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

1 Introduction

With the development of new types of sophisticated programmable logic devices, such as Complex PLDs 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 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 to achieve higher speed-performance results?

The paper is organized as follows. Section 2 provides the motivation illustrating why careful design is needed for PLDs, Section 3 discusses several practical examples of high-speed application circuits, and Section 4 contains final remarks.

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. More specifically, 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 liter-
nature 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 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 of applications, consider a generic finite state machine (a real example of such a circuit is given in the next section). If a finite state machine 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; for example, in the Altera MAX 7000 CPLDs, flip-flops that are fed by five or fewer product terms 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 PLD’s architecture.

3 Examples of High-Speed-Performance Applications

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 a large multiprocessor computer system.

3.1 Simple Sequential Circuit

The example discussed in this section is a fairly small sequential circuit consisting of a finite state machine 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). Figure 1 shows the manner in which each of the inputs to the circuit is registered; the arrangement of registers shown latches all of the inputs into the in_r register (all inputs are treated identically at the input latch) and then does not latch again until all bits of the in_r register are cleared (by the finite state machine). The structure of the finite state machine is indicated by the bubble diagram in Figure 2; the observant reader will realize than the machine implements a classic priority-based arbitration scheme. To illustrate performance of this type of 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
The results of implementing the two versions are shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>5-bit version in CPLD</th>
<th>13-bit version in CPLD</th>
<th>5-bit version in FPGA</th>
<th>13-bit version in FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed Performance</strong></td>
<td>100 Mhz</td>
<td>100 Mhz</td>
<td>45 Mhz</td>
<td>20 Mhz</td>
</tr>
</tbody>
</table>

**Table 1 - Speed-Performance of Sequential Circuit in CPLD and FPGA.**

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 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 not suitable for implementing circuits that require “wide” logic gates (the 14-input AND-gates for this example), whereas CPLDs can easily implement such applications.

### 3.2 Data Paths and Associated Control

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. This is particularly true for some RISC processors that provide minimal support for interfacing to other chips. As an example, the MIPS R4400 requires a complex external circuit termed an “external agent” for interfacing the processor to other chips. The external agent is required to massage both the 64-bit data and 64-bit address bits from (and to) the R4400.

There are many situations in which wide data path circuits are needed for computer system design, some examples of which are given below.

#### 3.2.1 Wide Data-Paths

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.

The logical processing in such data-paths may be time-critical, but another strong requirement is the number of I/O pins. 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.

![Figure 2 - Small Finite State Machine.](image-url)
some cases it may be possible to implement an entire wide data-path circuit in a single large FPGA, but it is often desirable to partition the circuit into “bit slices” and assign each bit slice to a smaller, less expensive device. Multi-device partitioning is usually not less expensive in total than using one large chip, but smaller FPGAs are often available in faster speed grades than larger ones, and so better performance of the design can be achieved.

An example of a wide data-path circuit is illustrated in Figure 3, which shows an arrangement of registers and multiplexers used in the NUMAchine multiprocessor as part of the interface to the MIPS R4400. Although not shown in the figure, this circuit was (manually) partitioned into four bit-slices, each of which was similar, but not identical. To provide a concrete example of speed-performance for this type of data-path, we mapped it into the fastest speed-grade available in the Altera FLEX 8000A FPGAs. In this case, the speed-performance achieved was between 40 and 50 MHz, averaged over the four bit slices. While this performance is very high for an FPGA (FLEX 8000A provides among the highest speed-performance of SRAM-based FPGAs), it was not acceptable for our design, which needed to run at more than 50 MHz.

It is widely accepted by designers who use PLDs that FPGAs are the best choice for data-path circuits, because wide logic gates are not required and the number of flip-flops needed is large. Given this, it is surprising that we were not able to achieve the required performance in an FPGA, but we were successful when targeting a CPLD. The four bit-slices were each mapped into an Altera MAX 7000 CPLD, and the resulting speed-performance averaged 83 MHz (well beyond our requirements). The reason that the CPLDs provided better performance has to do with 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 because it shows that the “accepted” guidelines for which type of PLD to use cannot be followed blindly; both CPLDs and FPGAs should be investigated when speed-performance is the primary concern.

### 3.2.2 Control for Data-Paths

Circuits that control data-paths have a very specific structure. Even if the data flows in both directions, the data path can only be opened in one direction. Depending on the complexity and the amount of multiplexing, control can be realized by one, or more state machines. If there is one state machine, it controls both directions. If there are multiple machines their combined realization will be simpler, but they then have to be synchronized.

Each of these machines possesses a characteristic structure of a tree-like FSM diagram, where the main branching is done in the first state. Other than that, the decisions that would change the control flow are
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 for generation of the next states and outputs is as simple as possible. State assignment problems are very difficult, because it is hard to estimate the complexity of the logic required for any assignment (NP complete problem), and the number of possible assignments is large (for n state bits and S states there are \(2^n! / (2^n - S)!\) assignments). State assignment has been extensively studied, and approximate solutions can be outlined only when two-level AND-OR architectures are targeted. 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 assignment of codes to states for CPLDs and FPGAs. In FPGAs, one-hot encoding seems to be the best choice [1, 2], especially if joined with the standard FSM decomposition and state splitting techniques. One-hot encoding performs well in CPLD architectures also, provided that there are enough flip-flops available. Indeed, our experience shows that most medium-size (10-20) state machines perform best using one-hot encoding in CPLDs.

When a controller requires more than about 20 states, it is beneficial to examine assignments other than simple one-hot. We will describe here one heuristic that we found useful for a state machine that controls the data-path circuit described in the previous example. The state machine has 24 states, with 5 main branches and from two to eight states per branch. Many state encodings were tried for this example. First, we allowed the CAD software to automatically choose state assignments. The circuit would not fit into a MAX 7000 CPLD, but would fit in a FLEX 8000 FPGA, with a maximum clock rate was just 9 MHz! Next, one-hot encoding was tried, but the circuit still couldn’t fit in a CPLD. A valid state assignment was found using the following heuristic. State bits were assigned in a one-hot manner for each branch of the machine. Then, within each branch, additional bits were assigned in a way that simplified the logic to generate the outputs (this method is called face embedding for input encoding [3]). Finally, states with a large fan-in were assigned as many 0’s as possible to minimize the number of product terms feeding a flip-flop. Using this technique, the realized circuit used nine product terms and achieved a speed-performance of 67 MHz.

It was important to keep the number of product terms small because the CPLD being used supports faster paths for lower numbers of product terms; in other words, one must always consider the exact structure of the PLD if maximum speed-performance is to be obtained. This can sometimes involve many design iterations and much experimentation.

### 3.3 High-Speed Counter Design

In this section, a high-speed counter is described and refined. Implementation is considered in both the Altera MAX 7000 CPLDs and the FLEX 8000 FPGAs.

#### 3.3.1 Problem Definition

In our multiprocessor system, high-speed, loadable counters are required to count a number of hardware events that can occur each clock cycle at speeds up to 75 MHz. Consequently, they must be fast and large enough so they don’t overflow too quickly. For example, if an event being measured occurred on every 75 MHz clock cycle, a 32-bit counter would overflow in 57 seconds. This overflow interrupts a CPU, so the frequency of overflows should be minimized.

Although our real design requires a number of counters to reside on the same PLD, we will concentrate here on the development of a single counter. Our experience has shown that it is necessary to optimize the
single counter for speed and routing flexibility because both of these characteristics degrade when multiple counters are placed together in one PLD.

### 3.3.2 Basic Counter Design

A basic counter, based upon Altera’s 74161 macrofunction, was implemented in both MAX 7000 and FLEX 8000. By cascading eight of these parts, a 32-bit counter is formed. Altera has conveniently provided two implementations of this basic counter, one optimized for CPLDs and one optimized for FLEX 8000 parts, showing that design for CPLDs and FPGAs is different. Implementation results achieved using the fastest available parts are shown in Table 2.

The table shows that the CPLD-based counters achieve the maximum possible speed of $t_{pd}$ plus the setup time of the flip-flop (up to 32 bits, achievable speed for a counter is not dependent on the size of the counter!). This is because each counter bit needs up to 34-input AND functions that feed 4-input OR-gates, and the MAX 7000 implements this easily. For the FLEX devices, wide product terms are not available. Instead, a high-speed carry chain is employed to implement the required wide AND. It is the speed of this carry chain that limits the speed-performance of the FLEX-based counter. Roughly, a counter of double the size has twice the carry chain length and thus half the speed performance.

However, experience has shown that use of the carry chain hinders routability. The problem is that the carry chain is a fixed resource, so all bits of the counter must be physically placed in an ordered, packed format. This restriction makes routing more difficult because the FLEX interconnect is limited: Altera relies upon shuffling logic blocks during fitting to make up for the limited resources. Unfortunately, when carry (and cascade) chains are used the tools can no longer shuffle logic blocks, so routability is greatly impeded.

### 3.3.3 Improved Counter Design

Although the CPLD devices implement very fast counters, we require multiple counters on one chip. Because the MAX devices do not have enough logic cells or registers to implement a large number of counters, we must improve the FLEX-based counter. One high-speed alternative to a counter is to use a linear feedback shift register, but 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. By dividing the counter into sub-blocks, the slow carry chain is broken and the counter can run faster. The 32-bit counter was broken up into sub-blocks of size 1, 2, 1, 8, and 20 bits; this strange

<table>
<thead>
<tr>
<th>Part</th>
<th>Speed Grade</th>
<th># Logic Cells (% of Device)</th>
<th>Maximum Operating Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX 7096QC100</td>
<td>-7</td>
<td>32 (33%)</td>
<td>125 MHz</td>
</tr>
<tr>
<td>MAX 7128EQC100</td>
<td>-10</td>
<td>32 (25%)</td>
<td>100 MHz</td>
</tr>
<tr>
<td>FLEX 8452AQC160</td>
<td>-3</td>
<td>34 (10%)</td>
<td>50 MHz</td>
</tr>
<tr>
<td>FLEX 8452AQC160</td>
<td>-3</td>
<td>39 (11%)</td>
<td>54 MHz and 69 MHz*</td>
</tr>
</tbody>
</table>

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

\* Higher speed was achieved by manual placement of critical elements.
partitioning was chosen because it was convenient for blocks to be a multiple of 4 bits so that the 74161 macrofunction could be used. Although this partitioning works well, it is not necessarily optimal.

The initial implementation of this design yielded very poor results; the counter ran at speeds close to 20 MHz. By tuning the logic according to the architecture, performance was increased to 54 MHz, an 8% improvement over the basic counter. Further performance was extracted by hand placing the critical logic elements of the counter: roughly 25% of the logic cells had to be manually placed to boost the speed to over 69 MHz, a 38% improvement. These results appear in the bottom row of Table 2. By further tuning of the partitioning and logic, faster and wider counters should be possible to meet the 75 MHz requirement.

The improved counter uses only five more logic cells than the basic counter, yet it is more routable because the long 32-bit carry chain has been broken up into smaller chains of 8 and 20 bits. This flexibility makes it easier to route FLEX chips containing numerous counters, even though some of this flexibility is lost during the hand-placement.

In summary, Altera CPLDs can implement very fast counters, however, it is difficult to construct many large counters in one device. The FLEX FPGAs are 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 small cost in area. However, these gains cannot be realized without intimate knowledge of the FPGA architecture and a great deal of manual, meticulous 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