may be visited during the course of the input burst. In order for MIC behavior to be correctly represented, it must be guaranteed that the circuit will remain stable in the initial row until the input burst is complete. This is an important point and is a cornerstone of the burst-mode methodology. In essence, any minterm formed from input variables which can be reached during the course of an input burst must be *covered* by a stable entry in the flow table. The minterm defined by the completion of the burst will correspond to an unstable state which will cause a transition to the target row and fire the output burst.

The output burst, if any, may occur concurrently with the state change, or can be constrained to happen *after* the state change has occurred. To allow the flexibility for the later synthesis stages to choose either option, 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 *Single Transition Time – implies that each state variable will change at least once*, the monotonicity of output voltage changes is guaranteed, regardless of whether the value of a given transitioning output in an unstable entry is mapped to logic level zero or one.

Any entry in the flow table not reachable by any allowed sequence of input bursts is labeled as a *don't care* and can take on any value for the outputs or next-state values. For output bursts, it is not immediately evident which values will lead to the simplest circuit. Therefore, the assignment of specific values to the don't care entries is deferred for as long as possible. The inclusion of these don't cares can significantly simplify state reduction and boolean minimization, and also lead to more compact circuits.

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. This involves first calculating 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.

This is essentially the well-known state-reduction problem; unfortunately complications are introduced due to the MIC nature of the input bursts. "Traditional" methods normally apply only to SIC circuits, and when used for our burst-mode specifications may produce hazards in the final implementation.

Nowick et. al.[11] have developed the modifications necessary to the state-reduction and subsequent synthesis steps to guarantee that the resulting implementation will be hazard-free under burst-mode conditions. These modifications are not presently incorporated into MEAT. Currently we use a verifier [6] on the synthesized implementation. The verifier has been modified to operate with explicit timing assumptions. Hazards detected in the implementation are then reviewed to see if the circuit would exhibit correct behavior under reasonable delay assumptions. If these assumptions fall within acceptable bounds of fabrication and operational constraints then the timing assumption is entered into the verifier. If an unacceptable assumption is required then the circuit is fixed either by manual repair or by modifying the state-machine specification. The manual repair usually involves the addition of appropriate inverter chain to delay the race critical path.

The final choice of minimized states is an example of the *binate covering* problem. There are three constraints on this choice. First, and obviously, only compatible states may combined (*compatibility* constraint). Second, each state in the original design must be contained in at least one of the reduced states (*completeness* constraint). Third, selecting certain sets of states to be merged may imply that other states must also be merged (*closure* constraint). Grasselli and Luccio [7] have developed a tabular method for determining a closed cover of states, which is also in the process of being incorporated into MEAT. At present, MEAT requires the user to manually determine and enter a state covering. If any of the necessary constraints are not satisfied, MEAT will inform the user that the covering is invalid.

A new flow table representing the behavior of the minimized FSM is then generated by merging the specified rows of the original flow table. It should be noted that it is not always true that minimizing the number of states will simplify the hardware or increase performance. However, a reduced state machine can result in fewer state variables which in most cases does indeed result in a smaller and faster implementation.

A set of state variables must then be assigned to uniquely identify each row of the reduced flow table. These state variables are used as feedback signals in the final circuit. In contrast to synchronous control logic design, state codes may not be randomly assigned, but must be carefully chosen to prevent races. The MEAT state assignment algorithm is based on a method developed by Tracey[17]. The Tracey algorithm has the advantage that it produces STT state assignments which minimizes delay in the implementation. In cases where two or more state variables must change value when transitioning to a new state, all variables involved are allowed to change concurrently, or *race*. It must be guaranteed that the outcome of the race is independent of the order in which the state variables actually transition in order to produce a *non-critical* race which exhibits correct asynchronous operation. Several valid assignments may be produced, and each will be passed to the next stage for evaluation. Each state assignment will result in a unique implementation.

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 care entries typically present in the flow table causes the standard algorithm to be rather inefficient and increases the likelihood that more than one minimal expression will be found. The MEAT implementation contains optimizations for don't care dominant functions. Each possible solution is given a heuristic "weight" that indicates the expected speed and area cost of implementation using complex CMOS gates. When multiple state assignments have been produced in the previous step, the total weight of each unique SOP (sum of products) equation is then used to choose between the various instantiations.

The minimized equations produced in the previous step are then used to automatically generate transistor netlists (suitable for simulation) and a schematic. 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 a SOP equation is *folded* into a complex gate, the number of logic levels required to generate the output can be reduced. If the function is too large to be implemented as a single module, it can easily be broken up into a tree of complex gates with 2 or more logic levels, but better overall performance. Typical state machine implementations have response times between 3 and 5 2-input NAND gate delays.

Our complex gate design generates negative logic outputs (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 performance feature, since sizing the final inverter to match the specific load can be used to reduce rise and fall times. When outputs need to drive a large load, a buffer tree can be used.

It is important to note that while the verifier searches for hazards on the AND/OR logic equations produced by MEAT, the netlist complex gate synthesis is done in a manner which does not introduce new hazards.

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 large gain is not required.

3 Design Issues and Examples

Rather than presenting a series of complex designs, we will present a number of design vignettes which illustrate interesting points in the design space, and an example of MEAT usage.

3.1 Using Burst-Mode to Increase Performance

Burst-mode assumes that inputs and outputs are generated as discreet sets, or bursts. In general, this violates delay-insensitive and speed-independent assumptions. For example, assume that an input burst has completed, and the resulting output burst causes several outputs to be generated. One of the outputs could be generated before the others. This output can be received by a destination module which could in turn generate an output which is fed back as an input to the original module even *before* the rest of the outputs have been generated. This violates burst-mode operation as the next input burst has occurred before the previous output burst has completed. Burst-mode assumes that all outputs in the burst must be generated before the environment can respond to the output burst or computation interference may occur. The cases where computation interference can occur can be flagged and checked by circuit timing analysis.

If an input burst changes an internal state variable, speed-independent operation will generally require the state variable to stabilize before the output can be changed. Performance can be improved if outputs can change concurrently with state changes. MEAT accomplishes this by making the transitioning output a don't care in the unstable exit point of a row in the flow table. This places a priority on logic minimization, but usually will produce a circuit which can generate an output concurrent with state changes. The fundamental mode assumption guarantees that the AFSM is ready to accept the next input burst when it arrives, as the state variable transition has completed and the logic has stabilized. Unger has shown that it is possible to improve on this model [18], although his method is not presently incorporated into MEAT.

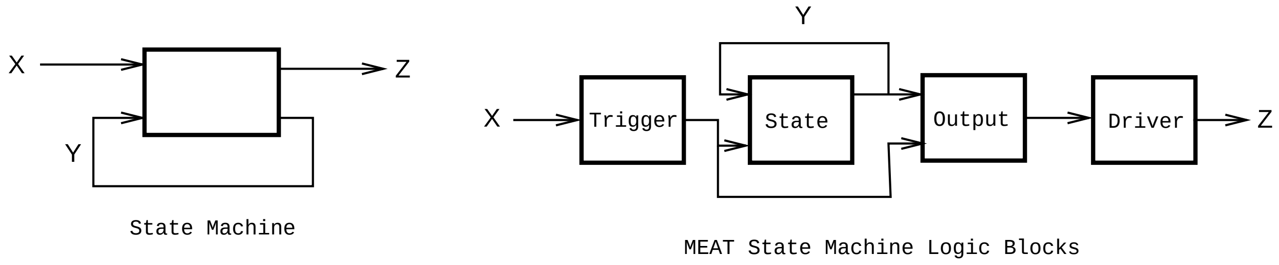

Figure 1: State Machine Generation

3.2 When Speed-Independent Circuits Fail: The Isochronous Fork

The performance enhancing local timing assumptions that MEAT supports are best exploited when they are constrained to a fixed physical extent, as is the case with individual AFSM modules. Hierarchical composition of these modules can then proceed under delay-insensitive rules since all of the external interfaces should be designed avoid timing assumptions. Inside an AFSM, the relative delay of wires and gates can be more easily controlled, analyzed, and modified as the constraints are all local. When these timing assumptions apply outside an individual module then the entire system must be analyzed to assure compliance with the timing assumption set. If this were the case, then there would be little to distinguish the circuit from a synchronous one.

A common performance and synthesis assumption made by many asynchronous circuit designers is that of speed-independence. The assumption that wire delay is zero leads to the *isochronous fork assumption*. This implies that multiple devices driven by a single component react to the signal change at approximately the same time. This model works well for situations where the transistors are slow and the paths are fast. Unfortunately this model becomes less valid as IC technology progresses and is certainly suspect even today.

Furthermore, whenever the rise or fall time of an isochronous fork is greater than the switching delay of any physical device, failure may occur due to variances in switching thresholds. Noise, long wires, and high-capacitance paths exacerbate the problem. Within a particular AFSM module, this problem can be managed successfully but between modules it is difficult. Martin [10] and Van Berkel [20] have both described circuit failures due to paths which did not behave in an isochronous fashion. Both failures were the result of using C-elements in module interfaces. C-elements inherently contain an isochronous fork. Namely the output of the C-element will be an output of the module as well as being fed back locally to maintain the C-element's state.

The philosophy we have used in the MEAT tool and in the design of our circuits is to remove isochronous forks from external interfaces. MEAT state machines are implemented as shown in Figure 1. Our philosophy is that we would rather increase the cost and difficulty of designing modules if it can simplify the composition of systems. Timing assumptions are always easier to analyze and fix in a small, local cell rather than across a series of modules. Systems are hard to design and low-level modules are relatively easy. If by making the module design harder, it becomes easier to do the inherently complex task then the overall difficulty is reduced.

The trigger box has two functions. First, high capacitance inputs (inputs with a slow rise time) will be passed through an inverter or Schmitt trigger. This will reduce the load on the input line, which can increase circuit performance. It also results in crisp rise and fall times of signals internal to the AFSM. 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. While this method does not remove the isochronous forks, it does permit them to be localized. In this case the isochronous forks created by sharing the inverters are easily controlled within the AFSM domain. Components within a particular AFSM are physically close. Hence wire delays of the internal signals and the trigger box delay are normally insignificant.

The driver block is used to generate positive output voltage levels and to increase the signal strength when the output is heavily loaded. Circuit performance is enhanced since it is sized to drive its output load appropriately. Isochronous forks in MEAT will only exist when a state variable is used directly as an output. In such cases, the output can be buffered by one or two inverters to assure the fork is isolated within the AFSM. While this decreases the performance of the circuit, the module can function in a delay-insensitive manner and can be safely used without analyzing it's load in a broader context.

This design style has been tested continuously over the last five years. We have designed several large asynchronous circuits which have generally worked the first time, merely using simulators to verify correct composition of the modules. The result of this experience has led to a high confidence factor in the method.

3.3 An AFSM example

In order to illustrate exactly what MEAT does, we will transcribe an actual synthesis run using MEAT to create a Post Office state machine called the SBUF-SEND-CTL. The behavior is initially specified as a burst-mode AFSM as shown in Figure 2. This example is taken from the suite of Post Office state machines publicly available for use by other researchers [14,13]. The specification of sbuf-send-ctl is textually entered for MEAT as follows:

```
:fsm sbuf-send-ctl
:in  (Deliver Begin-Send Ack-Send)            ;list of input variables
:out (Latch-Addr IdleBAR Send-Pkt)            ;list of output variables
:state  0 (Deliver)
        1 (IdleBAR * Latch-Addr)
:state  1 (Deliver~)
        2 ()
:state  2 (Begin-Send)
```

```
        3 (Latch-Addr~)
:state  3 (Begin-Send~)
        4 (Send-Pkt)
:state  4 (Ack-Send)
        5 (Send-Pkt~)
:state  5 (Ack-Send~)
        0 (IdleBAR~)
:state  4 (Deliver)
        6 ()
:state  6 (Deliver~ * Ack-Send)
        7 (Send-Pkt~ * Latch-Addr)
:state  7 (Ack-Send~)
        2 ()
```

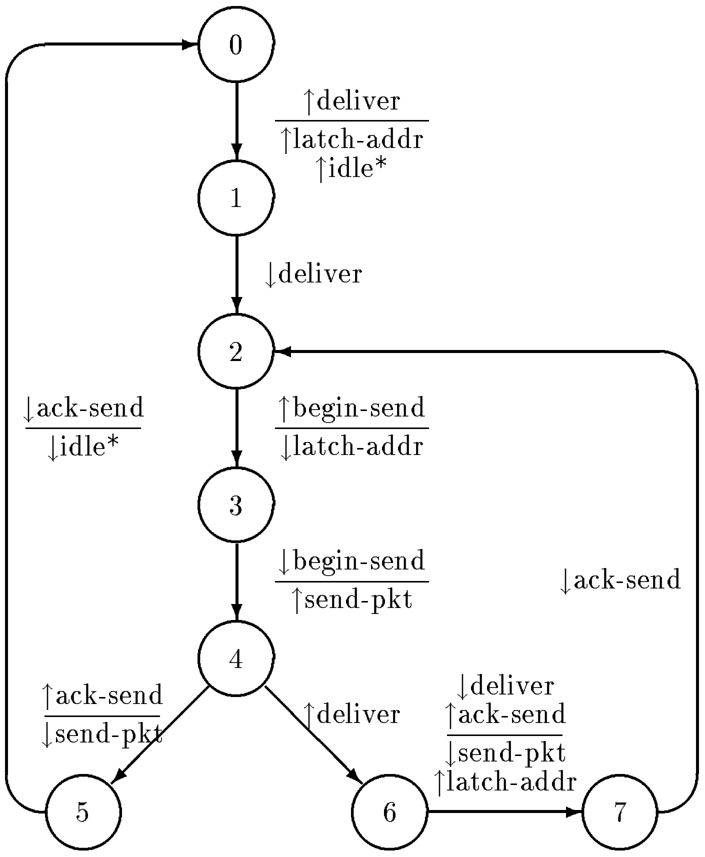
0
↑deliver / ↑latch-addr ↑idle*
1
↓deliver
2
↓ack-send / ↓idle*
↑begin-send / ↓latch-addr
3
↓begin-send / ↑send-pkt
↓ack-send
4
↑ack-send / ↓send-pkt
↑deliver
↓deliver ↑ack-send / ↓send-pkt ↑latch-addr
5
6
7

Figure 2: Sbuf-send-ctl State Machine

The following is a transcript from a MEAT session. The specification resulted in a single implementation with two state variables.

```
> (meat "sbuf-send-ctl.data")
```

```
Max Compatibles: ((0 5) (1 2 7) (3 4) (6))
Enter State set: '((0 5) (1 2 7) (3 4) (6))

SOP for "Y1":
  18: DELIVER + Y1*BEGIN-SEND~
SOP for "Y0":
  28: BEGIN-SEND + Y0*ACK-SEND~ + Y0*DELIVER
SOP for LATCH-ADDR:
  12: Y1*Y0~
SOP for IDLEBAR:
  30: ACK-SEND + BEGIN-SEND + Y0 + Y1
SOP for SEND-PKT:
  12: Y0*BEGIN-SEND~
 HEURISTIC TOTAL FOR THIS ASSIGNMENT: 100
```

The implementation can then be verified for hazard-free operation by the verifier. The verifier reads the specification and implementation. For this example, the state variables and outputs generated by MEAT are implemented as two-level AND/OR logic. Each signal is generated independently of the others. Only direct inputs are shared, so the same inverted signal in different output logic blocks will use separate inverters. Separate inverters will result in verification errors in the burst-mode speed-independent analysis. In this example, the $\overline{\textit{begin-send}}$ signal is shared by *Y1* and *send-pkt*. The two inverters are merged and the output is forked to both logic blocks. This implementation is then verified. The verifier points out a d-trio hazard [19] which is removed by adding an inverter to change the sequencing of begin-send into the *Y0* logic. The implementation is then verified as hazard free as follows:

```
> (verifier-read-fsm "sbuf-send-ctl.data")

Max Compatibles: ((0 5) (1 2 7) (3 4) (6))
Enter State set: '((0 5) (1 2 7) (3 4) (6))

> (setq *impl* (merge-gates '(1 11) *impl*))
> (verify-module *impl* *spec*)
10 20 30 40 50
Error:  Implementation produces illegal output.

> (setq *impl* (connect-inverter 10 6 *impl*))
> (verify-module *impl* *spec*)
10 20 30 40 50 60 70 79 states.
T
```

The canonical SOP equations generated by MEAT are then transformed into complex gates for implementation. The CMOS circuit for *Y0* is shown in Figure 3.

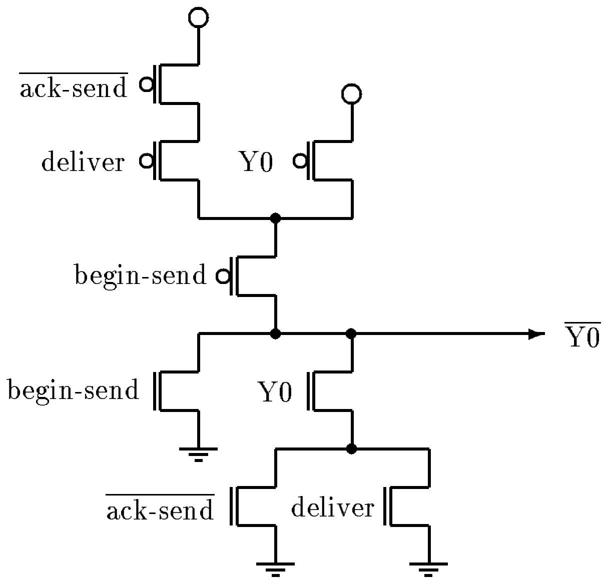

Figure 3: Complex CMOS Gate for sbuf-send-ctl Y0

3.4 D-Trio Hazards, Assumptions, and Possible Elimination

Figure 4 shows a static *d-trio* or nonessential function hazard which is found in some of the state machines produced by MEAT. D-trio hazards are fundamental and cannot be removed in every case, but they will be detected by the verifier In this cases the hazard occurs because the input burst resulted in an internal state change while the output burst contained no transition for the *Done* signal. The d-trio hazard in this example can produce a static 1-hazard on the *Done* signal. The input burst is perceived by the *Done* output logic after the state change burst thereby creating the hazard.

The $W8$ signal of the logic with the d-trio also contains an isochronous fork. If we ignore the potential threshold deviations then timing analysis shows that the physical behavior will not exhibit the hazard. However, this circuit cannot be included in a system without analyzing the driver, load, and stray capacitance on the $W8$ input or errors will result.

By modifying the trigger logic in the Sendr-Done state machine shown in Figure 4, we can both eliminate the d-trio hazard and the external isochronous fork. This incurs *no* performance penalty. The $\overline{W8}$ signal to the *Done* logic remains delayed by a single inverter, while the $W8$ signal to the state logic becomes double inverted rather than fed directly into the logic from the input.

The double inversion has the effect enforcing correct *sequencing* of the order of arrival of the $\overline{W8}$ signal to the *Done* logic. Transitions on $\overline{W8}$ will always be perceived by the *Done* logic before changes in the state variable, resulting in hazard-free circuit operation. Transitions are ordered such that the assertion of the state variable is not critical to the performance of the circuit, so the double inversion of $W8$ into the state logic has no

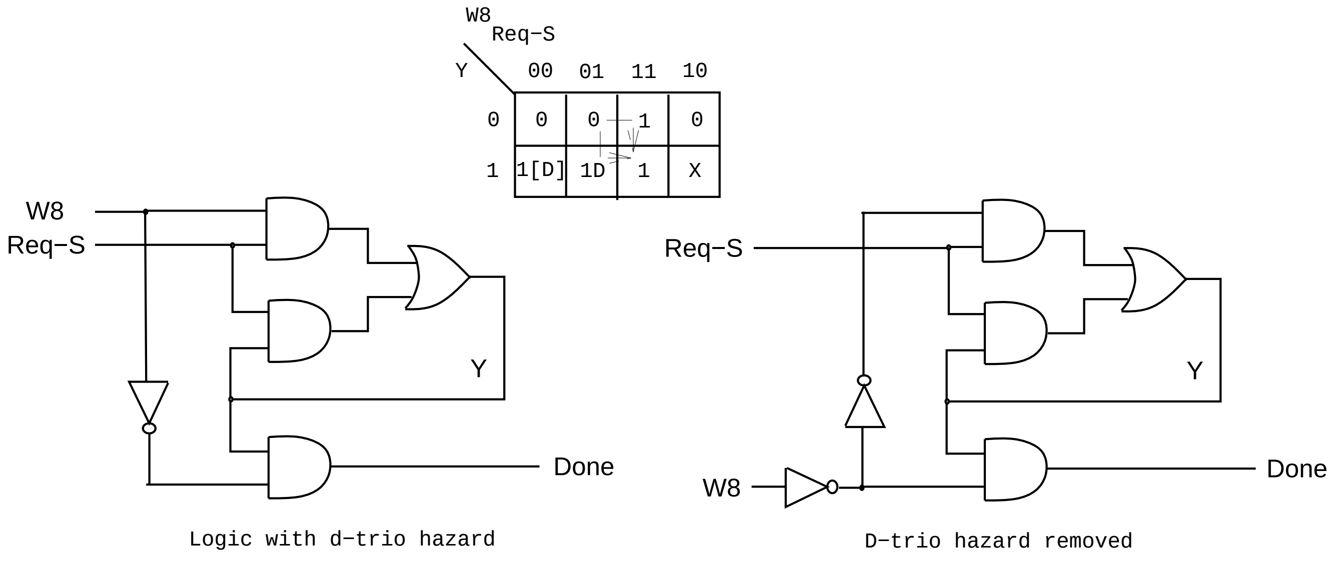

Figure 4: Hazard removal from "Sendr-Done" state machine

deleterious effect.

4 Summary

The goal in the development of the MEAT tool was to generate fast, compact, efficient, asynchronous control circuits. Demonstrating the performance advantages of asynchronous circuits is important if the discipline is to attract commercial attention. Our Post Office design methodology was no exception; as long as the circuit was fast nobody cared how we did it except us. We view this as a sad reality, since asynchronous design has other benefits as well as a conceptual elegance.

Building a large, fully self-timed circuit has resulted in many insights. The need for synthesis and analysis tools that compare with those available to the synchronous design community is of primary importance. We hope that MEAT is a step in the direction of attracting more broad based interest. We have publicly offered both the MEAT tool and many of the Post Office state machines to the IC CAD design community in hopes that others will improve on this step. The need for more robust circuit behavior and for higher performance levels is ubiquitous.

MEAT, like any CAD tool, is incomplete. The backend only produces schematics. Manual layout is prohibitively time consuming. Some form of automatic layout is necessary unless we abandon the complex gate approach and take advantage of standard cell and technology mapping approaches. Automatic layout is a difficult task and should also include automatically sized transistors for the performance needs of the design. Using standard cells will result in some lost performance but the synthesis task is easier. We are investigating both options.

There are other performance oriented factors that should be included. As a design is passed down through the different stages of MEAT, some information is lost. The complexity of the algorithms and simplicity of the circuits could be enhanced by preserving some of this information. State graphs lack the formalisms required to analyze compositions of these circuits for safety, liveness, deadlock, and other properties. We are currently investigating a process calculus as a means of specifying and generating MEAT state graphs as well as proving correct operation and construction. MEAT also needs to be connected to existing CAD tools. An example is the connection to a timing analyzer so that the timing assumptions can be automatically analyzed for compliance.

Most importantly MEAT only generates control circuits. For the Post Office experience we assumed that the datapath design would be similar to synchronous methods. We found out the hard way that this is not the case. Hence there is a significant need to adapt existing datapath generators to accommodate asynchronous methods such as micropipelines [16]. We also need to integrate the MEAT capability with the rest of our CAD suite since at present there is too much user interaction. Part of this has already been done, but the integration process is incessant by nature.

Approximately a fifth of the Post Office control path design was done manually, and the rest was done using MEAT. The automated part of the design took one-fourth the amount of design time and was virtually error free. Those errors were corrected when Steve Nowick pointed out a flaw in our minimization algorithms. Our design style has proven to be a very natural transition for existing hardware designers, primarily since it is based on traditional finite state machine control. Our synthesis techniques have generated compact high-performance circuits that work, and the complexity of the synthesis algorithms has proven to be viable for large designs.

References

[1] Erik Brunvand and Robert Sproull. Translating Concurrent Programs into Delay-Insensitive Circuits. In *IEEE International Conference on Computer Aided Design: Digest of Technical Papers*, pages 262–265. IEEE Computer Society Press, 1989.

[2] J. A. Brzozowski and J. C. Ebergen. On the Delay-Sensitivity of Gate Networks. *TC*, 41(11):1349–1360, November 1992.

[3] Steven M. Burns and Alain J. Martin. *The Fusion of Hardware Design and Verification*, chapter Synthesis of Self-Timed Circuits by Program Transformation, pages 99–116. Elsevier Science Publishers, 1988.

[4] Tam-Anh Chu. On the models for designing VLSI asynchronous digital systems. Technical Report MIT-LCS-TR-393, MIT, 1987.

[5] 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.

[6] David Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits. An ACM Distinguished Dissertation.* MIT Press, 1989.

[7] A Grasselli and F. Luccio. A Method for Minimizing the Number of Internal States of Incompletely Specified Sequential Networks. *IEEE TEC*, June 1965.

[8] A. B. Hayes. Stored State Asynchronous Sequential Circuits. *IEEE Transactions on Computers*, C-30(8), August 1981.

[9] Lee A. Hollaar. Direct implementation of asynchronous control units. *IEEE Transactions on Computers*, C-31(12):1133–1141, December 1982.

[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, *Advanced Reserach in VLSI: Proceeedings of the Decennial Caltech Conference on VLSI*, pages 351–373. MIT Press, 1989.

[11] 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.

[12] S. M. Nowick, K. Y. Yun, and D. L. Dill. Practical asynchronous controller design. In *1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*. IEEE Computer Society, 1992.

[13] Steven M. Nowick and David L. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In *1991 IEEE International Conference on Computer-Aided Design*. IEEE Computer Society, 1991.

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

[15] Kenneth S. Stevens, Shane V Robison, and A.L. Davis. "The Post Office – Communication Support for Distributed Ensemble Architectures". In *Proceedings of 6th International Conference on Distributed Computing Systems*, pages 160 – 166, May 1986.

[16] Ivan Sutherland. Micropipelines. *CACM*, 32(6):720–738, June 1989. Turing Award Lecture.

[17] J. H. Tracey. Internal state assignments for asynchronous sequential machines. *IEEE Transactions on Electronic Computers*, EC-15:551–560, August 1966.

[18] S. H. Unger. A Building Block Approach to Unclocked Systems. In *Proceedings of the 26th HICSS Conference*, January 1993. To appear.

[19] S.H. Unger. *Asynchronous sequential switching circuits.* Wiley-Interscience, 1969.

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

[21] C. H. (Kees) van Berkel. *Handshake circuits: an intermediary between communicating processes and VLSI.* PhD thesis, Technical University of Eindhoven, May 1992.

[22] Peter Vanbekbergen, Francky Catthoor, Gert Goossens, and Hugo De Man. Optimized synthesis of asynchronous control circuits from graph-theoretic specifications. In *International Conference on Computer-Aided Design.* IEEE Computer Society Press, 1990.