The Post Office Experience: Designing a Large Asynchronous Chip

by

Bill Coates

Al Davis

Ken Stevens [1]

Hewlett-Packard Laboratories

June 7, 1993

ABSTRACT

The Post Office is an asynchronous, 300,000 transistor, full-custom CMOS chip designed as the communication coprocessor for the Mayfly scalable parallel processor. Since communication latency is a critical system performance factor, the Post Office had to be designed both for high speed operation, and to function in machine configurations containing widely varying numbers of processing elements. Performance and functional requirements led to the development of a new asynchronous design style called *burst-mode*. The Post Office complexity forced us to develop a set of synthesis tools to support this design style. The paper provides a case study of this design experience, including a description of the Post Office behavior and its architecture, the design style that we employed, a brief description of our tools, and a retrospective view of the design process.

Keywords: asynchronous, parallel processing, adaptive routing, VLSI.

[1] Presently with the University of Calgary Computer Science Dept.

1 Introduction

The Post Office was designed to support inter-node communication for the Mayfly parallel processing system[1]. 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 processing elements (or **PE's**) to be interconnected. This implies that the *physical extent* of the system is not fixed and this poses serious problems when considering a synchronous 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[2]. In the case of extensible systems such as Mayfly, where the total number of PE 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 some 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[3,4,5], delay-insensitive[6,7,8,9], or various forms of *single-* and *multiple-* input change circuits[10]. Yet another distinction could be made on the nature of the control specification: graph based[11,12], programming language based[6,8,13], or finite state machine based[3,4]. 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[14,15]. The design style space is large and each design style has its own set of merits and demerits. It is worthwhile to note that most of these design styles focus on the design of the control path of the circuit.

Compiled implementations based on programming language like specifications[6,7,8,13], while elegant and robust, may suffer in performance since 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[16], 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 require hazard free behavior under either the SI or DI arbitrary delay model. 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 specialized 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[10]. 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 had been developed[10,17] but either required input restrictions or involved implementation techniques using generated clocks and latches that seemed too slow for our purposes. The search for an acceptable alternative led us to the development of a new asynchronous circuit design style that we call **burst-mode** [18]. Burst-mode circuits support a restricted form of MIC behavior and our implementation method does not require performance inhibiting local clock generation or flip-flops. The nature of the restriction is described in section 4.2.

As 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[19] 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[3] is another outcome of these earlier interactions.

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 AFSM's, 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 22%, and the other 3% is unused space on the rectangular 84-pin 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 the Post Office design and its architecture, our design style, the MEAT tools, and a retrospective view of how well the design process went.

2 Post Office Behavioral Description

The Post Office[20,21] is an autonomous packet delivery subsystem designed to be a coprocessor to a controlling CPU. The interconnect topology uses a wrapped hexagonal mesh to create what we call a **surface**. Each surface has a hexagonal boundary and multiple surfaces can be similarly interconnected by abutment. The Post Office is therefore a seven-ported device: six byte-wide external ports connect the Post Office to adjacent processing elements in the hexagonal mesh, and a 32-bit word-wide internal port provides a connection to the local CPU. The Post Office has been designed to permit all seven ports to be concurrently active.

The Post Office receives a packet from the CPU and delivers it over the communication network to a receiving Post Office, which in turn hands the packet to its CPU. Messages inherently vary in length, but packet lengths are fixed at 36 words and have the format shown in Figure 1. The Post Office only uses the destination address field of the packet. The usage of the remaining 3 words of the header is dictated by Mayfly software convention. The Post Office contains an adaptive routing mechanism that can dynamically calculate the correct ports through which to transfer a packet. General communications

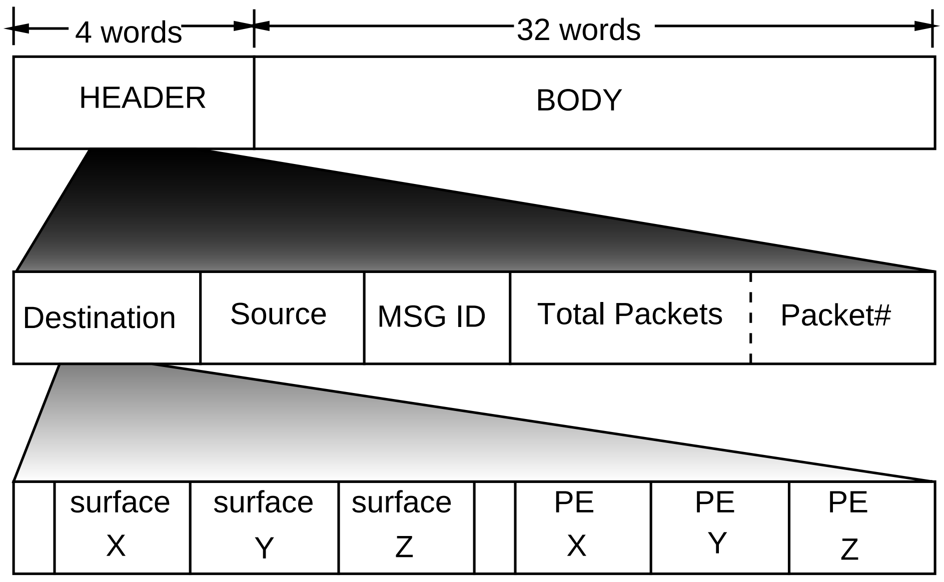

Figure 1: Packet Format

efficiency can be significantly enhanced if the routing mechanism is capable of detecting congestion and routing packets around such congested areas in the communication network. This mechanism can also be extended to permit messages to be routed around nonfunctional nodes and ports.

The capability to dynamically route packets around congestion and failed components implies that the order of packet arrival may vary from the order in which the packets were sent. The capability to reorder packets at their final destination is essential to ensure both deterministic behavior and to permit proper reassembly of multiple-packet messages by the destination CPU. This also implies that each packet contains a unique message identifier, the packet number, and the number of total packets in the message as shown in Figure 1. Although this slightly increases the amount of redundant information that must be transferred, it considerably reduces congestion related message delays. Congestion avoidance also increases the amount of packet traffic that can be in transit within the *postal* system before saturation effects are noticeable in terms of sharply increased packet latency times.

Messages are assumed to be delivered error free in our current implementation and therefore no checksums or parity checking is performed. A more production oriented version of this chip would need to provide an error detection and notification mechanism. In order to avoid deadlock, a mechanism must be employed that either guarantees deadlock will not occur, or ensures that the probability of deadlock is much less than the probability of an unrecoverable component failure. The Post Office avoids physical deadlock by preventing unrestricted incremental resource claiming, thereby ensuring that each Post Office will not be congested indefinitely. Note that live-lock is still a possibility since the adaptive routing mechanism does not include the maintenance of an *adaptation history* and therefore there is a theoretically possible chance that the postal fabric may appear full indefinitely with all of its packet traffic in stable adaptive orbit. However the probability of this situation is many orders of magnitude smaller than the probability of component failure and we therefore considered it acceptable to dispense with any of the costly solutions to this problem. Also, there is no way that the Post Office can prevent a software deadlock situation where multiple user processes are all stalled waiting on each other. From the Post Office perspective, such a software deadlock situation looks exactly the same as a situation where all PE's are busy but no messages are being sent.

For notational convenience, packets at a particular Post Office can be classified into 3 categories: transit, inbound, and outbound. Transit packets pass through a Post Office, inbound packets are destined for the local CPU, and outbound packets are created at the local node. The Post Office dynamically employs 4 types of routing algorithms: **virtual cut-through**, **best-path**, **no-farther**, and **random**. The algorithm choice depends on the specific congestion situation encountered by a particular packet. When a transit packet arrives at some external port, and an ideal external destination port of this Post Office is free, the packet will be immediately forwarded to the appropriate destination port. This method is called **virtual cut-through** [22]. The **best-path** method always takes the packet one step closer to its final destination. The **no-farther** routing is one which neither increases nor decreases the distance of the packet from its final destination but allows it to orbit intervening congestion. The **random** method sends the packet out the first available port. In this case congestion may have the current Post Office surrounded, and it may be necessary to go farther away from the destination in order to make progress.

Virtual cut-through is only available for transit packets. If an ideal forwarding port is not available then the packet is stored in a central packet buffer pool in the Post Office. Inbound and outbound packets are always stored in the buffer pool prior to being forwarded to the local CPU or out onto the postal network. On the initial attempt to route a buffered packet, the best-path method is used. If a best path port is unavailable, a **stagnation counter** is incremented. The best-path is retried until the stagnation count for a particular packet exceeds its threshold, at which point no-farther routing is also an option. Failure again increments the stagnation count. After a second threshold is exceeded then random routing is also an option. In any case where multiple available ports are found under any routing option situation, best-path will be taken first, no-farther second, and random last.

The Post Office routing algorithm is based on a labelling of PE's in the interconnected surface topology shown in Figure 2. This figure depicts what we call an E–n surface where $n = 3$, indicating that each edge of the surface contains 3 PE's.

The topology also permits equal sized hexagonal surfaces to be interconnected in the same manner. For example putting 6 more E–3 surfaces around the edges of a single E-3 results in a configuration containing 2 E–3 surfaces on each edge of the 7 surface ensemble. We call such a configuration an S–2E–3 system. Both the surface and the ensemble are extensible and utilize the same routing algorithm. Each

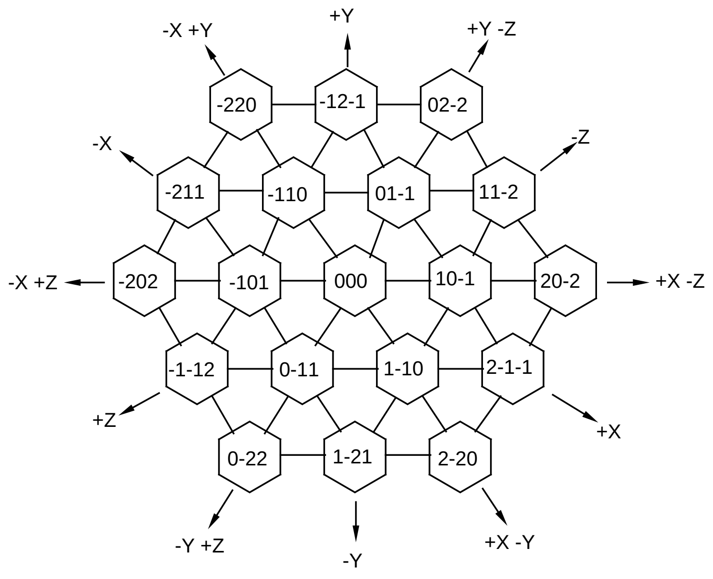

Figure 2: PE Labelling Convention

of the PE's (or surfaces) is assigned a unique location number which is a 3-tuple that corresponds to an <x,y,z> location in the surface. Note in Figure 2 that the X, Y, and Z axes correspond to the edges of the surface rather than a particular path. The reason for this choice is that it simplifies the routing algorithm. Each link traversed by a packet in either direction crosses 2 of the axes, one in the positive direction and another in the negative direction. Also note that the sum of the 3 location components is always equal to 0. This property is used to detect illegal addresses. If a message is to be sent from the center PE to any other PE in the surface it is easy to understand how to make the decision as to what Post Office port should be used. For example, suppose a message is being sent from <0,0,0> to <1,-2,1>. The relative difference is <1,-2,1>. Since the y component is the largest and the value is negative, the **best-path** must be to send the message out either port that heads in the $-Y$ direction. The **no-farther** paths are the two paths which do not contain a $-Y$ component.

The proper routing decision is much less obvious when the source PE is at the periphery of the surface, since a wrap link is a possibility. For example, the $-X, +Y$ port of PE <-1,2,-1> is wrapped to the $+X, -Y$ port of node <1,-2,1>. The relative difference between these two paths is <-2,4,-2> which does not obviously correspond to a single $-X, +Y$ move. One option would be to use a table based routing scheme. The surface topology is inherently extensible and our goal is to permit a large number of them to be interconnected. The size of a routing table is quadratically dependent on the number of PE's and would be prohibitive for larger ensembles. Therefore the Mayfly Post Office does not use routing tables and simply computes the routing decision using a method that is both simple and fast. The current implementation uses 5-bit fields for the X, Y, and Z destination fields. This choice takes advantage of the speed of 3 parallel 5-bit ALU's and is capable of correctly routing packets in configurations up to an S–16E–16 which if fully populated would contain 519,841 processing elements! The result is a Post Office element which is fast, general, and scalable.

The Post Office routing method removes the potential complexity imposed by the wrap lines by normalizing addresses to *center-based* coordinates. The method is:

1. Subtract the current packet location (the local PE address tuple) whose value is stored at initialization time in a Post Office control register from the destination address. The result is an *unnormalized* **relative address.** For an E–n surface, if one of the components of the resultant value is greater than $n-1$ then a wrap line will be needed.

   When no component in the relative address is greater than $n-1$ the address is already in the center-based format and the following four normalization steps are not taken.

2. A normalization vector is used to convert the unnormalized relative address to a *normalized* form, which makes the current PE appear to be in position <0,0,0> of the surface. The vector has the value:

   $$< 2n-1, n-1, n >$$

   For an E–3 surface, as shown in Figure 2, this value is <5,2,3>.

3. This normalization vector must then be aligned with the unnormalized relative address. The relative positions in the normalization vector must be preserved in the alignment, so a right end-around shift is used. There are two alignment conditions:

   (a) The $2n-1$ component in the normalization vector is aligned with the component of largest magnitude in the relative address when no component in the address is zero.

   (b) Otherwise the $n-1$ component is aligned with the zero component in the unnormalized relative address.

4. The sign of the $2n-1$ term of the normalization vector is set to match the sign of its aligned component. The signs of the $n-1$ and n components are the negation of the $2n-1$ component's sign.

5. The normalization vector is subtracted from the unnormalized relative address to yield the normalized relative address.

6. At this point the routing decision is simple. The **best paths** must reduce the largest magnitude component of the normalized relative address since this magnitude indicates the present distance of the packet to its final destination. In cases where there are no zeros in the normalized relative address there are two possible ports. The *preferential* best path reduces the greater of the two smaller axial components. This guarantees that at the next PE that there will be two best path

options. A zero component in the normalized relative address indicates that there is only one **best path** route. The goal is to keep multiple **best paths** options available whenever possible.

3 Post Office Architecture

The Post Office consists of a set of asynchronous control circuitry, communication links, and RAM. The control circuitry decides how to best utilize the given communication links to move packet data from local to remote RAM. Figure 3 shows the major logic blocks in the Post Office. There are five major components, the external port interface, adaptive routing and forwarding, bus arbitration block, static buffer controllers, and the internal port or PE interface. Each of these logic blocks are independent agents, and can execute concurrently with the other components. The static buffer controller consists of 19 independent controllers and an associated packet buffer, seven of which can execute concurrently.

Submodules in the Post Office communicate via the *spine bus* using a four-cycle hand-shaking protocol. Each module must obtain mastership of the spine bus before any transaction can occur. There are 29 logic blocks which can obtain bus mastership. The Bus Arbiter block centrally services bus mastership requests. Whenever any module requires bus operations, it will assert its *Bus-Request* signal. When the requesting module receives the *Bus-Acknowledge* signal, it may drive the control and data signals on the bus. The arbitration logic is centralized as it results in faster servicing of bus requests when compared to distributed schemes. However, centralization requires significantly higher wiring complexity.

Once a logic block has mastership of the bus, it may execute one or more transactions. The spine bus signals include three self-timed control lines $\overline{Req}$ $\overline{Ack}$ and $\overline{Nak}$, a three-bit opcode, an eight-bit address, a three-bit sender address, a six-bit routing result addresses, and 128 bits of packet data. Typical transactions on the spine bus are to allocate a port for packet delivery, or transmit a 128 bit **line** of packet data between two blocks.

The PE's CPU communicates with the Post Office via a 32 bit addressable register interface. The Post Office can run in either a polling or interrupt mode for either arriving and/or outgoing packets. A set of control registers can mask and disable the interrupts. There are also FIFO registers for loading and unloading packets, plus several control, status, and debugging registers.

Upon power-up or initialization, the current surface size and processor address must be loaded into control registers before packets can be correctly routed to their destination. The CPU must also write the normalization vector into the routing register. Data transfers are done on a 144 byte packet everywhere in the Post Office except at the PE interface where the CPU can signal completion of a packet read or write by the appropriate command.

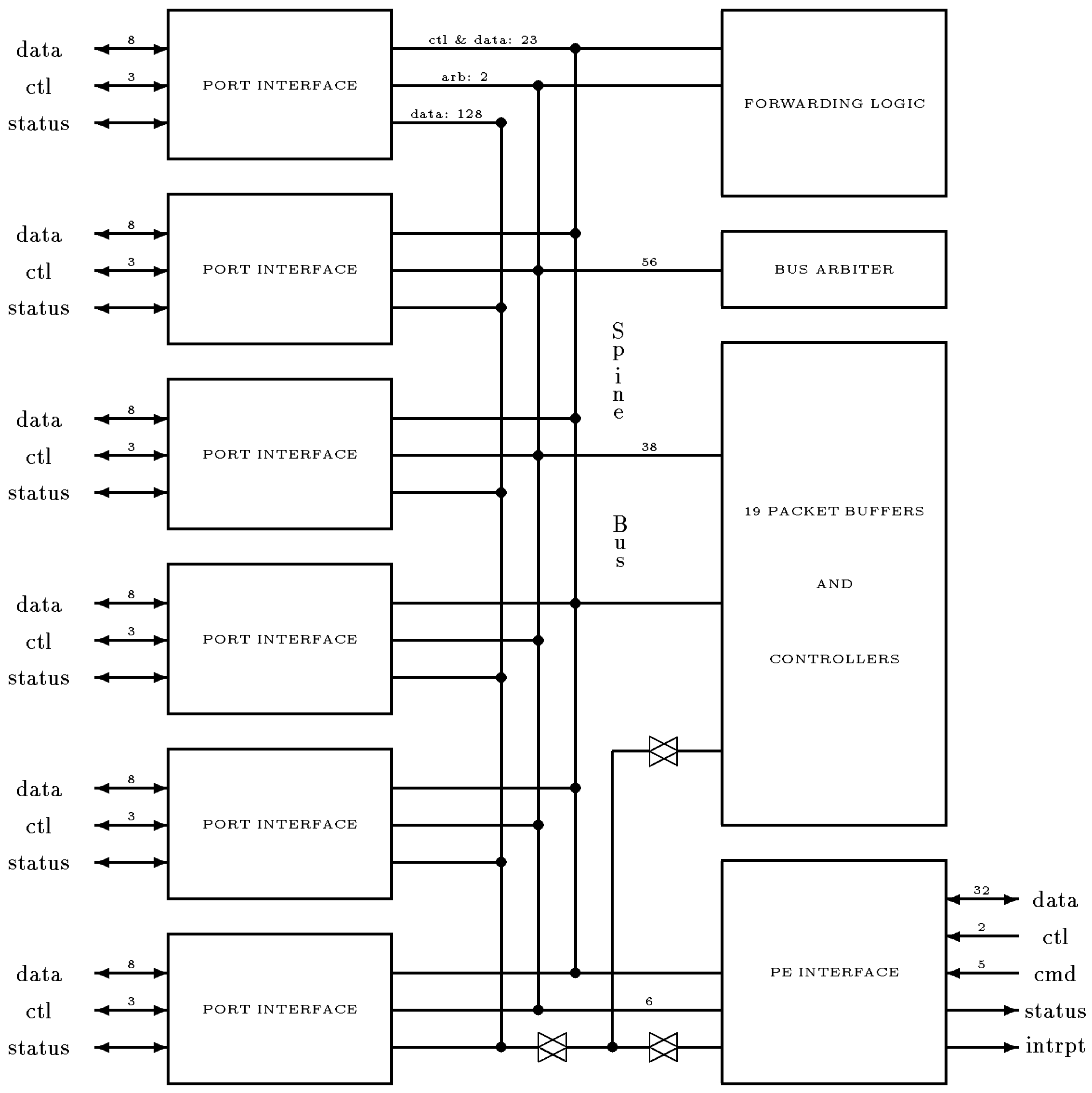

Figure 3: Post Office Block Diagram

All registers and memory in the PE interface are static. Although the Mayfly architecture is designed to service message traffic quickly[23], an inbound or outbound packet can remain unread or partially loaded in the buffers indefinitely since the CPU's software protocol will determine the timing of packet reads and writes.

Reliability requires that at least one Post Office node contain a valid copy of every packet being delivered. The PE interface loads each outgoing packet into the *packet buffer* block. The source Post Office is responsible for keeping a valid copy of the message until it has been accepted and placed into the packet buffers of another Post Office (which could be several hops away due to cut-through). The first buffer location can then be freed for storing another packet.

Packet buffer storage requests are verified by the forwarding logic block. If there are insufficient buffers to guarantee deadlock free operation, the request will be rejected. Otherwise, the routing results will be latched and a free packet buffer address will be returned to the requester. This packet buffer address will not be reused until the packet has been successfully forwarded. This block will also forward centrally stored packets to a free port for delivery using adaptive best path, no-farther, and random routing.

The packet controller block contains the 19 packet buffers and the associated controllers. The controllers snoop spine bus transactions. When the forwarding logic matches a free port to a buffered packet, the associated controller transmits the packet to the outgoing port.

The speed of the PE interface may vary greatly depending on CPU speed, the amount of data to be transmitted, software drivers, or CPU state. Centrally buffering outbound packets at the source and destination nodes increases the packet latency, but frees the postal network from PE and software dependencies. This allows the bandwidth of the external ports to be fully utilized as all external Post Office transfers are independent of the PE interface. Although the latency of lightly loaded networks is increased, performance loss under heavy loads and congestion will be much lower. This is an important property for a scalable communication controller.

The port interface controls transfers across an eight-bit bidirectional link connecting two Post Office chips. One interface becomes the sender, while the other becomes the receiver. Hand-shaking across three control wires use a two cycle protocol. The port interface will receive a service request to forward a packet to the adjacent chip. If the interface is busy, either sending or receiving a packet, the delivery request will be rejected. When a delivery request is made to an idle interface, the interface must then arbitrate for control of the external link. If this interface wins the arbitration, then the delivery is accepted and this interface becomes the sender. If the arbitration is rejected, the connected Post Office won the arbitration. This interface becomes a receiver and the delivery request is rejected.

The first four bytes loaded by a receiving port interface contain the destination address of the packet. After latching the address the routing logic calculates the delivery port(s). The packet is then immediately forwarded using virtual cut-through to the destination port or the packet buffers if this node is the final destination. When cut-through forwarding through the shortest path(s) cannot occur, the interface will attempt to store the packet in the packet buffers. If there is no room in the buffers due to deadlock prevention, delivery to this node is rejected.

4 Design Style

4.1 Datapath

Data is transferred between elements using a four-cycle self-timed bundled-data protocol [24]. This is a weaker model than that of speed-independence used for control signals inside state machines[25]. There are also some datapath circuits which are controlled in a *clocked* manner, 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 generated in burst mode concurrently 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.

An example can be found in the port interfaces. The number of bytes which have been transmitted is recorded by a clocked counter. A state machine is signalled by the counter when the last byte has been transmitted, indicating the completion of the data delivery phase. This counter is implemented from eight-transistor, two-phase, dynamic, shift register stages. It is clocked by a state machine that latches and enables the packet data onto the external bus. The timing assumption is that the shift register *done* signal will arrive before the completion of the latch/enable cycle, which is easily verifiable.

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 and process parameters. Hence timing assumptions were used only where they could be verified easily at the implementation level and where completion signal generation was too costly in either area or speed.

Each RAM block is the size of a packet, 1,152 bits. They are configured in 128 $\times$ 9 arrays, so it is fairly efficient to detect when a word line has been driven. Data validity and pre-charge is sensed on a single bit-line pair, with the assumption that all bit-lines will exhibit similar delay.

Many resources in the Post Office are accessible by other components. Access to these resources are made in a nondeterministic but mutually exclusive fashion. For example, two port interfaces may attempt to transfer data along the spine bus. Also, port interface A and B may both request forwarding a packet through port interface C. These two cases require different types of arbitration circuits. In the first case, the two accesses will be serialized on the spine bus; the loser of the arbitration will block until the resource is freed. In the second case, the accesses should *not* be serialized! The winner will transfer its packet through port C, but the loser should not be blocked until the winner's transfer completes. Rather, it should abort and proceed with other duties. Standard arbitration will serialize accesses for the first example, but a *naking* or *nonblocking arbiter* is required for the operation specified in the second example.

There are 13 naking arbiters in the Post Office. We found nonblocking arbiters to be a simple yet intriguing asynchronous circuit and posted it as a design and implementation exercise to our peers [26] who are designing asynchronous tools and circuits. Our solution consists of a "sequencer" (which is built out of ME elements and a few NAND gates) and a state machine specified by MEAT.

4.2 Burst Mode Controllers

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 implied stability requirement 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. 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 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. In the latter case, there is no way to safely distinguish in the asynchronous world whether C is still supposed to occur or whether the 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.

5 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 associated input and output bursts.

The first automated task performed by MEAT is to generate a flow table[10] 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[15]. 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 solution 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 is complex, it can easily be broken up into a tree of complex gates with improved overall performance[27]. Typical state machines

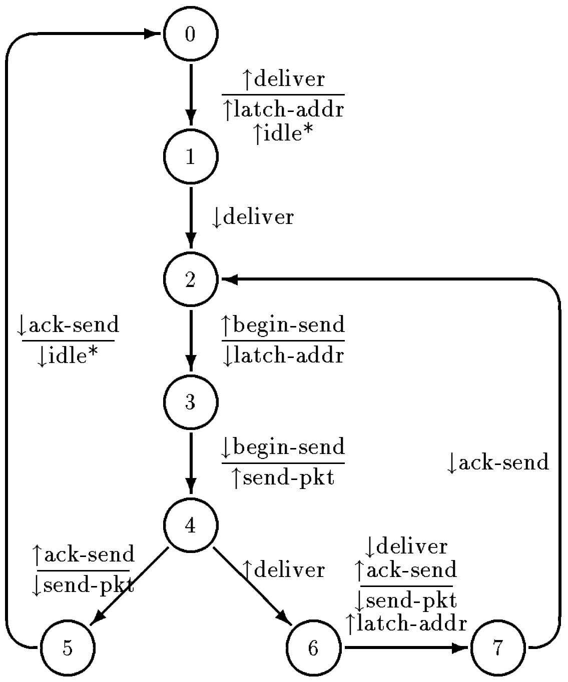
0
↑deliver / ↑latch-addr ↑idle*
1
↓deliver
2
↓ack-send / ↓idle*
↑begin-send / ↓latch-addr
3
↓begin-send / ↑send-pkt
↓ack-send
4
↑ack-send / ↓send-pkt
↑deliver
↓deliver ↑ack-send / ↓send-pkt ↑latch-addr
5
6
7

Figure 4: Sbuf-send-ctl State Machine

in the Post Office have an input to output delay of 2 to 5 inverter delays.

6 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 4. 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 [28,3].

The specification of sbuf-send-ctl from Figure 4 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 $A\tilde{}$ 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, Y0 and Y1.

```
> (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 $\overline{\textit{begin-send}}$ 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[10] 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 5. 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.

7 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[29], these systems, including full scale computers[5], were board level designs

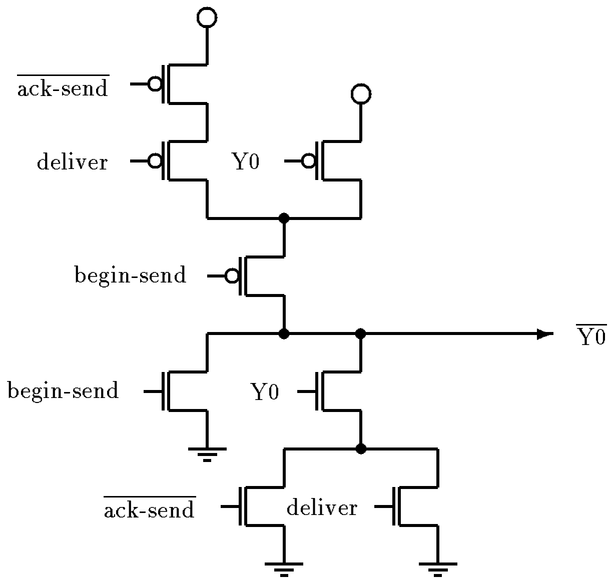

Figure 5: Complex CMOS Gate for sbuf-send-ctl Y0

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 development of the burst-mode MIC method, the use of localized timing assumptions, and the complex gate approach. All of these methods have proven to significantly enhance the performance of the resultant design. The next problem was that too much of our limited manpower budget (4 people) 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. We were then forced to fix that oversight.

We then found that our SIC based logic minimization methods were not valid for burst-mode operation. We are indebted to Steve Nowick for his assistance in this discovery and its eventual solution. 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 SI or 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 uses a library of SI or DI 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:

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

This circuit is verified as hazard free under burst-mode assumptions but would fail to verify as a speed-independent circuit. Implementing the AFSM using a set of speed-independent library modules, such as C-elements, Merge elements, and Toggles (indicated by triangular elements), produces the circuit in Figure 6.

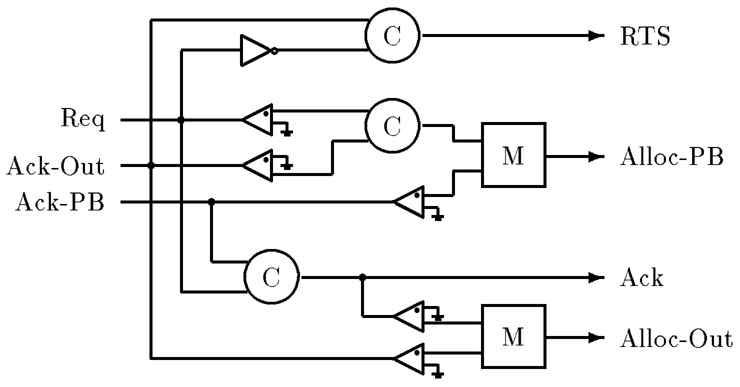

Figure 6: Speed-independent MP-Forward-Pkt Implementation

The MEAT version requires 6 gates and 5 inverters while the SI version requires 63 gates and 20 inverters. 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 state transition. 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 hand-shake 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 early sub-circuit 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, the wires 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 wire. 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 is 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. Busses are prone to be highly capacitive and as such are slow, inefficient, and noisy. A glitch on a bussed 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[27]. 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, busses should probably be avoided altogether. This implies a shift in architectural design styles. Driving large busses causes the vast majority of the delay in the Post Office circuitry. Sutherland's ACM Turing Award paper [30] presents a more appropriate style.

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 rather 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 isochronic forks are used [31,32]. Eliminating these slow signals will reduce instantaneous power consumption, resulting in reduced noise, and mitigate some of the problems associated with isochronic forks.

Many of our performance and reliability problems in the Post Office design were due to our lack of appreciation for pieces of the synchronous design process that remain important for asynchronous design. These include proper floor planning, tool quality, minimizing manual layout, etc. Our faith in the intrinsic modularity of asynchronous circuits was the prime contribution to the oversight. We were effectively confusing the global clock with the global planning process. The aggregate cost of this confusion was a factor of 2 performance reduction. 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 busses 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. It is interesting that the most serious design error in the Post Office caused a deadlock in the PE port's arbitration circuitry, and forced another fabrication run. The deadlock was caused by the usage of a C-element where an AND gate behavior was correct.

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

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

[3] 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.

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

[5] 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.

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

[7] 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.

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

[9] 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.

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

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

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

[13] 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.

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

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

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

[17] 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.

[18] A. L. Davis, B. Coates, and K. Stevens. Automatic Synthesis of Fast Compact Self-Timed Control Circuits. In *Proceedings of the IFIP Working Conference on Asynchronous Design Methodologies*, 1993.

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

[20] Kenneth S. Stevens. The Communications Framework for a Distributed Ensemble Architecture. AI 47, Schlumberger Palo Alto Research, February 1986.

[21] 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.

[22] R. M. Fujimoto. *VLSI Communication Components for Multicomputer Networks.* PhD thesis, Univ. of California at Berkeley, August 1983.

[23] A. L. Davis, B. Coates, R. Hodgson, R. Schediwy, and K. Stevens. Mayfly System Hardware. Technical Report HPL-SAL-89-23, Hewlett-Packard Laboratories, April 1989.

[24] 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.

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

[26] S. M. Nowick and D. L. Dill. Practicality of state-machine verification of speed-independent circuits. In *International Conference on Computer-Aided Design (ICCAD).* IEEE Computer Society Press, 1989.

[27] 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.

[28] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Synthesis of Verifiably Hazard-Free Asynchronous Control Circuits. In Carlo H. Sequin, editor, *Proceedings of the 1991 UC Santa Cruz Conference on Advanced Research in VLSI.* MIT Press, 1991.

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

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

[31] 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.

[32] Kees van Berkel. Beware the Isochronic Fork. *Integration, the VLSI journal*, 13(2):103–128, 1990.