A Comparison of Self-Timed Design using FPGA, CMOS, and GaAs Technologies

E. Brunvand* N. Michell K. Smith

Dept. of Computer Science, University of Utah
Salt Lake City, UT, 84112

Abstract

Asynchronous or self-timed systems that do not rely on a global clock to keep system components synchronized can offer significant advantages over traditional clocked circuits in a variety of applications. One advantage is that because of the separation of timing from functionality in these systems, the same circuit may be implemented in a variety of technologies without modification to the circuit. In this paper we explore one approach to self-timed design and describe implementations of an example circuit in three different technologies. The simple routing chip used as the example has been described by writing a program in OCCAM, *translated into a circuit consisting of a small set of basic modules, and implemented using Actel FPGA, CMOS, and GaAs technologies.*

1 Introduction

As VLSI technology improves, the systems that can be built become larger, faster, and more complex. Along with these improvements, however, come many problems directly associated with the speed and scale of the new systems. Asynchronous design is currently attracting renewed interest as a method for coping with some of the problems associated with improved VLSI technology.

In this paper we present an example of a simple self-timed routing chip implemented in three different integrated circuit technologies. This demonstrates our ability to design functioning self-timed modules in three very different technologies and gives evidence that self-timed design is useful across technologies with very different performance characteristics.

The example chip is described as an OCCAM program and translated automatically into a self-timed circuit [6]. This circuit is described in terms of a small set of modules designed to facilitate this style of self-timed design [3, 11]. If these modules are available in different technologies then the same circuit can be easily implemented in whichever form best suits the application. The self-timed nature of the module library assures that the same circuit will continue to function correctly regardless of the timing characteristics of the implementation technology. In fact, parts of the system may be upgraded into different technologies and substituted into the complete system incrementally. Because the parts are self-timed, the entire system will continue to function despite the different speeds of the component parts.

The self-timed modules used in this example system are currently instantiated as a set of macros for Actel field programmable gate arrays (FPGAs) [1, 4], as a set of CMOS cells using the MOSIS SCMOS technology [6], and as Gallium Arsenide (GaAs) cells used with the PPL design environment [8]. We describe implementations of the automatically generated router circuit in each different technology.

2 Circuit Modules

Although self-timed circuits can be designed in a variety of ways, the circuits considered here predominantly use two-phase transition signalling for control and a bundled protocol for data paths. Two-phase transition signalling uses transitions on signal wires to communicate. Only the transitions are meaningful; a transition from low to high is the same as a transition from high to low and the particular voltage on each wire is not important. The communication protocol uses two wires: A wire called *Req* to request service, and a wire called *Ack* to acknowledge completion of that service.

A bundled data path is a compromise to complete self-timing that uses transition control wires and a set, or *bundle* of conventional data wires. The *bundling constraint* requires that the data bundle and the control wires be constructed such that the value on the data bundle is stable at the receiver before a signal appears on the control wire. This

*This work is supported in part by DARPA award J.FBI.89.102, and in part by NSF award MIP-9115372

condition is similar to, but weaker than, the equipotential constraint [10]

The circuit modules used in all three technologies are based on those described in more detail elsewhere [6, 3, 11]. They include the following circuits:

Merge The "OR" function for transitions, implemented by an XOR gate.

Join The "AND" function for transitions, implemented by a C-element.

Call A module that acts as a hardware subroutine call allowing multiple access to a shared resource. The Call module routes the *Req* signal from a client to the subroutine, and after the subroutine acknowledges, routes the *Ack* back to the appropriate client. The requests must be mutually exclusive.

Select A module that steers an input transition to one of two outputs based on the value of a Boolean *select* signal. The *select* signal is a bundled signal with respect to the input transition.

Q-select A module like a select, except the *select* signal is not bundled and may be changing even when the Q-select is looking at it. Thus, it requires some way of sampling the *select* signal reliably.

Latch A module that latches bundled data signals upon receipt of transition control signals.

Carry Completion Adder A form of adder that reports when the addition is complete by sensing when the carry chain is complete.

3 Example: A Simple Routing Chip

The example circuit, whose block diagram is shown in Figure 1, is a switch for cut-through packet routing in a multiprocessor interconnection network similar to the Torus Routing Chip described by Dally and Seitz [7].

This routing circuit was designed by writing a program in a version of OCCAM [9, 6], a concurrent programming language useful for describing collections of small concurrent processes that cooperate through communication. This program description, shown in Figure 2, was translated automatically to a circuit description using a technique described elsewhere [6]. This translation process results in a netlist of circuit modules from the set of modules described earlier.

This version of the router uses four-bit wide data paths on all channels extended with a single tag bit to indicate the end of a packet. Address information is contained in

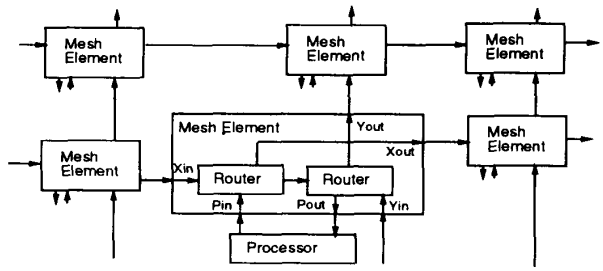

Figure 1: Routing Circuit Block Diagram

the first two words of a packet. Packets are routed first in X and then in Y. The first word is used as the current address and decremented as the mesh-element reads it. If the result is non-zero, the packet is forwarded in the same direction it came from; if the result is zero, the word is dropped and the packet changes direction. When the Y address is decremented to zero, the packet has arrived at its destination and is routed to the attached processor.

An abstract schematic for the control part of a single router macro is shown in Figure 3. Note that it takes two router macros to make a single mesh-element macro as shown in Figure 1, so the full mesh-element control requires two copies of the schematic.

The control path of the router uses a pair of Q-select macros connected in a ring to check, in round-robin fashion, whether data are available on either of the two input channels. When the channel indicates that there are data available, the router begins accepting a packet from that channel. The router will consume an entire packet from that channel and then check for further packets beginning with the other input channel.

The data path of the router includes a carry completion decrement unit to decrement the address of incoming packets. When the address reaches zero the packet switches directions in the router. Incoming words from the packet are buffered in a transition latch.

3.1 Actel FPGA Implementation

The self-timed modules described earlier have been implemented as a library of macro cells for use with Actel field programmable gate arrays. The library is integrated with the Workview suite of schematic capture tools offered by ViewLogic. These modules are all implemented in terms of the basic cell library provided by Actel [1], and have been implemented and tested for functionality.

Control modules all use two-phase transition signalling and are implemented in a small number of Actel basic logic modules ranging from a single logic module for a C-element

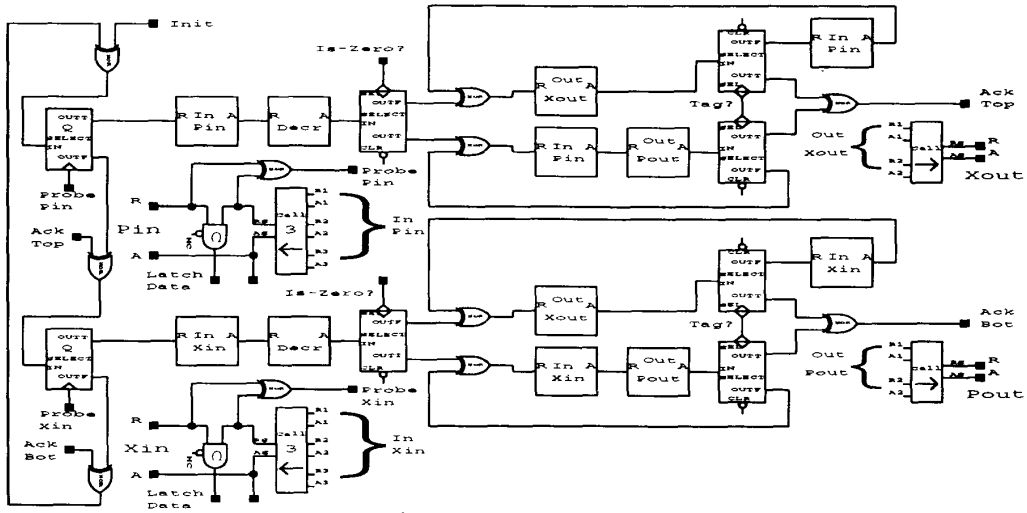

Figure 3: Control Path for a Router Macro

or XOR, to five logic modules for a Call module, and up to eight logic modules for a Q-select module.

For example, consider a C-element. The two-input version of this gate will force its output low when both inputs are low. The output will stay low until both of the inputs are high at which time the output also goes high. If the inputs are in opposite states, the output holds its last value. The library implementation of this circuit is shown in Figure 4 using the *MXT* multiplexer macro from the Actel macro library. This version of the C-element has a $\overline{\text{clr}}$ signal that forces the output and internal state of the C-element low. The Actel module on the left of the figure implements the function of the C-element, and the module on the right forces the output and internal state low on a $\overline{\text{clr}}$ signal. If the clearing signal is not required, the C-element may be implemented using a single Actel macro. Other control modules are similarly assembled using the basic cell library provided by Actel.

Data path modules are also implemented using the Actel basic cell library. A transition latch, for example, is implemented in a single Actel logic module. A carry-completion adder bit, on the other hand, requires five Actel logic modules.

The small size of the Actel basic macro makes it an ideal choice for implementing novel circuit structures. The self-timed modules are each implemented in a small number of Actel basic modules and have been tested for functionality. More detailed descriptions of the implementations of these modules can be found in other documents [4, 5].

The Actel version of the routing chip has been implemented using an Actel 1010A FPGA. This FPGA contains 295 Actel logic modules which, according to Actel, are equivalent to 1200 gate array gates, or 3000 PLD/LCA gates [1]. The automatically generated netlist was connected manually using the ViewDraw schematic capture system and the placement and routing information for the Actel part was generated using Actel's ALS software [1]. This example version uses 39 I/O pins and 294 Actel logic modules and corresponds to 99% utilization of the logic modules available on the Actel 1010A FPGA. The chip has been programmed and tested for functionality, and correctly implements the function of the mesh-element.

3.2 CMOS Cell Set Implementation

The CMOS version of the self-timed module library has been implemented using the MOSIS SCMOS design rules. This allows the designs to be fabricated using a variety of scaling factors through the MOSIS chip brokerage service.

The implementation of a C-element as a CMOS module is shown in Figure 5. The stack of transistors at the left of the figure implement a dynamic inverting C-element. When both inputs are low, the output of that stack is pulled high. When both inputs are high, and $\overline{\text{CLR}}$ is not asserted, the output is pulled low. The transistor network on the right inverts the output of the stack to become the output of the C-element, and provides the feedback to hold the C-element output at its current value when the inputs are in different states.

The other control and data modules are also implemented as small collections of CMOS transistors. These modules range in size from twelve transistors for an XOR module, to around a hundred transistors for the more complex modules. The CMOS implementations are described in more detail in other documents [6].

The mesh-element router example contains $\approx$7,500 tran-

```
;; Start with some definitions
(Defvar *width* 5)                          ;Define the data path width
(Define "eop" '(subseq data (1- *width*) (1- *width*))) ;Alias for tag bit
(Add-Gate "notzero" '(*width* 1))           ;Gate to sense non-zero address
(Add-Gate "decr" '(*width* *width*))        ;A decrement-by-one gate
;; Send data from in to out while data are not tagged
(Process send-data ((Chan in out))
  (While (not eop)                          ;while data are not tagged
    (! out data)                            ;send data out
    (? in data))                            ;and get the next byte
  (! out data))                             ;send last byte
;; Switch data from input to one of two outputs depending on 1st byte.
(Process switch-data ((Chan in Aout Bout))
  (If ((notzero data)                       ;if data are non-zero
    (send-data in Aout))                    ;send it out on Aout
  (True                                     ;otherwise
    (? in data)                             ;drop it, get new byte
    (send-data in Bout))))                  ;and send out on Bout
;; Take data from whichever input channel is ready and route it to
;; either of the two output channels.
(Process router ((Chan Ain Bin Aout Bout))
  (Block ((Var data(*width*)))              ;define a local variable
    (While True                             ;do forever
      (Fair-alt                             ;choose fairly
        ((True (? Ain data))                ;data on the A input
          (Set data (decr data))            ;decrement address
          (switch-data Ain Aout Bout))      ;send out data
        ((True (? Bin data))                ;data on the B input
          (Set data (decr data))            ;decrement address
          (switch-data Bin Aout Bout))))))  ;send it on out
;; The router element. Route data streams first in the X and then
;; in the Y direction based on information in header bytes
(Process mesh-element ((Chan Xin Yin Pin Xout Yout Pout))
  (Block ((Chan mid(*width*)))
    (Par
      (router Pin Xin Xout mid)             ;route in X direction
      (router mid Yin Yout Pout))))         ;route in Y direction
```

Figure 2: A Simple Routing Process

sistors using the current version of the CMOS cell library. These cells were designed conservatively with extra transistors used to buffer inputs and outputs of the individual modules. The resulting layout using a 2μ CMOS process with placement and routing by the MOSIS Fusion service [2] measured 4164 by 2104 microns and fit into a MOSIS standard 40-pin 4600 by 6800 micron pad frame. The chip was then fabricated through MOSIS and tested.

3.3 GaAs PPL Implementation

The module library has also been implemented in GaAs as a set of cells for use with the PPL tools developed at the University of Utah [8]. The modules are designed using direct-coupled FET logic (DCFL) circuits and fabricated by Vitesse through the MOSIS chip brokerage service.

The DCFL style of GaAs circuits encourages the use of NOR gates as circuit building blocks. For example,

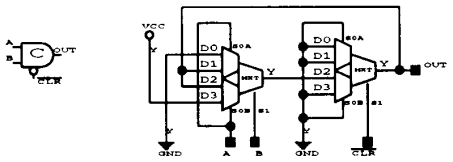

Figure 4: Actel Implementation of a C-Element with Clear

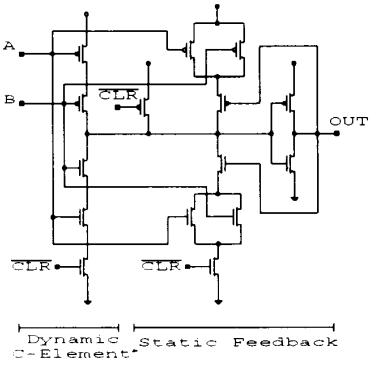

Figure 5: A Static CMOS C-element

the DCFL version of a C-element is built from a two-level network of NOR gates which implement the function $A \cdot B + A \cdot Out + B \cdot Out$. Other self-timed control and data modules are similarly implemented using predominantly NOR-style circuits.

To gain experience with GaAs technology we have built several PPL GaAs test chips and the results have been very positive. One test chip contains ring oscillators, a self-timed 8-bit counter and pieces of the control section of the self-timed router. All parts of this test chip have been found to function correctly. Ring oscillator delays as low as 60ps unloaded, and 128ps loaded have been measured. Test results show that the self-timed counter built using Toggle modules and XOR gates counts at an average rate of 300MHz. The fastest chip counted at 400MHz. The pieces the router include the Q-select loop in the control path that implements the OCCAM ALT statement, and, as a separate piece, the remainder of the control path for a single two by two router. A single transition takes 7ns to complete one circuit of the Q-select loop used to choose which input channel to accept an incoming packet from. A single transition also takes 7ns to traverse the loop constructed from the remaining pieces of the control path.

The GaAs version contains only a single router element rather than the full two-router mesh element. The chip is implemented using Vitesse's 1.2μ GaAs DCFL technology

Table 1: Comparison of Three Versions of the Example Router

Technology	Input Time	One Router Latency	Two Router Latency
Actel FPGA 1.2μ CMOS	132ns	180ns	333ns
SCMOS Cells 2μ CMOS	59ns	73ns	132ns
GaAs PPL 1.2μ DCFL GaAs	4ns	9ns	N.A.

and measures 2670μ x 2230μ with 1860μ x 1308μ available for active circuitry in the form of PPL tiles. For this circuit, these tiles contain ≈4,000 GaAs transistors.

3.4 Performance Measurements

Performance data for the three different implementations is shown in Table 1. The performance numbers presented for the implementations will be in terms of the elapsed time between the issuing of a request signal to the chip and the assertion of an acknowledge signal by the chip. The first number is how quickly an implementation can accept new words at an input channel. The second and third numbers measure the latency of a single word through the mesh-element. Recall that the mesh-element is made up of two copies of a two by two router circuit. Depending on the address in the header, data must pass through one or both of those routers. The performance of the parts is the measured time between an input request, and the output request at the appropriate output channel. The times reported for the three implementations are for the data words and not the address word. The first word of a packet, being an address, incurs an extra delay through the decrement module that is not required for subsequent data words of a packet.

4 Conclusions

Many of the advantages of self-timed circuits over more traditional synchronous circuits stem from the separation of timing from functionality found in self-timed designs. This allows a single self-timed circuit to be implemented in a variety of technologies each displaying very different performance characteristics. To demonstrate this flexibility, we have implemented an example self-timed circuit in three different technologies and compared the results.

The example routing circuit was compiled automatically from an OCCAM program description. The result of this compilation is a netlist of circuit modules from a small set of self-timed primitives. These primitives are currently implemented in three technologies: macros for use on Actel FPGAs, a set of CMOS cells, and a set of DCFL GaAs cells. These technologies represent a wide range of price and performance characteristics. They also suggest an interesting development path for experimenting with self-timed design styles. Novel designs may be prototyped quickly using the FPGA version of the module library. As the system becomes more stable, parts of the system may be recast in a faster or denser technology to incrementally improve system performance. The self-timed nature of the circuits allows this incremental replacement and mixing of parts that operate at different speeds with no change to the circuit or retiming of the system. All that is required is to implement that circuit part using a different technology.

References

[1] Actel Corporation. *ACT Family Field Programmable Gate Array Databook*, March 1991.

[2] Ronald F. Ayres. FUSION: A new MOSIS service. Technical Report ISI/TM-87-194, Information Sciences Institute, 1987.

[3] Erik Brunvand. Parts-R-Us: A chip aparts ... Technical Report CMU-CS-87-119, Carnegie Mellon University, 1987.

[4] Erik Brunvand. A cell set for self-timed design using actel FPGAs. Technical Report UUCS–91–013, University of Utah, 1991.

[5] Erik Brunvand. Implementing self-timed systems with FPGAs. In W. R. Moore and W. Luk, editors, *FPGAs*, chapter 6.2, pages 312–323. Abingdon EE&CS Books, 1991.

[6] Erik Brunvand. *Translating Concurrent Communicating Programs into Asynchronous Circuits.* PhD thesis, Carnegie Mellon University, 1991. Available as Technical Report CMU-CS-91-198.

[7] William J. Dally and Charles L. Seitz. The torus routing chip. *Distributed Computing*, 1:187–196, 1986.

[8] Jun Gu and Kent F. Smith. A structured approach for VLSI circuit design. *Computer*, November 1989.

[9] Inmos. *Occam Programming Manual*, 1983.

[10] C. L. Seitz. System timing. In *Mead and Conway, Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[11] Ivan Sutherland. Micropipelines. *CACM*, 32(6), 1989.