Designing for High Speed-Performance in CPLDs and FPGAs

Zeljko Zilic, Guy Lemieux, Kelvin Loveless, Stephen Brown, and Zvonko Vranesic
Department of Electrical and Computer Engineering,
University of Toronto, Canada
email: zeljko | lemieux | kelvin | brown | zvonko @eecg.toronto.edu

Abstract

We review some of the techniques for designing with FPGAs and CPLDs used in our design of a large-scale multiprocessor. Several issues concerning high-speed design are addressed: selecting the appropriate programmable logic family, obtaining the fastest circuits with given families, and supplementing the synthesis software. We describe techniques that we used for designing controllers and counters with both CPLDs and FPGAs.

1 Introduction

With the development of new types of sophisticated programmable logic devices, such as Complex PLDs (CPLDs) and FPGAs, the process of designing digital hardware has changed dramatically over the past few years. The number of applications for large PLDs has grown so rapidly that many companies have produced competing products and there is now a wide assortment of devices to choose from. A designer who is not familiar with the various products faces a daunting task in order to discover all of the different types of chips, try to understand what they can best be used for, choose a particular company's device, and then design the hardware.

The purpose of this paper is to discuss the practical issues that face designers who wish to implement circuits in today's sophisticated CPLDs and FPGAs. Our focus is on the most demanding class of application circuits: those that require state-of-the-art speed-performance. The specific PLDs used are Altera MAX 7000 CPLDs and Altera FLEX 8000 FPGAs, and the circuits are mapped using Altera's MAX+Plus II (ver. 5.2) CAD system. We have chosen Altera PLDs because of their high performance in both the CPLD and FPGA categories. Using examples from a modern multiprocessor system design, we show that only through careful (and clever) design effort can the maximum speed-performance available in today's PLDs be realized. More specifically, we will address the following questions:

- For specific applications, which devices provide higher speed-performance: CPLDs or FPGAs?
- How can circuits be designed to achieve the highest possible speed-performance in a given device?
- How can CAD tools be assisted, or designs modified to achieve higher speed-performance results?

2 Motivation

Issues facing designers who wish to use PLDs are fairly straightforward when applications are relatively small. For this reason, our focus is on applications that require larger PLDs, namely those that fit within the Complex PLD (CPLD) and Field-Programmable Gate Array (FPGA) categories. We will illustrate that speed-performance achievable for a given application circuit is greatly affected by which of these two categories of chips is selected. For example, circuits that require fairly wide gates (such as state machines or decoders) almost always operate faster in CPLDs.

Even within a single category of device, products from different manufacturers (or even the same manufacturer) can result in significant differences in performance. Section 3 will illustrate this by showing the effects of matching the structure of a design to the architecture of the chip being used. It is important to note that such subtleties can be appreciated only through experience with the devices. PLD marketing literature often gives the impression that a certain level of performance is available for a wide range of application circuits; the reality is that maximum performance can be obtained only for subcomponents of applications that are well-matched to the PLD architecture. A corollary is that while today's CAD tools are sophisticated enough to map fairly abstract descriptions of a circuit into a PLD, maximum performance will only be obtained for circuits that are described in a way that provides an obvious mapping from the circuit description into the device.

As an example of how PLD architecture affects speed-performance, consider a generic finite state machine (a real example of such a circuit is given in the next section). If a FSM is to be implemented in an FPGA, then the amount of logic feeding each state machine flip-flop must be minimized. This follows because in FPGAs flip-flops are directly fed by logic blocks that have relatively few inputs (typically 4 - 8). If the state machine flip-flops are fed by more logic than will fit into a single logic block, then multiple levels of logic blocks will be needed, and speed-performance will decrease. For this reason, designers usually use "one-hot" state machine encoding when targeting FPGAs, so that the amount of logic that sets each flip-flop is minimized. Even in a CPLD architecture, speed-performance of a state machine can be significantly affected by state bit encoding;

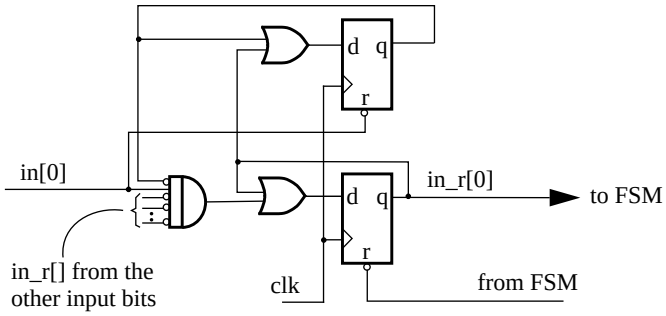

Figure 1 - *Small Sequential Circuit.*

for example, in the Altera MAX 7000 CPLDs, flip-flops that are fed by five or fewer product terms OR-ed together will operate faster than those that require more than five terms. In general, designers who wish to obtain maximum performance for applications need to constantly consider the nuances of their PLDs architecture.

3 High-Speed Design

In this section, we describe several examples of application circuits that we have implemented in both CPLDs and FPGAs. The purpose of this discussion is to show the relative speed-performance of each example in both types of devices and also to illustrate the effects that the way in which the circuit is described to the CAD system can affect performance. All of the design examples are real circuits from the NUMAchine multiprocessor system [6].

3.1 Simple Sequential Circuit

The example discussed in this section is a fairly small sequential circuit consisting of a FSM with registered inputs. Figures 1 and 2 illustrate the structure of the circuit (note that the exact functionality of the circuit is not important for our purposes). Fig. 1 shows the manner in which each of the inputs to the circuit is registered; the arrangement of FFs shown captures all of the inputs into the *in_r[]* register (all inputs are treated identically at *in[]*) and then does not latch again until all bits of the *in_r[]* register are cleared (by the FSM). The structure of the FSM is indicated by the bubble diagram in Fig. 2, from which the observant reader will realize than the machine implements a classic priority-based arbitration scheme. To illustrate performance of this circuit in both CPLDs and FPGAs, we implemented two versions of the circuit: one with only five input bits, and one with 13 input bits (the real circuit used in our multiprocessor has 13 bits).

Table 1 clearly shows that the CPLD implementation of the sequential circuit is much faster than the FPGA version. However, the most interesting aspect of this result is the *difference* between the five-bit and 13-bit versions of the circuit. Both versions operated at about 100 MHz for

	CPLD		FPGA	
	5-bit	**13-bit**	**5-bit**	**13-bit**
Speed	100 Mhz	100 Mhz	57 Mhz	40 Mhz

Table 1 - *Speed-Performance of Sequential Circuit.*

the CPLD implementation, while the 13-bit version was much slower than its smaller counterpart for the FPGA. This is a good example of how FPGAs are less suitable for implementing circuits that require “wide” logic gates (the 14-input AND-gates and the FSM logic for this example), whereas CPLDs can easily implement such designs.

3.2 Data Path Implementation

One of the most challenging aspects of design of high-performance computer systems is the connection of the microprocessor(s) to the rest of the system components. It is convenient to implement the requirements of wide data path circuits for interfacing to RISC processors and other system components (such as memory, I/O devices, and communication controllers) in large PLDs. In fact, in real systems much of the available board space is often occupied for this purpose. Invariably, the data flow is bidirectional and involves multiplexing and simple processing, such as address/data multiplexing, change in data width, and simple bit manipulation.

It is widely accepted by designers who use PLDs that FPGAs are the best choice for data paths, because wide logic gates are not required and the number of flip-flops needed is large. Also, there is a large number of I/O pins is required. For example, one simple bidirectional 64-bit latch with tri-state control requires more than 130 pins. These high I/O requirements, combined with simple logic needs is a good match for the resources in a typical FPGA, such as the 4-input look-up-table-based FLEX 8000 series from Altera. In some cases it may be possible to implement an entire wide data path in a single large FPGA, but it is more cost effective to partition the circuit into *bit slices* and assign each bit slice to a smaller device.

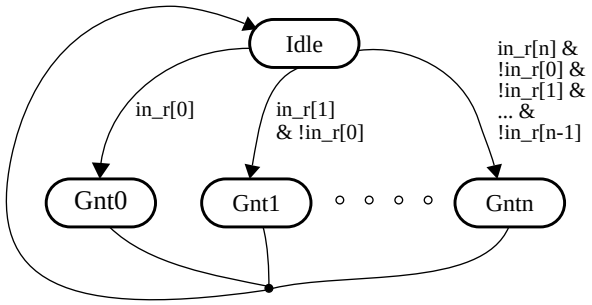

Figure 2 - *Small Finite State Machine.*

On the other hand, when speed of the concrete design is taken into account, CPLDs may be a better choice. For example, an implementation of the 64-bit wide datapath that we use in NUMAchine processor interface can run as fast as 83 MHz using CPLDs, while the FPGAs can only achieve 44 MHz. CPLDs provided better performance because of their simple structure that provides for very high-speed paths from input pins, through AND-OR logic and flip-flops, to output pins.

This is an interesting example which shows that the "accepted" guidelines for which type of PLD to use cannot be followed blindly; both CPLDs and FPGAs should be investigated.

3.3 Control for Data Paths

Circuits that control data paths have a very specific structure. The control can be realized by communicating state machines. Each of these machines possesses a characteristic structure of a tree-like FSM diagram, where the main branching is done in one of the initial states, as shown in Fig. 3. Other than that, the decisions that would change the control flow are rare. Experience shows that for large state machines of this type, CPLDs provide the only architecture that can ensure high speeds.

The most significant factor affecting speed-performance of state machines lies in the *state assignment*, the selection of codes for each state of the machine so that the resulting combinatorial logic. State assignment problem is very difficult because it is hard to estimate the complexity of the logic required for any assignment (an NP complete problem), and the number of possible assignments is large, on order of ($2^n!$). State assignment has been extensively studied, and approximate solutions can be outlined only when two-level AND-OR architectures are targeted. However, for multiple-level architectures, such as those found in FPGAs and some modern CPLDs, a simple AND-OR structure is not a precise enough model. Several heuristics have been developed for the state assignment for CPLDs and FPGAs. For both architectures, one-hot encoding seems to be the best choice [1, 2], especially if joined with the standard FSM decomposition and state splitting techniques. Indeed, our experience shows that most medium-size (10-20) state machines perform best using one-hot encoding in CPLDs.

Controller Implementations Using CPLDs

For controllers implemented using CPLDs, because of the limited number of FFs available, it is often desirable to minimize the number of state bits used. However, it is often impossible to use minimal length encoding. Indeed, for our larger state machines, minimal length assignments either could not fit into the designated CPLD or resulted in poor speed-performance. The reason is that the number of product terms that result by such assignments is too large.

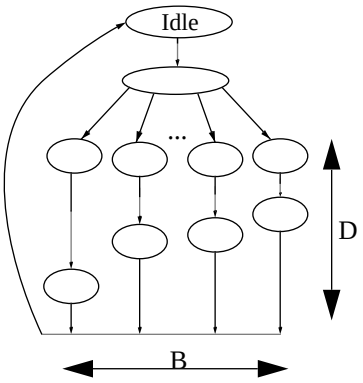

Figure 3 - *A Tree-like FSM.*

In order to improve the speed, as well as to be able to fit the designs, we were forced to investigate alternatives. One observation that helped shape the search for the solution is that by minimizing the number of 1's in the encoding the resulting logic becomes simpler. A key observation that shaped this class of solutions is that there is a typical structure in state machines, as already shown in Fig. 3.

The state assignment scheme that we propose is a combination of one-hot encoding to decode a "branch" and a binary encoding of position in the branch. If there are *B* branches with at most *D* states, then the number of bits needed is *B log D*, as opposed to the worst case *B*D*. Additional bits that describe a position within a branch are assigned in a way that simplifies the logic to generate the outputs (this method is called *face embedding* for input encoding [3]).

Such assignments were used for some of our most complex state machines. One such controller consisted of 52 states, had 27 inputs and controlled 41 outputs. The controller was decomposed into several smaller (1-4 states) and two large FSMs, with 17 and 20 states. To encode the states of these two machines, we needed 8 and 11 state bits, respectively. Both large state machines had branching structure as in Fig. 4. With our state assignment technique, the controller achieved a speed-performance of 55 MHz, using the relatively slow (with 12 ns pin-to-pin delay) Altera CPLD device 7192-12 and the "normal" logic synthesis style. With the same parameters, the machines produced using both the Altera-provided binary encoding and one-hot encoding ran at 40 MHz.

Controller Implementation using FPGAs

It is much harder to design fast state machines using lookup-table based FPGAs, as compared to CPLDs. First, the state assignment problem is much harder than for the two-level logic based devices. Second, and more important,

FPGAs are slower and more unpredictable, and so much more care is needed to achieve fast designs using FPGAs. Hence, in general they can be used in high speed applications only for relatively simple controllers.

We propose a scheme for designing controllers in FPGAs that further exploits the structure of the controllers; it uses on a large scale the decomposability of FSMs. It produces an assignment that uses less state bits than the one-hot encoding, and yet is faster for a given structure of controllers. The assignment roughly corresponds to orthogonal "coordinates" in the state of the system, and at the same time exploits if some of these are not time critical.

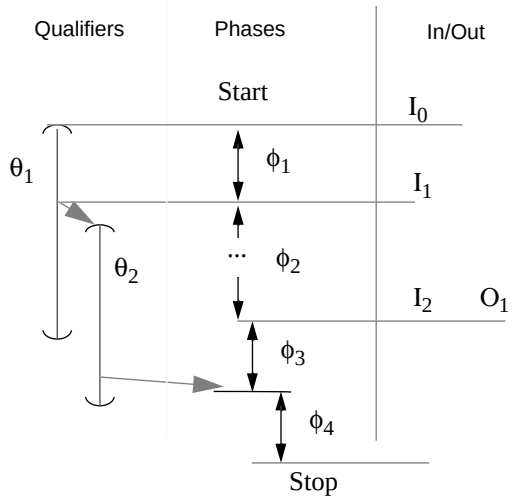

Figure 4 - *Example of a controller run*

As in the previous section, we rely on the structure of state machines that idle until there is a request for execution, and then perform a *run*. We use two types of information for describing the run. We use a notion of a *phase*, which is responsible for sequencing, i.e. executing the correct sequence. Phases are always strictly defined in time (its beginning and end). Usually, a phase can be imagined as a state that has one incoming and one outgoing point. Then, a single flip-flop with a small fan in can be used to realize the phase. In addition, we use *qualifiers*, which describe permanent characteristics of a run. For example, *phases* can correspond to specific actions like opening up paths to registers, arbitration phases and the like, while the *qualifiers* can denote the direction of operation (e.g. read/write), the type of the data item being processed and other features characteristic for a run. Fig. 4 shows an example of one such run initiated by an input I_0, which starts the phase ϕ_1, and memorizes the qualifier θ_1. The run is dependent on the inputs I_1 and I_2 at the end of the next two phases and qualifier θ_2 which starts the last phase.

The main reason that this scheme achieves the fast design lies in the fact that only phase bits change for most of the time, similarly to the case when one-hot encoding is used. Furthermore, we eliminate a special idle state at which the machine starts. This is important because the logic needed to generate this state is often the most complex in the entire state machine. Furthermore, qualifiers of a run are not time critical; their activation and deactivation can be moved in time for several clock cycles. This gives us an additional opportunity to eliminate these false critical paths.

The state space of the controller given is on the order of the product of the number of qualifiers and the phases, while the total number of bits needed for encoding the state machine is on the order of the sum of these. Such an encoding uses as many bits as possible to encode the orthogonal characteristics of the state space. While in FPGAs we are not primarily concerned with the number of bits used for the controller encoding, this approach has significant advantage even for the speed-performance because Altera FPGAs are hierarchical, and so produce significantly faster circuits when there is a locality in the state bit placement.

Using this approach, we designed a small bus interface controller that can transfer three sizes of data in two directions (16, 32 and 64-bit wide), and which performs several additional functions, like remapping of the address space and recognition of the type of the bus to which it is attached. It also includes conversion from 64 bit wide bus to the 32 bits on one bus, and 16 bits on another. Additionally, one bus has time multiplexed address and data lines, while another has separate address and data lines. The controller was able to run at 62.9 MHz on the Altera FLEX 8452A-3 device. We estimate that one-hot encoding would result in speeds well below 50 MHz. To encode over 25 states, we used 12 bits.

3.4 High-Speed Counter Design

In our multiprocessor system, a high-speed counter is required as part of a hardware monitoring system. The counter must be loadable, clearable, readable, and generate an overflow signal for cascading or interrupting a CPU. Clearly, the counter must be large enough to minimize overflows (32 bits is sufficient). Additionally, latency is not important, but being able to count an event every clock cycle is.

Basic Counter Design

A basic 32-bit counter was implemented in both FLEX 8000 and MAX 7000 parts by cascading eight of Altera's 74161 macrofunctions. Altera has optimized these macros separately for each device family; the FPGA implementation uses the dedicated carry chain while the CPLD just uses a single product term for each bit. The first two rows of Table 2 show the highest possible speed of this counter

using the fastest and smallest possible devices capable of a fit, namely the FLEX 8820A-2 and the MAX 7096-7.

	Device Type	# Cells (% of Device)	Maximum Frequency
Basic Design	CPLD	32 (33%)	125 MHz
	FPGA	34 (5%)	65 MHz
Design A	FPGA	39 (5%)	103 MHz
Design B		48 (7%)	164 MHz

Table 2 - *Speed-performance of 32-bit counters.*

The speed of CPLD-based counters is independent of the counter size (up to about 32 bits) and equals the maximum possible speed set by t_{pd} plus flip-flop setup time. This is because each counter bit needs to OR four product terms of up to 34 inputs, and the CPLD implements this easily. For the FPGA devices, wide product terms are not available. Instead, a dedicated carry chain is employed to implement the required wide AND. The speed of this carry chain limits the speed of the FLEX-based counter because a counter of double the size has twice the carry chain length and approximately half the speed performance.

Using the carry chain in FPGAs, however, hinders routability as well as limiting the speed of large counters. Because the carry chain is a fixed resource, all bits of the counter must be physically placed in an ordered, packed format. The packed format makes routing less likely to succeed because local density is higher and there are fewer placement alternatives. Furthermore, the ordered format restricts routing because CAD tools rely upon shuffling the order of logic blocks to improve limited interconnect connectivity. As described in the next section, both routability and speed can be reclaimed through clever design.

Improved Counter Design

As mentioned earlier, CPLD-based counters easily attain their highest speed capabilities, but FPGA-based counters suffer from slower speed and also poor routability. A linear feedback shift register is a very good alternative because it is easily routed and runs at very high speeds. However, this is not practical for our purposes because of the overhead involved in converting the shift register contents back into a count [4]. Fortunately, [5] describes a binary counter design which can be scaled to virtually unlimited size yet increment in constant time. We employed this technique in the Altera FLEX devices with promising results.

The key observation to make with the constant-time counter is that the high-order bits are incremented very infrequently, but they must all be able to change within a single clock cycle. The counter can be broken into blocks of increasing size, where each block's carry-in is actually a load enable for its flip-flops. The load enable removes the carry chain from the critical path by allowing carry propagation during the long idle time between increments of the block. However, a fast load-enable is crucial when an increment must be performed. Careful attention must be paid to this load-enable signal to achieve good counter performance.

We initially partitioned the counter into blocks of size 1, 2, 1, 8, and 20 bits (because multiples of 4 are easy to handle). At first, this design yielded very poor results; the counter ran at speeds close to 30 MHz*. By tuning the logic according to the architecture, such as forcing certain inputs and functions together in a lookup-table to reduce critical signal logic depth, speed was increased to about 80 MHz* (!). Further performance was extracted by hand-placing the critical logic elements of the counter: roughly 25% of the logic cells were manually placed and speed was measured at 103 MHz. This result is entered as design A in Table 2.

In design A, the critical path lies in logic that generates the load-enable signal because there are too many blocks to fit in a LAB, each requiring fast load-enable signals, and the blocks were small so overhead would grow too quickly if the logic was duplicated. Once these problems were identified, a new partitioning of 4, 4, and 24 bits was determined to be much better. Since it is no longer time critical, the carry chain in the 24 bit block is broken many times to aid placement. Also, by replicating the lower 4 bits, a fast load-enable signal can be generated within a LAB for the second and third blocks. The resulting speed was a surprising 164 MHz and is entered as design B in the table.

It is worthwhile to note that design A and B both obtain very high counter speeds with little logic overhead; only 48 logic cells were needed to implement a 32-bit counter. This is significant when compared to counter designs that are suggested by Altera. Altera's 16-bit prescaled counter runs at 143 MHz in a slower device (the A-2 speed grade was not available at that time), yet it used 101 logic cells. The logic savings is important to our application because it demands multiple counters to fit in inexpensive devices.

Although speed-performance was greatly improved, the effect on routability has not been shown. To demonstrate the routability improvement, a more complex circuit which requires four 32-bit counters is used. To reduce pin requirements and increase interaction, the counters share a read and write port via a 32-bit bidirectional data bus. A 4-to-1 multiplexer, which is pipelined for speed reasons, and

---

* Speed was extrapolated to 8820A-2; actual speed was 33% slower in an 8452A-3.

tristate buffers connect the counters to the data bus. This circuit was routed in Altera FPGAs of various sizes and pin capacities, and a typical result is presented in Table 3. In this table, the number of unrouted signals after each pass of the router was recorded. The number of passes and the number of unrouted signals illustrate the difficulty the router had with the circuit. In all our experiments, the improved counter required fewer passes of the router (usually only one) and always routed. In fact, the example in the table shows that only the most improved version was able to route.

Counter Partitioning	Number of Unrouted Signals after each Pass
full carry chain	40, 33, 33, 33, 33, 33
1, 2, 1, 28	26, 23, 22, 22, 18, 15
1, 2, 1, 8, 20	22, 12, 4, 2, 2
4, 4, (8+8+8)	7, 0 (successful)

Table 3 - *Routability of Different Counter Organizations.*

Further Insight

Although many benefits were realized with counter design B, obtaining a high-speed design required a great amount of effort. While optimizing, many difficulties were encountered and solved. By describing some of these pitfalls, we hope to help other high-speed FPGA designers.

First, the load-enable signals for higher-order subblocks are heavily loaded and can easily limit the speed performance; Altera devices quickly degrade when fan-out is large. We limit fan-out to 16 logic cells by replicating the load-enable signal. Altera's default way of doing this is to use an extra logic cell as a buffer, but the normal lookup-table inputs are too slow. The fastest way to produce a duplicate signal in Altera devices is to use the cascade input or to replicate the logic that produces it. We use the former approach with good results.

Also, replicating high fan-out signals to improve speed works, but there should be a way to mark them as "equivalent" so the CAD tool could automatically distribute them. Otherwise, assignments are made manually and we have seen this hinder routability.

Care must be taken when carry chains are used to produce wide-AND functions because they are glitch-sensitive. This was observed when generating the load-enable for the second subblock; glitches were delayed in the interconnect and arrived close enough to a clock edge to cause false increments. By latching the load-enable signal close to the carry chain output, the glitch is removed. While this causes the upper bits of the counter to be delayed by one clock cycle, it increases performance by removing the first subblock's carry chain from the load-enable critical path.

Dividing the counter into subblocks made initializing the counter very tricky. After loading a new value into the counter, a few idle cycles are required to settle the carry chain. Also, control signals must also be properly delayed to compensate for the pipelined load-enable.

Features such as cliques, which are a hint that certain logic should be kept together, help improve speed but are not strict enough. Better results were obtained by forcing certain logic into a cell and by forcing placement locality, although the latter step can hinder routability.

Summary

Altera CPLDs can easily implement very fast counters. In comparison, FPGAs are faster and better suited for this purpose, but performance and routability are compromised when the carry chain hardware is used. Both performance and routability can be improved by enhancing the counter design with a very a small cost in area. However, these gains cannot be realized without intimate knowledge of the FPGA architecture and a great deal of manual work on the part of the designer.

4 Final Remarks

We have presented several examples of high-speed digital circuits and have shown implementations in both CPLDs and FPGAs. For most applications, speed-performance in CPLDs is higher than in FPGAs. The examples also show that in order to obtain maximum speed-performance, it is often necessary to have intimate knowledge of the structure of the PLD, and to investigate many design alternatives.

References

[1] S. K. Knapp, *Accelerate FPGA Macros with One-Hot Approach*, Electronic Design, Vol. 38 No. 17, pp 71-78, Sept. 1990.

[2] Z. Zilic and Z. Vranesic, "On Retargeting With FPGA Technology," The First Canadian Workshop on FPGAs, pp. 14-1 - 14-5, June 1993.

[3] P. Ashar, S. Devadas, and A.R. Newton, *Sequential Logic Synthesis*, Kluwer Academic Publishers, 1992.

[4] Douglas W. Clark and Lih-Jyh Weng, "Maximal and Near-Maximal Shift Register Sequences: Efficient Event Counters and Easy Discrete Logarithms," IEEE Transactions on Computers, Vol. 43 No. 5, May 1994.

[5] J.E. Vuillemin, "Constant Time Arbitrary Length Synchronous Binary Counters," 1991 IEEE 10th Symposium on Computer Arithmetic, Grenoble, France, June 1991.

[6] Z. G. Vranesic et al. "The NUMAchine Multiprocessor", Computer Systems Research Inst. Tech. Report, CSRI-324, Toronto, April 1995.

[7] Altera Corporation, Application Brief 124, "FLEX 8000 Handbook", 1994.