The Post Office Experience: Designing a Large Asynchronous Chip

Al Davis
Bill Coates

HP Laboratories
Palo Alto, CA 94019

Ken Stevens

Computer Science Department
University of Calgary
Calgary, Alta T2N 1N4

Abstract

The Post Office is an asynchronous, 300,000 transistor, full-custom CMOS chip designed as the communication component for the Mayfly scalable parallel processor. Performance requirements led to the development of a design style which permits the design of sequential circuits operating under a restricted form of multiple input change signalling called burst-mode. The Post Office complexity forced us to develop a set of design tools capable of correctly synthesizing transistor circuits from state machine and equation specifications, and capable of verifying the correctness of the resultant circuitry using implementation specific timing assumptions. The paper provides a case study of this design experience.

1 Introduction

The Post Office was designed to support internode communication for the Mayfly parallel processing system[8]. The Post Office handles all of the physical delivery aspects of packet communication. This includes local buffering, dynamic adaptive routing and congestion avoidance, deadlock avoidance, and virtual cut-through. The Mayfly topology was designed to be extensible and permits an unbounded number of PEs to be interconnected. This implies that the *physical extent* of the system is not fixed and poses serious problems when considering an implementation strategy which uses a common global clock. Clock skew is a possible headache for any synchronous design style, and is magnified as technology progresses[1]. In the case of extensible systems such as Mayfly, where the total number of boards is unbounded, the synchronous choice becomes intractable. We therefore chose an asynchronous design style for the Post Office implementation.

Another critical design constraint was the need for a high performance implementation, since message passing performance would be critical to the success of the Mayfly system. Proponents often argue that asynchronous circuits are inherently faster since they are controlled by locally adaptive timing rather than the usual global worst-case clock frequency constraints. While we believe that this claim has merit, we feel that in general it is misleading. Asynchronous circuits require more components to implement the same function. This may result in longer wires, increased area, and reduced performance. When compared to a very well tuned synchronous design, a functionally equivalent asynchronous implementation may actually run slightly slower. The need for speed heavily influenced our particular asynchronous design style.

Notational Comment: We use the terms asynchronous and self-timed synonymously. All asynchronous or self-timed design 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[17,10,7], delay-insensitive[12,3,23,16], or various forms of *single-* and *multiple-* input change circuits[22]. Yet another distinction could be made on the nature of the control specification: graph based[15,4], programming language based[12,23,2], or finite state machine based[17,10]. 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[11,21]. The design style space

0-8186-1060-3425/93 $03.00

is large and each design style has its own set of merits and demerits. It is worthwhile to note that virtually all of the design styles focus on the design of the control path of the circuit.

Compiled implementations based on programming language like specifications[12,3,23,2], while elegant and robust, suffer in performance because they are presently compiled into intermediate library modules rather than into optimized transistor networks. A module of significant concern is the C-element. C-elements are common circuit modules in asynchronous circuits and eliminating them completely is unlikely. C-elements are both latches and synchronization points. Too much synchronization reduces parallelism and performance.

The methods which produce DI circuits, while not perfect[13], are the most tolerant of variations in device and wire delays. This tolerance improves the probability that a properly designed circuit will continue to function under variations in supply voltage, temperature, and process parameters. 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 therefore it seems impractical to assume arbitrary delays in order to ensure hazard free operation of the circuits. Our approach has therefore been to insure hazard free operation under sets of timing assumptions that can be verified as being within acceptable windows of fabrication and operational tolerance.

We chose to pursue a finite state machine based style for two reasons: 1) the finite state machine concept is a familiar one for hardware designers like ourselves, and 2) the graph and programming language based synthesis methods that we knew about were too slow for our purposes. The finite state machine based design style does not use C-elements, although C-elements are used sparingly and in stylized ways in interface circuits such as arbiters.

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[22]. 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[22,5] but either required input restrictions or involved implementation techniques that were unsuitable for our purposes. As a result we developed a design style that we call **burst-mode** which permits a certain style of multiple input change. Our burst-mode implementation method does not require performance inhibiting local clock generation or flip-flops.

As our Post Office state machines got too complex for hand synthesis, we decided to create a tool kit that was capable of automatically synthesizing the transistor level circuits from burst-mode specifications. We call this tool kit **MEAT**. During the development of MEAT, we were fortunate to have Steve Nowick spend two summers with us. He incorporated David Dill's verifier[9] into the tool kit, and modified the verifier to accommodate our burst-mode timing model. Steve had considerable influence on our ideas and his locally clocked design style[17] is another outcome of these earlier interactions. We are indebted to Steve for his influence on our design style.

The Post Office was fabricated using MOSIS revision 6 design rules in a 1.2 micron CMOS process. The circuit contains 300,000 transistors and has an area of 11 $\times$ 8.3 mm. There are 95 AFSMs, most of which operate concurrently and which account for 19% of the chip area. Datapath circuitry takes up 45%, pads cover 11%, wire routing requires 19%, and the other 6% is unused space on the rectangular die.

Our view in the beginning of the Post Office design effort was that we could use existing tools for the datapath, but that since adequate performance oriented synthesis tools did not exist for asynchronous controllers, we would have to create such a tool. Hence the focus of our design method and the MEAT tools was directed at the control path. The goal was to synthesize optimized transistor level schematics that were fast. It was our mistaken impression that once MEAT worked, the design of such a large and complex device would be well supported by our CAD capability. What follows is a brief description of our design style, the MEAT tools, and a retrospective view of how well the design process went.

2 Design Style

The Post Office datapath circuitry was designed using a style similar to that used for conventional synchronous circuits. The differences are minor and are a direct result of the need to cooperate with the signalling protocols imposed by the AFSMs in the control path. Data is transferred between elements using a four-cycle self-timed bundled-data protocol[19]. This is a weaker model than that of speed-independence used for control signals inside state machines[14]. There are also some datapath circuits which are controlled in a *clocked* domain, rather than in a self-timed fashion. These datapath cells contain a stoppable clock or are clocked by state machine outputs. The stoppable or raw clock signals are usually burst-mode generated in parallel with an associated set of asynchronous hand-shake signals. The minimum delay of these hand-shake signals must be greater than the maximum delay required by the clocked circuitry. We use conventional timing analysis tools on these paths to insure that the assumption is correct given our expected process parameters and an error margin.

Making timing assumptions for datapath logic has two consequences. First, the logic is smaller and simpler because no completion signals are generated. Second, there is an increased potential for errors, as correct operation is now dependent on physical circuit layout, process, and environmental parameters. The result is that these timing assumptions must only be made within a module which can be subsequently verified. It is dangerous to export timing assumptions between modules since the transitive nature of the path inequalities rapidly leads to intractable analysis and verification complexity.

The control path is specified as burst-mode AFSMs and their implementation is synthesized by the MEAT tool. Burst-mode permits a *conjunctive* input burst to arrive prior to responding with an output burst. There is an implicit fundamental mode assumption similar to SIC AFSMs in that once an input burst is received, the AFSM must be given time to respond with its output burst and settle into a new stable state prior to the arrival of a new input burst. Burst-mode is a restricted form of MIC signalling best illustrated by a simple example.

For any given exit arc from a state, the arc is labelled with the input burst that will cause the transition, and the associated output burst that will be the AFSM's response. As such this looks like a traditional Mealey FSM model. Let there be 4 input variables (A, B, C, and D) and 3 output variables (R, S, and T). A state transition may be labelled $\uparrow A \uparrow B \downarrow C / \downarrow R \uparrow T$. We use a positive logic convention and hence the meaning is that if signals A and B go high and C goes low then the AFSM should result in a low going transition of R and a high transition on T. The order of the 3 input transitions is unspecified and therefore may occur in any order including concurrently. The same unspecified order applies to the output burst.

The MIC restriction is that for any set of arcs leaving a single state, completion of their input bursts must be mutually exclusive. For example, given a state with the previous state transition, another state transition from the same state could be $\uparrow A \uparrow B \downarrow D / \downarrow R \uparrow S$. However a state transition labelled $\uparrow A \uparrow B / \downarrow R \uparrow S$ would be illegal. There is no way to safely distinguish in the asynchronous world whether C is still supposed to occur or whether AFSM should just respond to changes on A and B. It would also be illegal for a transition to be labelled $\uparrow A \downarrow B \downarrow C / \downarrow R \uparrow T$ since each state inherently must look for a known transition direction for each input variable that it must respond to.

The other consistency requirement is that input and output transitions must strictly alternate, e.g. for any given directed path in the AFSM for any input variable X, $\uparrow X$ must follow a $\downarrow X$ and vice versa. Omitting X on an intervening transition implies no change. The same must be true for output variables. A corollary to this requirement is that any circuit in the AFSM description will have an even number of alternating input and output variable changes. The MEAT tools analyze the AFSM specification and signal an error if these MIC restrictions are not met. If the specification is valid then MEAT generates the logic equations which are then *folded* into a complex CMOS gate. A schematic is also produced.

3 MEAT - a Tool for Control Circuit Synthesis

The MEAT synthesis tool is fast enough that alternative design options can be explored. The designer is freed from the task of understanding the underlying transformations required to produce hazard-free asynchronous circuits. The burst-mode specification has proven to be both a natural and efficient method for specifying AFSMs. Presently MEAT does not contain a state graph editor so the graphical state machine description is then specified textually in the MEAT entry format. Each arc in the state diagram is mapped to a single statement in the text file, which indicates the source and destination states along with the associ-

ated input and output bursts.

The first automated task performed by MEAT is to generate a primitive flow table[22] from the textual AFSM specification. This is a two-dimensional array structure which captures 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 AFSM. 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**.

All rows will have a stable entry where an input burst begins. Other entries in the same row may be visited when an input burst occurs. 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. At this point, an unstable state will be entered which will cause a transition to the target row specified and fire the output burst. 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. As it is not immediately evident which values will lead to the simplest circuit, the assignment of specific values to the don't care entries is deferred for as long as possible.

The next step 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. After specifying the reduced 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.

The final choice of minimized states must be chosen by the designer. 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). If any of the above constraints are not satisfied, MEAT will inform the user that the covering is invalid.

A set of state variables must be assigned to uniquely identify each row of the reduced flow table. 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[21]. The Tracey algorithm has the advantage that it produces **Single Transition Time (STT)** state assignments. 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. 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. 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 increase the likelihood that more than one minimal expression will be found. Each equation is given a heuristic "weight" that indicates the expected difficulty of implementation and speed of operation using complex CMOS gates. When multiple state assignments have been produced in the previous step, the total weight of each unique SOP equation is used to choose between various instantiations.

The minimized equations produced in the previous step are then used to automatically generate transistor net lists, suitable for simulation, representing complex CMOS gates. A graphical schematic diagram is also produced to help guide the layout process. 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 sum-of-products equation is *folded* into a single complex gate, the number of logic levels required to generate the output can be reduced. If the function

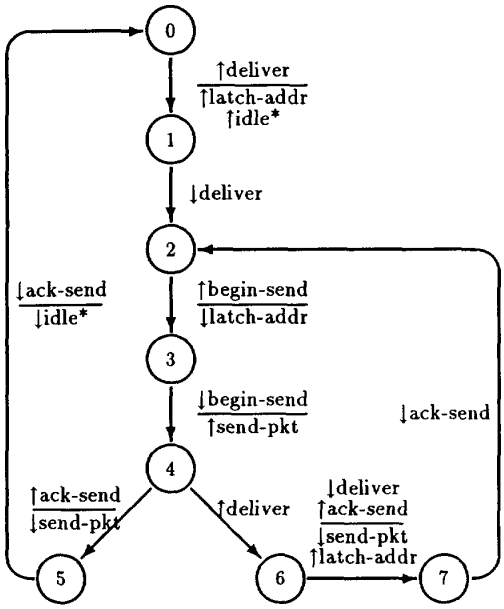

Figure 1: Sbuf-send-ctl State Machine

is complex, it can easily be broken up into a tree of complex gates with improved overall performance[20]. Typical state machines in the Post Office have an input to output delay of 2 to 5 inverter delays.

4 Design Example

In order to illustrate the synthesis process from the designers point of view we will use a Post Office state machine called *sbuf-send-ctl* as a design example. The state machine is specified in Figure 1. We have created a pool of Post Office AFSMs that we have made available to other researchers and synthesized implementations of this state machine in other methods can be found in[18,17].

The specification of sbuf-send-ctl from Figure 1 is textually entered for MEAT by describing the AFSM name, input variables, output variables, and then each state transition in 2 text lines. The first line describes the current state and the input burst, while the next line specifies the destination state and the output burst. In the text A would correspond to $\uparrow A$ and A would be equivalent to $\downarrow A$ from the graphical version of the state machine. The textual sbuf-send-ctl specification is:

```
:fsm sbuf-send-ctl
:in  (Deliver Begin-Send Ack-Send)
:out (Latch-Addr IdleBAR Send-Pkt)
: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 ()
```

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 is then 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 $\overline{begin\text{-}send}$ signal is shared by *Y1*

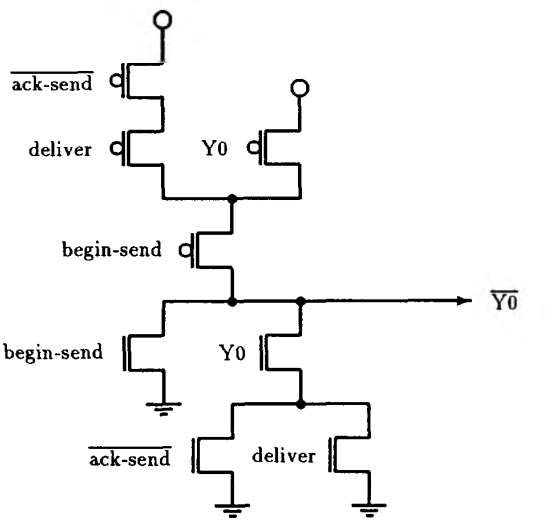

Figure 2: Complex CMOS Gate for sbuf-send-ctl Y0

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[22] which is removed by adding an inverter to change the sequencing of begin-send into the *Y0* logic. A MEAT transcript of these verification steps 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 2. The complex gates are then manually implemented using the Electric layout editor. The physical layout is then simulated with COSMOS to check for layout errors. Some minor modifications to COSMOS were required in order to simulate the entire chip.

5 In Retrospect

When we started the Post Office effort we had all been actively designing reasonably complex asynchronous hardware systems for at least 8 years, and in one case since the early 1970's. With the exception of the ISM chip[6], these systems, including full scale computers[7], were board level designs rather than chips. We had also designed a number of complex synchronous widgets as well. The experience imposed a common belief that while it was undeniably a bit harder to design and implement the AFSMs correctly, the inherent modularity of asynchronous subsystems was a tremendous advantage at the system level. What we did not appreciate was the fact that all of our previous asynchronous designs were *proof-of-concept* prototypes whose goal was to demonstrate functional feasibility rather than performance. The performance oriented Post Office project caused us to reexamine much of we considered to be standard asynchronous design practice. We also failed to appreciate some of the problems imposed by the improvement of IC technology. Some of these problems were coupled with a desire to use the scalable design rules and the convenience of fabrication through MOSIS, and the eventual reality that the project would span a period of approximately 7 years (a research politics side effect in our previous company). The net result was a feeling that the only new challenge would be the architecture of the Post Office. **We were wrong!**

The SIC AFSM methodology soon proved to be both a performance bottleneck and consumed unreasonable amounts of silicon real estate. The result was the burst-mode, localized timing assumption, and complex gate approach which has proven to be a significant improvement. The next problem was that too much of our limited manpower budget[1] was being spent in the inherently error prone manual synthesis of AFSMs. MEAT started as a *quick and dirty hack* which would correctly synthesize the AFSMs. Since we were not experienced CAD people, we grossly underestimated the need to pay attention to algorithmic complexity. On its first production test, MEAT was given an AFSM to synthesize that had already been implemented manually and the test chip was already operational. It was a small design with 18 states, 10 input variables, and 6 outputs. After a weekend and 10's of thousands of garbage collections, we stopped it to find that a few percent of the job was done. After sleepless weeks, we finally had something acceptable.

[1] Four people did all of the hardware design and implementation for the entire Mayfly system.

We then found that our SIC based logic minimization methods were not valid for burst-mode operation. We then corrected this oversight[2]. The MEAT capability gave us the performance leverage that we felt we needed. By using locally verifiable timing assumptions to increase the performance within well contained modules but by requiring modules to interact with each other in a speed-independent fashion, a reasonable design balance exists between performance and modularity. The use of bundled datapath protocols between modules reduces the wiring area budget but precludes DI behavior.

AFSM module speeds and area costs are excellent. In order to illustrate the savings we compare a state machine called *MP-Forward-Pkt* in our design style with an equivalent implementation that compiles the specification to library modules. In order to factor out the additional benefits of the complex gate approach we compare the implementation of these two variants of the circuit using a straightforward AND/OR implementation. The MEAT specification is:

```
:in       (Ack-Out Ack-PB Req)
:out      (Alloc-Outbound RTS Alloc-PB Ack)
:init-out (Alloc-Outbound)
:state  0 (Ack-Out)
        1 (Alloc-Outbound~ * RTS)
:state  1 (Req * Ack-Out~)
        2 (Alloc-PB * RTS~)
:state  2 (Ack-PB)
        3 (Ack * Alloc-PB~)
:state  3 (Ack-PB~ * Req~)
        0 (Ack~ * Alloc-Outbound)
```

The implementation in logic gates is generated by MEAT as:

$$\begin{aligned} Y1 &= \text{Ack-Out} + Y1 \times \overline{\text{Req}} \\ Y0 &= \text{Ack-PB} + Y0 \times \text{Req} \\ \text{Alloc-Out} &= \overline{Y1} \times \overline{\text{Ack-PB}} \times \overline{\text{Req}} \\ \text{RTS} &= Y1 \\ \text{Alloc-PB} &= \overline{Y0} \times \overline{\text{Ack-Out}} \times \text{Req} \\ \text{Ack} &= Y0 \end{aligned}$$

The circuit verifies under our burst-mode and fundamental mode timing assumptions but will fail the test for SI behavior. Compiling the AFSM to a speed-independent circuit containing C-elements, Merge elements, and Toggle elements (indicated by triangular elements) produces the circuit in Figure 3.

The MEAT version requires 6 gates and 5 inverters while the SI version requires 63 gates and 20 inverters.

[2]Thanks to Steve Nowick for his assistance in this discovery and its eventual solution.

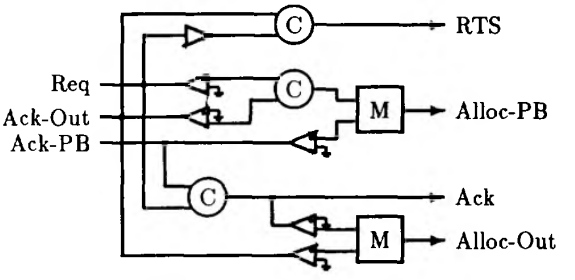

Figure 3: Speed-independent MP-Forward-Pkt Implementation

The worst case state response time of the MEAT circuit is 2 gate delays plus an inverter delay as opposed to the SI version which is 6 gate delays plus 3 inverter delays. This 3:1 speed improvement ratio changed to 2.33:1 for the average case. The difference typically increases with AFSM complexity and is substantial for a complex subsystem such as the Post Office.

The Post Office datapath components are all controlled by AFSMs. Most of these components (subtractors, counters, comparators, RAM, latches, etc.) are equivalent to synchronous designs. These circuits are typically smaller and faster than asynchronous datapath modules since they do not include circuitry for generating completion signals. The datapath modules are controlled by pulses generated as outputs by their AFSM controllers. Although these signals are not local to an AFSM, their extent is bounded by the AFSM and datapath component pair. At times, worst-case datapath delays were synthesized to handshake with the AFSM. Certain datapath components where there can be a wide variance in delays are designed to sense completion and generate acknowledgments (such as the RAM cells). The extra logic to generate completion signals from each slave operation usually will result in more logic but if carefully designed should not reduce performance.

The decision to use synchronous clocked datapath logic has not been particularly difficult nor error prone in this large circuit. However, clocked dynamic circuits potentially introduce additional failure modes. One design flaw in an original version was discovered only after it had been integrated into the completed circuit. A dynamic counter was supposed to be reset in the idle state. When this module was fabricated and tested individually the circuit was not left idle for large periods of time so the flaw was not detected. In the full Post Office the charge on the internal nodes

of these counters dissipated during extended periods where the Post Office was not needed by the rest of the system. The result was lost state in the counter and a functional failure.

While bundled data protocols save area, they cannot be routed randomly between components. The worst case delay must be analyzed to assure that the data has arrived before it is utilized. In certain cells, like the Post Office RAMs, arrival of the data can be sensed by the discharge of a precharged line on the slowest line. However, bundled data being driven directly from one source to the next, such as between the Post Office chips, rely on delay path analysis. If hardwired delays are used, and they are not sufficiently long, there is no way to repair the circuit without another fabrication cycle. There's no such thing as turning down the clock in asynchronous systems.

Bundled data can efficiently be used for slave data components. The routing logic in the Post Office is a successful example of pipelined control with bundled data. Bundled data transfers which are latched (such as between Post Office chips and internal buffers) are also reasonable applications. However, a bundled protocol should **NEVER** be used for encoding address selection and control signals on a bus. Unfortunately we had to learn this the hard way. Buses are prone to be highly capacitive and as such are slow, inefficient, and noisy. A glitch on a bused control line can easily cause an AFSM to respond incorrectly. Setup time for the bundled control signals before the enabling signal arrives becomes very critical.

While we continually tried to focus on performance, we found that performance is an elusive target. Simply counting transistor delays is a false metric. A larger design with many more devices can result in a faster circuit if the gains and capacitances at each stage are balanced[20]. As device size shrinks and doping increases, inter-node capacitance and wire lengths become increasingly critical. Fast circuits can only be achieved when the output to input load ratio is small and the device gain is high. Point-to-point communication is the best way to achieve speed. Asynchronous methods lend themselves well to concurrent, pipelined architectures if designed properly. However, the entire design *philosophy* needs to avoid inefficient shared, capacitive structures. For high performance systems buses should probably be avoided altogether. This implies a shift in architectural design styles. Driving large buses causes the vast majority of the delay in the Post Office circuitry.

The complex gates generated in MEAT have resulted in compact, fast circuits. However, care must be taken to insure that the size of these gates is small to reduce the inter-node capacitance and increase the gain. The design of such gates can introduce parasitics between the power rails and the output which can result in a large body effect. As devices continue to shrink, this can nullify or even result in slower designs using large complex gates than a series of smaller NAND gates. One complex gate advantage is the use of an inverter on the output, which provides low output loading and increased gain from the complex gate and inverter pair.

Reducing intermodule capacitance and introducing buffers between or within AFSMs will both increase reliability and performance. Slowly rising signals are subject to device threshold variances, which can cause failures when isochronous forks are used. Eliminating these slow signals will reduce instantaneous power consumption, resulting in reduced noise, and neutralize the problem of isochronous forks.

Many of our performance and reliability problems in the Post Office design were due to our blind faith attitude that the intrinsic asynchronous modularity capability would cure many ills. While it is true that we have removed the global clocks, global planning is still important. Simply cobbling circuits together in a bottom-up fashion leads to trouble, and only works well for adjacent, communicating control circuits. This is difficult when the layout is done manually. Manual layout may always be necessary for certain modules that are either heavily replicated or are on the critical performance path. However it should not be the general practice. Manual layout of a 300,000 transistor circuit is certainly insane and was a prime contributor to our capacitive woes. In any large chip floor planning is critical to the performance of the design. The problem is that the floor plan is always done first, and the plan can seldom be kept intact as the modules get implemented and do not quite conform to the plan. The problem is exacerbated due to what we will call *implementation momentum.* Namely if you just spent a hundred hours manually laying out a complex component that just misses the floor plan, there is an undeniable tendency to compromise the floor plan in order to avoid another hundred hours of layout. In designs like the Post Office containing hundreds of modules there is ample opportunity to make many such compromises. The negative impact that these compromises make on the final design performance is significant, and in the Post Office case the resultant floor plan is poor and costs us at least a factor of 2 in total performance.

In addition, power and ground signals retain their

global nature in asynchronous circuits. While metal migration is not as big a problem in asynchronous circuits due to inherent duty cycle variance, care must be taken to allow sufficient current carrying capacity to prevent noise. We unfortunately learned the hard way that as VLSI circuits are scaled down, the global power and ground lines should increase in width relative to the feature size. This change wreaks havoc yet again with the floor plan. Even with the serious floor plan flaws, and the poor judgment made using buses for interface communication, the Post Office can sustain transfer rates up to 200 MBytes per second.

Our design style virtually eliminates intra-module C-elements. C-elements can be viewed as a simple AFSM and in our design style their behavior is convolved into the controller designs in a direct way. The performance benefits of C-element removal are clearly substantial. However the Post Office does contain 54 C-elements. It is interesting to note that **NONE** of them are used alone as a protocol preserving signal rendezvous. They are **ALL** used in pairs. The standard arbiter circuit is a common example of this usage. There are two sides only one of which is active and through which the shared resource acknowledge must be passed in a protocol preserving fashion while the other side remains inactive. Only recently have we realized that in this role even these C-elements could be replaced by smaller and faster circuitry.

6 Conclusions

Building a large, fully self-timed circuit has resulted in many insights, the most important of which we have attempted to pass on. Different design styles and varying design targets will undoubtedly provide some new insights but many will be similar. The need for integrated synthesis and analysis tools that compare in quality and scope with those available to the synchronous design community is of primary importance. This is also a moving target. Some synchronous tools will work just fine, others will need only minor modifications, while still others will have to be created specifically to handle asynchronous designs. MEAT is a step in the right direction, but many more steps are necessary. Some form of automatic layout is necessary unless we abandon the complex gate approach in order take advantage of standard cell approaches. Automatic layout is a difficult task, but some performance will be lost in the standard cell approach. We are investigating both options.

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 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 composed of multiple AFSM modules.

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. 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.

The inherent modularity of asynchronous designs and their composability into larger modules makes self-timed design of large systems very attractive. The fact that they can then be incrementally improved for performance makes them even more so. A number of open challenges remain. While we have found that testing large asynchronous designs is relatively simple at the board level, it is difficult when the design is a single chip. Since our controllers do not contain latches, it is difficult to use scan paths to improve testability. Electron-beam testers do not help much because it is difficult to image hand-shake signals. We view these problems as opportunities for future research. We also hope that our experience and even our tool efforts will be of future benefit to others embarking on the design of large asynchronous system components.

References

[1] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley, 1990.

[2] 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.

[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] William S. Coates. "The Design of an Instruction Stream Memory Subsystem". Master's thesis, University of Calgary, December 1985.

[7] A. L. Davis. The Architecture of DDM1: A Recursively Structured Data-Driven Machine. Technical Report UUCS-77-113, University of Utah, Computer Science Dept, 1977.

[8] A. L. Davis. Mayfly: A General-Purpose, Scalable, Parallel Processing Architecture. *Lisp and Symbolic Computation*, 5(1/2):7–47, May 1992.

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

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

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

[12] Alain Martin. Compiling Communicating Processes into Delay-Insensitive VLSI Circuits. *Distributed Computing*, 1(1):226–234, 1986.

[13] Alain Martin. The Limitations to Delay-Insensitivity in Asynchronous Circuits. In William J. Dally, editor, *Sixth MIT Conference on Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.

[14] C. Mead and L. Conway. *Introduction to VLSI Systems.* McGraw-Hill, 1979. Chapter 7.

[15] Teresa Meng. *Synchronization Design for Digital Systems.* Kluwer Academic, 1990.

[16] Charles E. Molnar, Ting-Pien Fang, and Frederick U. Rosenberger. Synthesis of Delay-Insensitive Modules. In Henry Fuchs, editor, *Chapel Hill Conference on Very Large Scale Integration*, pages 67–86. Computer Science Press, 1985.

[17] 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.

[18] 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.

[19] I. E. Sutherland, R. F. Sproull, C. E. Molnar, and E. H. Frank. Asynchronous Systems, Volume I. Technical report, Sutherland Sproull and Associates, Palo Alto, CA, January 1985.

[20] 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.

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

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

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