Architecture and Circuit-Level Design of an SRAM-Based Field-Programmable Gate Array

Paul Chow, Soon Ong Seo+, Jonathan Rose*, Kevin Chung%, Immanuel Rahardja, and Gerard Paez@*

**Department of Electrical and Computer Engineering*

+ATI Technologies, Toronto,

%Xilinx Toronto Development Centre

@National Semiconductor

University of Toronto

Abstract

Field-Programmable Gate Arrays (FPGAs) are now widely used for the implementation of digital systems and many commercial architectures are available. Although the literature and data books contain detailed descriptions of these architectures, there is very little information on how the high-level architecture was chosen and no information on the circuit level or physical design of the devices. This paper describes the design of an SRAM-programmable FPGA, starting from the high-level architecture through the circuit design issues to the physical layout

The logic block and routing architecture of the FPGA was determined through an experimentation process with benchmark circuits and custom-built CAD tools. The resulting logic block is an asymmetric tree of four-input lookup tables that are hard-wired together and a segmented routing architecture with a carefully chosen segment length distribution. At the circuit design level, we address area-speed trade-offs in the design of the logic block circuits and in the connections between the logic and the routing structure.

All commercial FPGA designs are done usng full-custom hand layout to obtain absolute minimum die sizes. This is both labor and time-intenstive. We propose a design method with a 'mini-tile' that contains a portion of all the components in the logic tile, and so takes less full-custom effort. The mini-tile is replicated in a 4x4 array to create a macro tile. The mini-tile is optimized for layout density and speed, and is customized in the array by adding appropriate vias. This technique also permits easily changing the hard-wired connections in the logic block architecture, and the segmentation length distribution in the routing architecture.

1.0 Introduction

Field-Programmable Gate Array (FPGA) technology permits the design of many different complex digital circuits using a single off-the-shelf device [Brow92]. The time-to-market pressures and low financial risk has made FPGAs and Complex programmable logic devices (CPLDs) an increasingly popular vehicle for prototyping, and in many cases, actual production. There are many different architectures now available from over fifteen silicon vendors. Although the databooks and related publications usually describe the architecture in detail, there is little information on *how* the architecture was chosen, and no information on the circuit level or physical layout level of the design. The contribution of this paper is to describe the complete design of an SRAM-programmable FPGA, including this information. Our primary goal was to produce a high-speed architecture and circuit, with a reasonable logic density.

FPGAs were first introduced in 1986 by Xilinx using a memory-based programming technology [Cart86]. Since then there have been many new commercial architectures [Hsie90] [Brit93] [Clif93] [Smit93] [Tava95] and several new programming technologies including two types of anti-fuse [ElGa89] [Birk92] [Marp92] and floating gate transistors, which are UV and electrically erasable [Wong89] [Gupt90] [Pate95].

A few non-commercial FPGA architectures have been reported for which the design details are more readily available. The Triptych FPGA [Ebel91] matches the physical structure of the routing architecture to the fanin/fanout nature of the structure of digital logic by using short connections to the nearest neighbours. Segmented routing channels are used between the columns to provide for nets with fanout greater than one. No discussion is given about how the segmentation length distribution was selected. This routing architecture does not allow the arbitrary point-to-point routing available in general FPGA structures. The logic block implements logical functions using a multiplexer-based three-input lookup table followed by a master-slave D-latch, and can also be used for routing. Initial results show potential implementation efficiencies in terms of area using this structure. The Montage FPGA [Hauc92] is a version of the Triptych architecture modified to support asynchronous circuits and interfacing separately-clocked synchronous circuits. This is achieved by the addition of an arbiter unit and a clocking scheme that allows two possible clocks or makes the latches transparent.

Earlier work at the University of Toronto resulted in the implementation of an architecture (UTFPGA1) using three cascaded four-input logic blocks and segmented routing [Chow91]. UTFPGA1 used information from previous architectural studies but there was very little transistor-level optimization (for speed), and little time was spent on layout optimization. This was a first attempt that provided some insight into the problems faced in the design and layout of an FPGA. An earlier version of the work presented here appeared in [Chow93].

Modern trends in the design of computer architecture have shown the importance of considering both the compiler technology and the hardware technology at the same time, when designing for high performance [Henn90]. This is also true in the design of an FPGA, where it is important that the CAD tools collaborate with the architecture of the FPGA. At the University of Toronto, we have developed a number of custom CAD tools that have been used to perform “experimental” architectural studies. In this paper, we will describe two studies that influenced the architecture of our second-generation FPGA. We will also introduce a novel layout style for FPGAs that builds the basic tile using a set of identical mini-tiles and customization by the addition of vias. This allowed us to still achieve reasonable density while significantly reducing the time spent doing custom layout. These features have all been implemented in our test chip, called LEGO (**L**ogic that’s **E**rasable and **G**reatly **O**ptimized).

In the following section we summarize the results that led to the choice of our logic block and routing architecture. Section 3.0 discusses the circuit design issues and the layout of the chip is presented in Section 4.0. The performance of the chip and the design summary are provided in Section 5.0.

2.0 Architecture

The speed and density gap between FPGAs and MPGAs is mostly due to the routing structures used to connect logic components in each technology. In MPGAs, the logic elements are connected with mask-programmed metal wires. In FPGAs, logic block pins are connected using field-programmable switches. Regardless of the type of programmable switch used in the FPGA (e.g. static RAM-controlled pass transistors [Hsie90] or anti-fuses [Ahre90], or floating-gate based switching [Pate95]), the capacitance, resistance, and size of programmable connections makes them much slower and larger than a simple metal wire. In this section we describe two important architectural innovations that lead to higher-speed FPGAs: the creation of hard-wired interconnect between logic blocks, and the use of longer dedicated wire lengths.

2.1 Logic Block Architecture: Hard-Wired Logic Blocks

One way to improve the speed and density of FPGAs is to replace some of the programmable connections between basic logic blocks with hard-wired connections, which are simple metal wires [Chun91]. By using hard-wired links to construct more coarse-grained logic blocks (called Hard-Wired Logic Blocks, or HLBs) from several basic blocks, the delay and size of circuits can be reduced. For example, Figure 1(a) shows a network of basic blocks with five programmable connections in the routing along the critical path (which we define as the largest number of logic levels across all combinational paths, from block 1 to 5 in the figure) and nine programmable connections in total. Suppose that three of the basic blocks are hard-wired together to create a hard-wired logic block with a fast three-block path as shown in Figure 1(b). The resulting circuit in Figure 1(c) has only two slow programmable links instead of five along the critical path. This is a sizable reduction in routing delay. Also, the total number of programmable connections has been reduced from nine to four and this may lead to a reduction in routing area.

2.1.1 The Hard-Wired Logic Block Design Space

Given this general notion of hard-wired logic blocks, it is clear that there are many possible ways to implement it. Assuming that there is only one type of basic block that is connected in a given

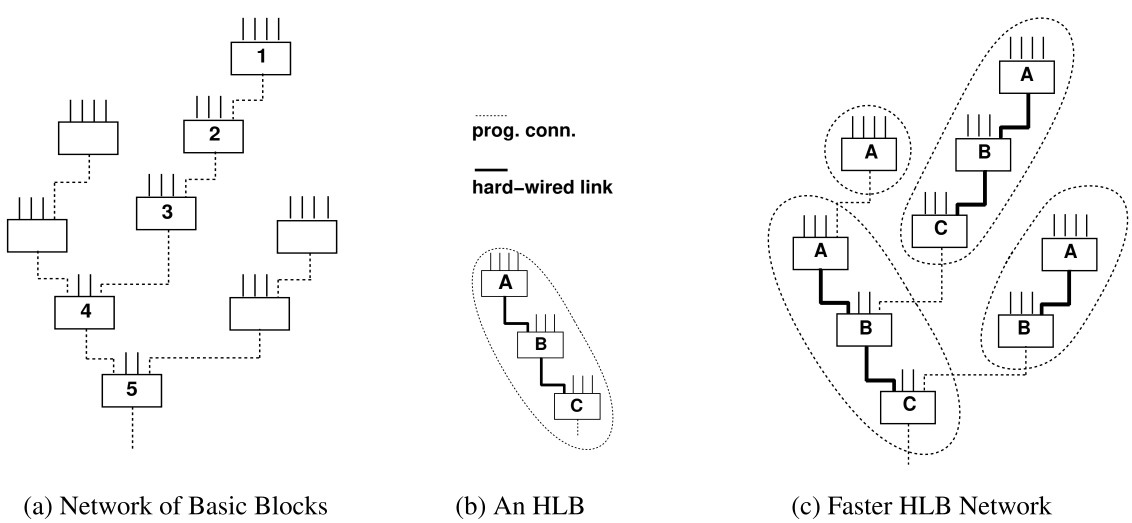

Figure 1 - *Hard-Wired Connections and Logic Blocks*

HLB, the choices lie in the topology of the interconnection between the blocks and the size of the basic block itself.

Each choice will provide different delay and area for a given circuit. In general, the more hard-wired links in a hard-wired logic block, the faster will be the circuits implemented using that logic block. However, a greater number of hard-wired connections provide less connection flexibility and may lead to lower density because basic blocks are wasted. In this section we give a brief description of the experiments performed to determine a good choice for the HLB topology. We omit the work done to determine the choice of basic block, which the interested reader can find in [Chun94], along with a more detailed description of these experiments.

The basic block we choose to work with is the 4-input lookup table (hereafter referred to as a 4-LUT) as previous studies have indicated that this is a reasonable choice for both speed and density [Rose89] [Rose90], [Sing91] [Sing92], [Koul92] and as such has been adopted by several commercial vendors [Hsie90], [Brit93], [Clif93].

We restricted the choice of topology to be a tree, as circuits often are tree-like in nature. Figure 2 illustrates several of the topologies that were investigated. The naming convention of these structures is as follows: it begins with the letter “L”, then the height of the HLB (in basic blocks), then a dash (“-”) followed by a listing of sizes of the subtrees from a pre-order traversal of the canonical HLB tree. Each subtree size is separated by a dot (“.”) with the restriction that leaf inputs and single-LUT subtrees are not listed.

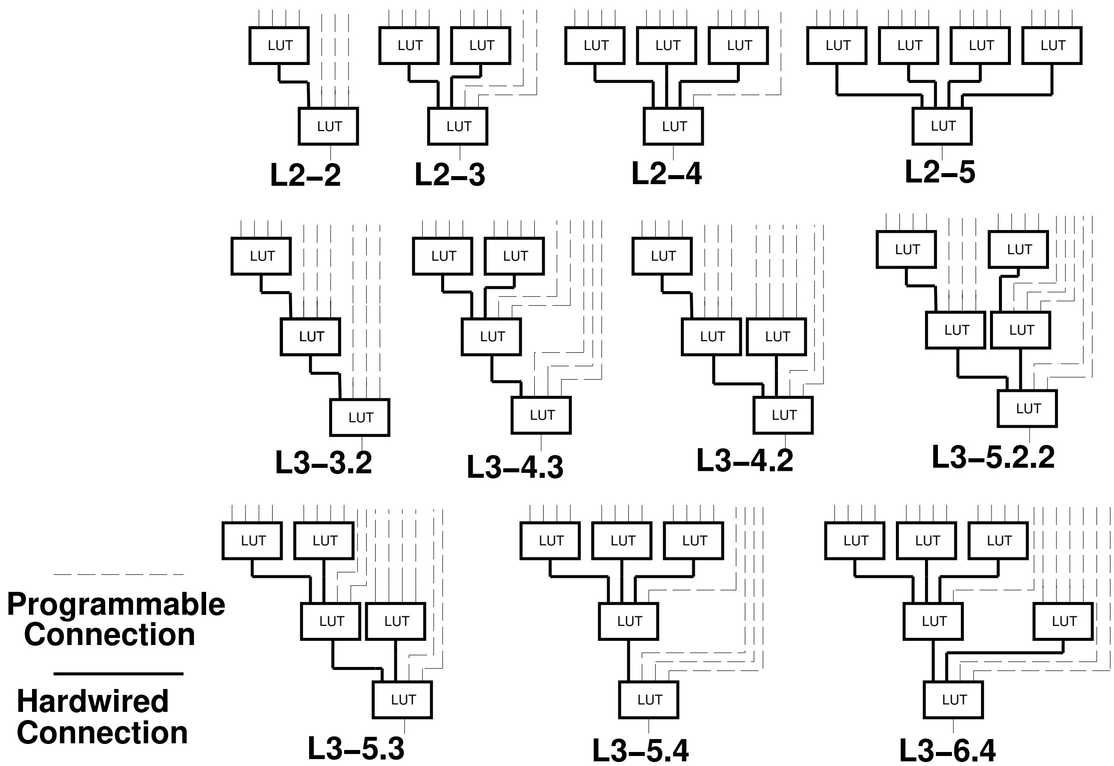

Figure 2 - *Example of Some of the Tree Topologies Investigated*

In addition to the HLB topologies illustrated in Figure 2, all possible tree topologies with nine or fewer 4-LUTs and three or fewer LUT levels were explored. Initial experiments indicated that although HLBs with more than nine 4-LUTs provide greater speedups they require too much area to be considered practical.

An important architectural assumption is that each HLB has a buffer on the output of each LUT basic block that is accessible to the programmable routing. This direct access has two important advantages: delay is reduced because an output can be accessed without propagating it through downstream logic blocks, and density is increased because unrelated pieces of logic can be packed together in a multi-output HLB.

The goal of the experiments described below is to select from among the many possible hard-wired 4-LUT architectures one that produces high system speeds, at a reasonable cost in area.

2.1.2 Experimental Method

To evaluate the HLB topologies, a set of 15 benchmark combinational circuits from the MCNC suite [Yang91] were each "implemented" as a set of FPGAs with each HLB topology. The area and delay of each implemented circuit is then calculated using area and delay models, and the results are averaged over all circuits for each HLB topology. By "implementation" we mean that each circuit is processed through a suite of CAD tools that take it from the logic level (equations) through to the physical placement upon the hypothetical FPGA and the global routing on that FPGA. Note that a custom logic synthesis tool, called TEMPT, was developed to allow synthesis into the hard-wired structures [Chun92].

The focus of this paper is not the details of these experiments but the results. As such the description of the area and delay models and the CAD tools needed to provide the implementation are omitted but can be found in [Chun94]. The only detail necessary for the discussion below is the fact that the CAD implementation stream has two modes: one in which the resulting FPGA circuit is optimized for speed (sometimes at great cost in area), and the other in which the circuit is optimized for area at the cost of some speed. Our method of selecting an appropriate logic block for LEGO is to inspect the results of both experiments using both optimizations and to deduce a reasonable compromise.

2.1.3 Experimental Results

Table 1 provides the summarized results of the implementation of the benchmark circuits for those HLB topologies that appear the most promising. Table 1 gives the highest-speed (lowest critical path delay) topologies for each *size* of HLB, where size is measured by the number of 4-LUTs in the HLB, when the circuits are optimized for both speed and area.

The first column of Table 1 lists the number of 4-LUTs in the HLB. If there are two rows for a given number of 4-LUTs, the second entry is an alternative HLB that offers a lower speed but at a lower area cost. The second column gives the label of the HLB topology, as described above. The third and fifth columns give the normalized speedup of the HLB with respect to L1 (a 4-LUT FPGA with no hard-wired links), for the speed and area-optimized implementations. The speed is calculated as the geometric average of the inverse of the critical path delay for each circuit. It is then normalized with respect to L1. The fourth and sixth columns of Table 1 give the normalized area with respect to L1, for both the speed and area cases. The un-normalized absoluted values for speed and area are given in the L1 row, assuming a 1.2 μm CMOS process. Absolute areas come from actual layouts of cells used in an area model. Absolute times come from a delay model, which incorporates delay data derived from the simulation of the cell layouts.

.

Number of LUTs in HLB	HLB Topology	Speed-Optimized Circuits		Area-Optimized Circuits	
		Speed Factor	Area Factor	Speed Factor	Area Factor
1	L1	1 (35 MHz)	1 (15.3 x 10^6 $\mu m^{2)}$	1 (22 MHz)	1 (6.7 x 10^6 $\mu m^{2)}$
2	L2-2	1.14	1.19	1.17	0.93
3	L2-3	1.26	1.21	1.17	0.96
4	L2-4	1.34	1.32	1.09	1.00
4	L3-4.2	1.28	1.31	1.27	0.93

Table 1 - *Speed and Area of HLBs-based FPGAs, Speed-Optimized Circuits*

Number of LUTs in HLB	HLB Topology	Speed-Optimized Circuits		Area-Optimized Circuits	
		Speed Factor	Area Factor	Speed Factor	Area Factor
5	L2-5	1.42	1.41	1.03	1.10
5	L3-5.2	1.37	1.37	1.24	1.04
6	L3-6.5	1.43	1.65	1.04	1.23
7	L3-7.3	1.43	1.66	1.19	1.28
8	L3-8.3.2	1.47	1.61	1.25	1.32
9	L3-9.4.2	1.50	1.70	1.24	1.42

Table 1 - *Speed and Area of HLBs-based FPGAs, Speed-Optimized Circuits*

The speed-optimized results show that the hard-wired links can increase system speed from 14% up to 50% compared to a flat non-hard-wired 4-LUT when the circuits are speed optimized. However, the area cost of this speed is an increase of 19% to 70% in circuit size. Some of this increase is caused by wasteful logic synthesis tools that sacrifice area for speed, but it also comes from the fundamental speed-area trade-off with hard-wired links as discussed above.

The area-optimized results show that several topologies are actually *more* area-efficient that the flat 4-input lookup table. All topologies provide a gain in speed, although less than that of the speed optimized case.

Our criteria in choosing a block is to select one that gives a good combination of speed and good density.

From Table 1, we reject those blocks that have too high an area penalty: those in the last four rows. Of the remaining blocks, L-3.5.2, L2-5 and L3-4.2 provide the best speed up. However, the L-3.42 block proved superior in the area-optimized case to the others, and so it was selected. Of note is that this block provided the best logic density overall, while also providing significant speed up. The L3-4.2 HLB is illustrated in Figure 2.

2.2 Routing Architecture

The FPGA routing architecture is the most important determining factor of system speed and logic density because the programmable switches (which are pass transistors driven by SRAM cells) have significant resistance and capacitance and require large area. The routing architecture, is the manner in which wire segments and switches are organized.

In the original Xilinx 2000 and 3000 architectures [Cart86] [Hsie88], a very simple architecture was employed: most wire segments spanned only the length (or width) of one logic block, and had switches at each end. One way to improve the speed of connections that travel long distances is to provide segments that span multiple logic blocks without being switched, as illustrated in Figure 1. In this way, a segment with appropriate length for a connection can be selected, which results in less resistance along the path. The general notion of segmented routing was first introduced in [ElGa89].

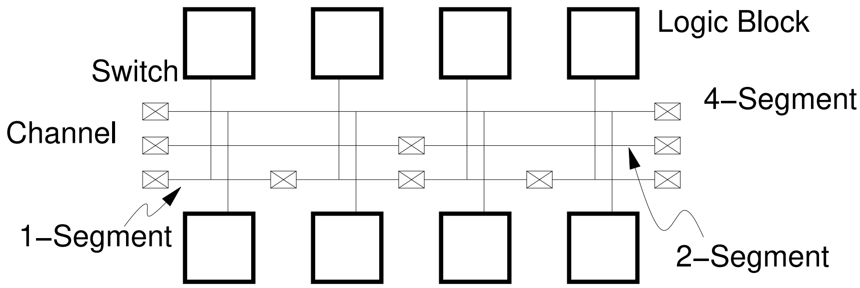

Figure 3 - *Segmented Routing Architecture*

A key architectural question is the determination of the number of segments of each length, called the *segmentation length distribution*. The greater number of longer segments, the greater the likelihood that long and otherwise time-consuming connections will be routed with fewer series switches, resulting in a faster circuit. An excessive number of longer segments, however, will mean that some of the long segments will have to be used for short connections, resulting in the waste of part of the segment. This waste leads to a decrease in logic density. Thus it is important to find a segmentation distribution that improves speed, but does not waste too much area.

There is a second important architectural feature of segmented routing architectures besides the distribution: a segment is called internally *populated* if it is possible to make connections from the middle of a segment to logic blocks or other routing segments. The advantage of un-populated segments is that it has less parasitic switch capacitance connected to the segment, which makes it faster. The disadvantage is that the reduction in routing flexibility (without population there cannot be internal fanout) may result in the need for more tracks and so a loss of logic density.

In this section we briefly summarize a study on segmented routing architecture distribution and population [Paez95] that was used as the basis for decisions on the routing architecture for LEGO. The basic approach was experimental, similar to the one described in the previous section: several benchmark circuits are "implemented" as FPGAs, each with a different segmentation distribution. These experiments required several CAD tools, including a detailed router specifically designed for FPGAs [Brow91]. The number of tracks needed to complete the route using segments of only length one is called the minimum. When the channel includes segments of other lengths, the number of tracks required above the minimum is measured.

2.2.1 Definitions and Experimental Method

Figure 1 illustrates the broad architectural features of the LEGO FPGA. The logic block (L) has pins that are connected to routing channels in a structure called a *connection block* (C). The *switch block* (S) provides connectivity among its attached channels. A channel consists of C and S blocks. Each channel contains W routing tracks. We assume that each track consists of only one type of segment length. We will consider only segments of length one, two, and three, which will be referred to as 1-segments, 2-segments and 3-segments.

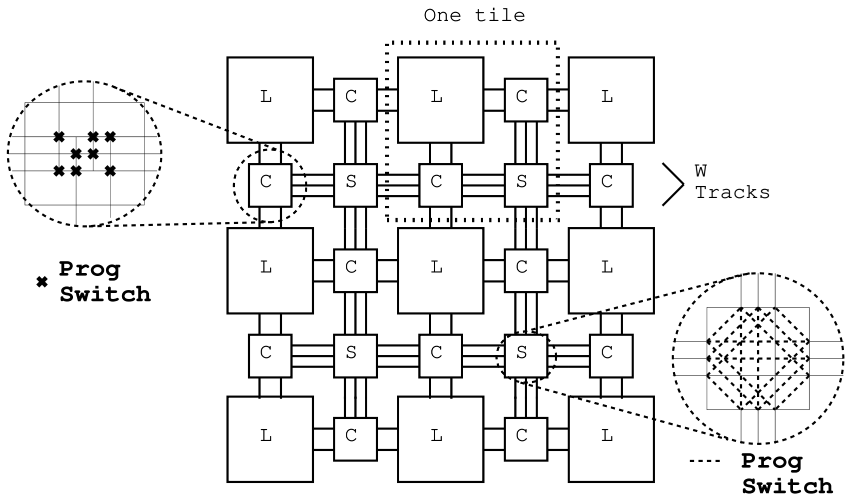

Figure 4 - *Architectural Definitions*

Let D_1 be the fraction of the W tracks that contain 1-segments. Similarly D_2 and D_3 are the fractions for 2- and 3-segments so that $D_1 + D_2 + D_3 = 1$.

Concerning the issue of the internal population of the segments, we divide it into two parts: whether or not the connection blocks internal to the segment (i.e., those not at its ends) should be populated, and whether or not the internal switch blocks should be populated. The following notation indicates the level of population: CB_u indicates that all unpopulatable connection blocks are un-populated, while CB_p means that they are populated. Similarly, for switch block population we use SB_u and SB_p. Thus there are four combinations of population. Notice that for 2-segments, only the middle switch block can be de-populated, and for 3-segments, there are two switch blocks and one connection block that can be de-populated.

2.2.2 Experimental Results and Decisions

The following experiments were performed: for all possible values of D_1, D_2 and D_3, using all four population combinations, the number of excess tracks needed over the minimum (for $D_1 = 1$) were measured and averaged for five benchmark circuits. We addressed the questions of population and distribution using these experiments. Note that the logic block used in these experiments was a single 4-input LUT, with a D flip-flop. While this was not the block chosen for LEGO, we believe the general results discussed below will hold for most logic blocks.

Population of Segments

The first question we address is that of population of the S and C blocks. Figure 1 is a plot, for $D_2 = 0$, of the average number of excess tracks versus D_1 for the five circuits. The four curves are the different combinations of population for the switch and connection block. Notice that as D_1 decreases from 1, D_3 increases from 0 ($D_3 = 1 - D_1$). Since this provides a greater number of longer segments with lower flexibility, we expect the number of tracks needed to successfully route will increase, and this is borne out in the figure.

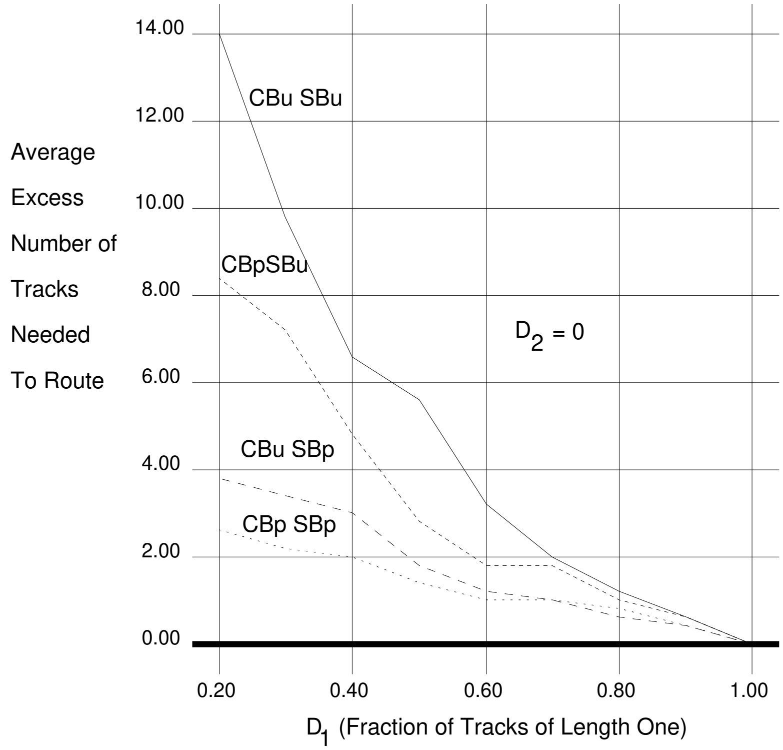

Figure 5 - *Population Comparison*

The data show that both cases in which the switch block is un-populated (the upper two curves) result in significantly more excess tracks. For this reason we decided to populate the switch block in LEGO. On the other hand, the bottom two curves (with the switch block populated) illustrate that only a minor increase in tracks is experienced when the connection block is de-populated. Since there is an advantage in speed for de-population (less capacitance on the track) we decided to de-populate the connection block in LEGO.

Segmentation Distribution

Table 2 gives the average number of excess tracks for many possible values of D_1, D_2 and D_3. Note that since $D_1 + D_2 + D_3 = 1$ there are only two independent variables, which we chose as D_2 and D_3 in the table. The rows vary D_2 and the columns vary D_3. This table is for architectures with both the C blocks un-populated and the S blocks populated, as discussed above.

Table 2 illustrates that even with 80% length three segments, that only about four extra tracks on average are required. (The minimum number of absolute tracks required for the five circuits ranged from 9 to 15.) We decided that we were willing to tolerate only about one extra track per channel above the minimum, so that the area cost of the higher speed tracks would be small. Thus architectures with D_2 in the range of 0.0 to 0.6 and D_3 in the range of 0.1 to 0.4 (those shaded in

Table 2) would reflect such a choice. The faster architectures are those with higher values of D_3.

D_2 \ D_3	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.0	0.0	0.4	0.6	1.0	1.2	1.8	3.0	3.4	3.8
0.1	0.2	0.6	1.0	1.2	1.6	2.2	2.8	3.4	
0.2	0.6	0.8	1.0	1.0	1.8	3.0	3.2		
0.3	0.6	0.6	1.2	1.8	2.2	2.4			
0.4	0.6	1.0	1.0	2.0	2.6				
0.5	0.8	1.2	1.4	2.0					
0.6	1.2	1.6	2.0						
0.7	1.4	1.8							
0.8	1.8	Absolute Avg #Tracks=12, Stand. Dev. = 2.2							

Table 2 - *Average Excess Tracks for All Distributions, Population* CB_uSB_p

In Section 4.3 we give the reasons that lead to the choice of 16 tracks per channel in LEGO. For the segmentation distribution, there is an additional constraint that arises from the one-tile style of layout that was used: there must be an even number of 2-segment tracks, and the number of 3-segment tracks must be a multiple of three. Taking this constraint and the above distribution into account, the following segmentation distribution was used in LEGO: nine of the 16 tracks are 1-segments, four tracks are 2-segments, and three are 3-segments. This corresponds to a segmentation distribution in which $D_1 = 0.56$, $D_2 = 0.25$ and $D_3 = 0.19$, which fits within the architectural range indicated above.

2.3 Architecture Summary

The architecture will be a symmetric array of the L3-4.2 HLB logic blocks, with a segmented routing architecture employing the distribution and population described above. Figure 1 illustrates such an array. We will refer to a *tile* as the portion of the array consisting of one logic block and the routing structures on two sides.

3.0 Circuit Design

In this section we discuss the circuit-level design of the FPGA architecture described above. The technology used in this design is a 1.2µm two-level metal n-well CMOS process.

Figure 2 illustrates the signal path starting at the input of an L3-4.2 lookup table, which was selected in Section 2.1. Following the lookup table is an output stage that contains a multiplexer to select the latched or unlatched version of the output before the signal is driven through a connection block (C block) onto the routing tracks. The signal then goes through one switch block (S block), or a number of S blocks, before it enters another C block where it will reach the input of the next logic block.

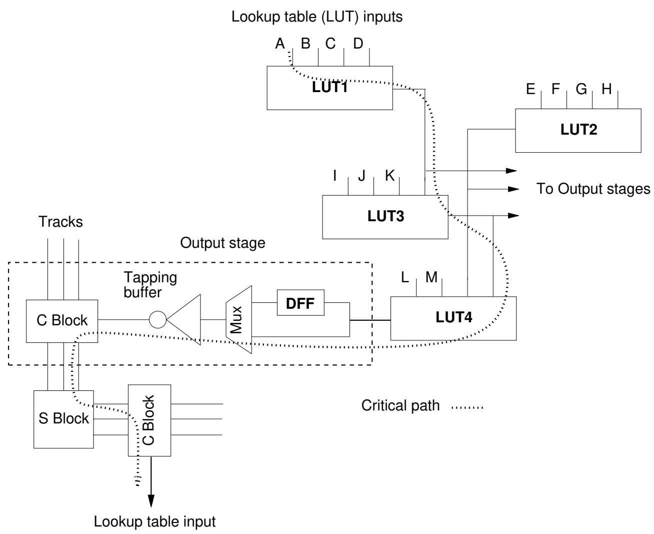

Figure 6 - *Signal path starting from logic block inputs*

In the following sections we discuss issues that arose in the design of this path. Simulation with HSPICE [Hspic90] was used to evaluate different circuit options.

3.1 Multiplexers versus Switches

The C block is used to connect signals in the routing tracks to the input pins of the logic blocks, and can be implemented either using multiplexers or point-to-point switches. Figure 2 illustrates the use of a multiplexer and Figure 2 shows a connection block using one switch for each track-to-pin connection. The boxes labelled SRAM are the programming bits. In UTFPGA1 [Chow91] we used multiplexers for the inputs and switches for the outputs. The reasoning was that since only one wire in the channel can be connected to an input pin, there is no need for the flexibility provided by switches. This reduces the number of bits needed to select the input connection and the total number of transistors needed. For the output, having a switch from a pin to each track provides fanout capability.

Table 3 compares the two possibilities on the basis of transistor count. The buffer inverters from the routing tracks are not counted. In LEGO, every C block is connected to two logic blocks with four inputs on each side. For the multiplexer technique, eight multiplexers with 30 transistors each are required to connect to the 16 tracks. This assumes that n-channel switches are used instead of full complementary switches, and that every track can be connected to every logic block pin. Four programming bits are needed to program each multiplexer as well as 32 inverters (4 inverters/mux x 8 muxes) to generate the complementary inputs. For the pass transistor technique, eight sets of 16 switches are required to connect to the 16 tracks. Each switch will require one programming bit. For either technique, each programming bit will require five transistors. The circuit for the programming cells is discussed in Section 4.1. Unit-sized transistors are used in the programming cells and the transistors that would be used in the switch or multiplexers are about three unit transistors in area. The result is that the pass-transistor technique is about 1024 unit transistors in area, compared to 944 for the multiplexer.

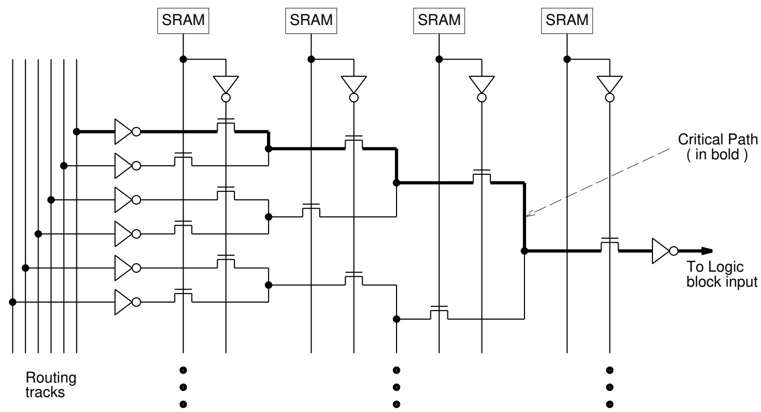

Figure 7 - *Connection block input using a multiplexer.*

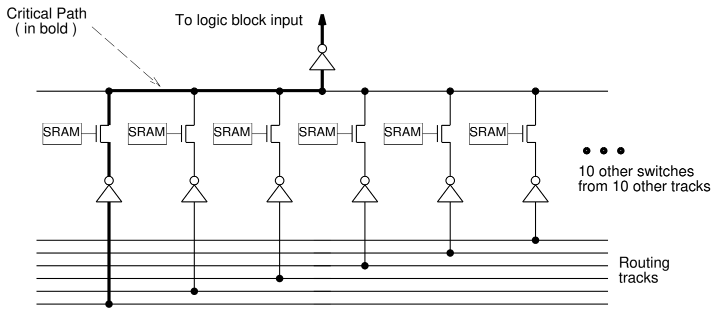

Figure 8 - *Connection block input using pass-transistors.*

Method	(Programming Bits) × (Transistors/Bit)	Inverters × 2	(Connection Transistors) × Scale	**Total Transistors**
Pass transistor	(8 × 16) × 5	0	(8 × 16) × 3	1024
Multiplexer	(8 × 4) × 5	32 × 2	(8 × 30) × 3	944

Table 3: *Area comparison for the two possible C block techniques.*

Although using a multiplexer results in less area, it has a significant performance impact because the signal must go through a number of series transistors. In LEGO we chose to use switches at both the inputs and the outputs of the logic block. The price is that a significant fraction of the area is devoted to the C blocks.

3.2 Isolation

Another important issue is the capacitance at the C block inputs. With switches connected as shown in Figure 2(a), the capacitance seen from the routing track depends on whether the switch is on or off. If a switch is off, the contribution to the capacitance is due to the switch transistor itself. If the switch is on, then any of the circuitry on the other side of switch will contribute to the load seen by the driving signal. This means that the loading will be a function of the number of C-block switches that are turned on. This is related to the problem noted in the design of the Triptych [Ebel91] logic block, and their solution was to use gate logic instead of switch logic. The problem in LEGO can be reduced by adding a buffer between the track and each switch input, which makes the load independent of how many switches are turned on. However, this still has one load per switch input. Further isolation can be provided by using the configuration shown in Figure 2(b). In LEGO, each connection block connects the tracks to two sets of logic block inputs, so an additional optimization can be done as shown in Figure 2, which reduces the number of buffers required by one half, also reducing the capacitance on the track.

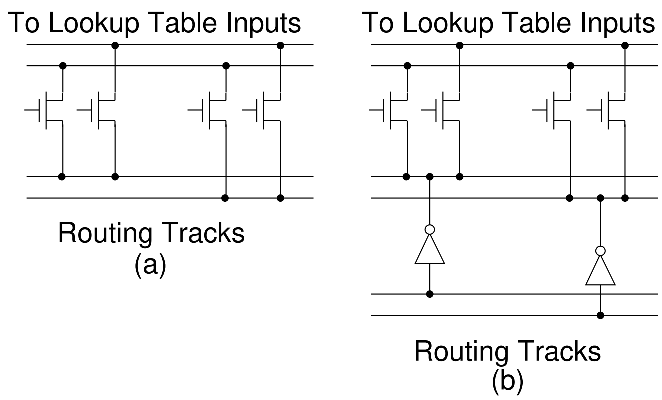

Figure 9 - *(a) C-block inputs with no receivers (b) C-block inputs with isolation.*

.

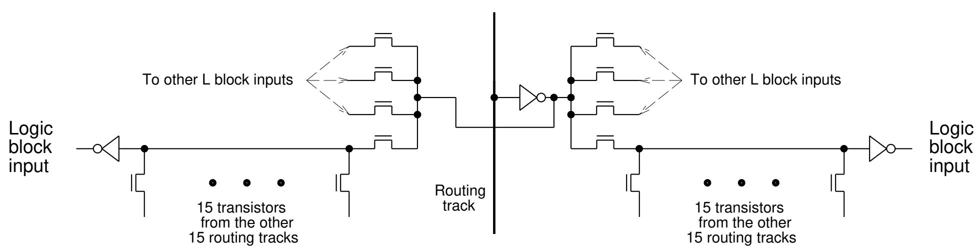

Figure 10 - *Track isolation using shared buffer.*

3.3 Lookup Table Design

For the LUT design, we have shown in Section 2.1 that hard-wired connections can significantly improve performance. At the implementation level, the actual placement of the hard-wired connections is also important.

Figure 2 shows a part of the multiplexer tree in the LUT. It shows that the load of input S is only one transistor and one inverter, while the load of input P is eight transistors and an inverter. Input

S is the least loaded input and also the input that determines the final output of the multiplexer. Therefore, when connecting the output of a LUT to the input of the next LUT, it is best to use the inputs that are closest to the output. In this way, the delay through the LUTs can be partially overlapped. Figure 2 shows the hard-wired connections used in LEGO. Note that the output of LUT2 is connected to input R of LUT4 (using the input labels of Figure 2), while the output of LUT3 is connected to input S. This is because the output of LUT3 will arrive the latest, so it should be connected to the fastest input, which is input S.

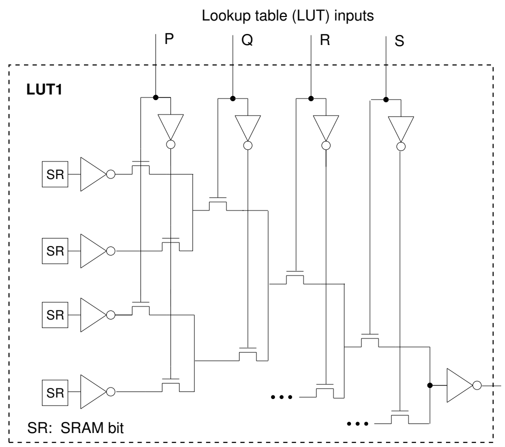

Figure 11 - *Top quarter of LUT multiplexer tree.*

3.4 Full Complementary Versus n-channel Only Switches

Routing switches are pervasive through the entire design. We explored whether the switches should be fully complementary or whether n-channel pass transistors would suffice. From an area perspective, the full switches would clearly be a disadvantage because both an n-channel and a p-channel transistor would be required, as well as possibly an extra inverter to provide the complementary signal needed to control the switch. However, we were also interested in optimizing for speed, so we also examined the trade-offs with respect to speed in the LUT, the C block and the S block. Figure 2 shows the results comparing the delay through a LUT implemented with full switches (labelled **fsw** in the figure) and one that only uses n-channel transistors (labelled **ptr**). It is clear that using pass transistors is the best choice when looking at both the rise time and the fall time. The result is the same for the C block and the S block. This means that using n-channel transistors is the best choice both for area and for speed. Even better performance could be achieved if the threshold drop through the pass transistors could be avoided. Commercial FPGAs often do this through process or circuit techniques.

3.5 Power-up and Programming Protection

A difficult problem occurs between the time after chip power-up but before the programming has been downloaded. At this point the configuration bits have not been programmed and it is possible that there are conflicts between drivers on the routing tracks. This problem has been considered previously. In UTFPGA1 [Chow91] the output drivers are disabled using a tristate buffer as shown in Figure 2(a). The buffers will be disabled during power up and programming. This design, however, impacts the speed because the disabling transistors of the tristate buffer will be in the critical path. Figure 2(b) shows the method used in LEGO. The output routing switch is controlled not only by a configuration bit, but also by another signal, shown as **Progb**, which is a global signal activated during power-up that disables all connections to the track, preventing the hazard.

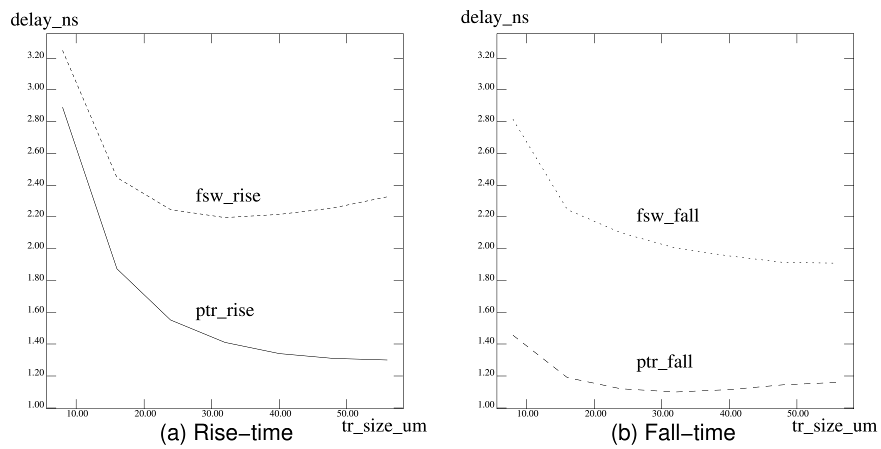

Figure 12 - *Comparison of rise and fall times using full switches (fsw) and pass transistors (ptr) in a LUT.*

.

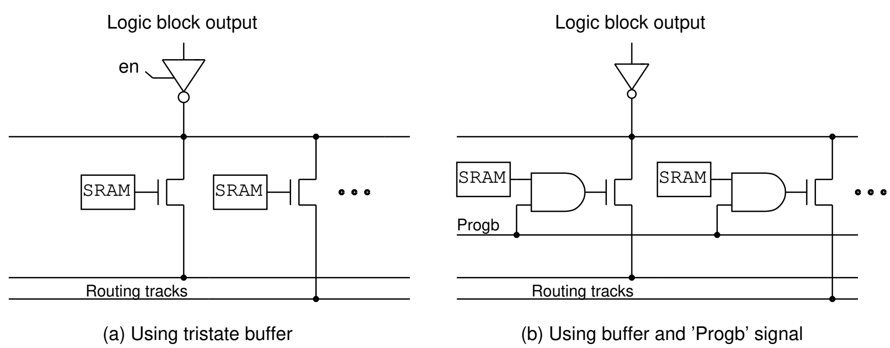

Figure 13 - *Hazard prevention alternatives.*

3.6 Transistor Sizing and Signal Transition Balancing

Transistor sizing is a major concern for achieving high performance. Small transistors limit the current while large transistors have a higher capacitive load. Therefore, some optimization is required. In Triptych [Ebel91], the method of Logical Effort [Suth91] was used to assist in transistor sizing and buffer insertion for speed. We chose to use actual simulation because we felt it was easier to balance the rise and fall times. We began by simulating the components of the critical path individually but later tuned the design by simulating the entire critical path at once.

Our initial assumption was that the n-channel pass transistors used for the S block switches and C block outputs had to be at least as large as the transistors of the drivers so as to reduce the effects of the switch sizes on the critical path delay. This allowed us to choose an initial driver size. We then tuned the design by simulating with different switch sizes. An example result is shown in

Figure 2, which shows the effect of varying the logic block output driver size on the total delay of the critical path. It can be seen that the rise and fall times are equal at about 37µm for the size of the n-channel driver pulldown. In LEGO we actually used 34µm because in our initial simulations we used nine tracks in the channels, which was later changed to 16 tracks, and thus a higher load.

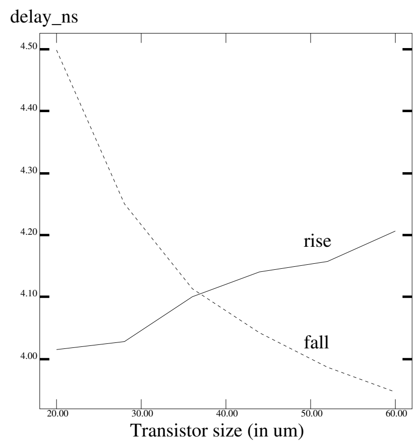

Figure 14 - *An example of varying switch sizes for transistor sizing.*

It should also be noted that the tuning was done by starting at the end of the critical path. This gives a better idea of the loads to be driven as sizes are fixed.

Figure 2 shows the critical path through the logic block section, and Figure 2 shows the critical path in routing using the tuned transistor sizes. From simulation, the total delay is about 4.1ns along the critical path broken into about 2.7ns for the logic block and 1.4ns for the routing.

4.0 Physical Design and Layout

Besides the concerns for optimizing the speed and area of the circuits, we also considered some of the issues related to the overall physical design and layout of the chip that would be used to implement the FPGA circuit. In this section we will describe the programming architecture of LEGO and some of the layout issues.

4.1 Programming Architecture

Three techniques for programming the configuration bits were identified. The simplest is to connect all of the bits into a long shift register. This requires only one pin to input the programming data. Although this is the slowest method, the speed of programming was not of concern in this project. Of greater concern is that each bit must contain a master-slave section because it behaves as a shift register, and hence each bit is very large. An alternative, which requires smaller bit cells, is to organize the programming bits into rows and columns as in a regular memory. This will require significantly more pins for the parallel data as well as for selecting the word to be programmed.

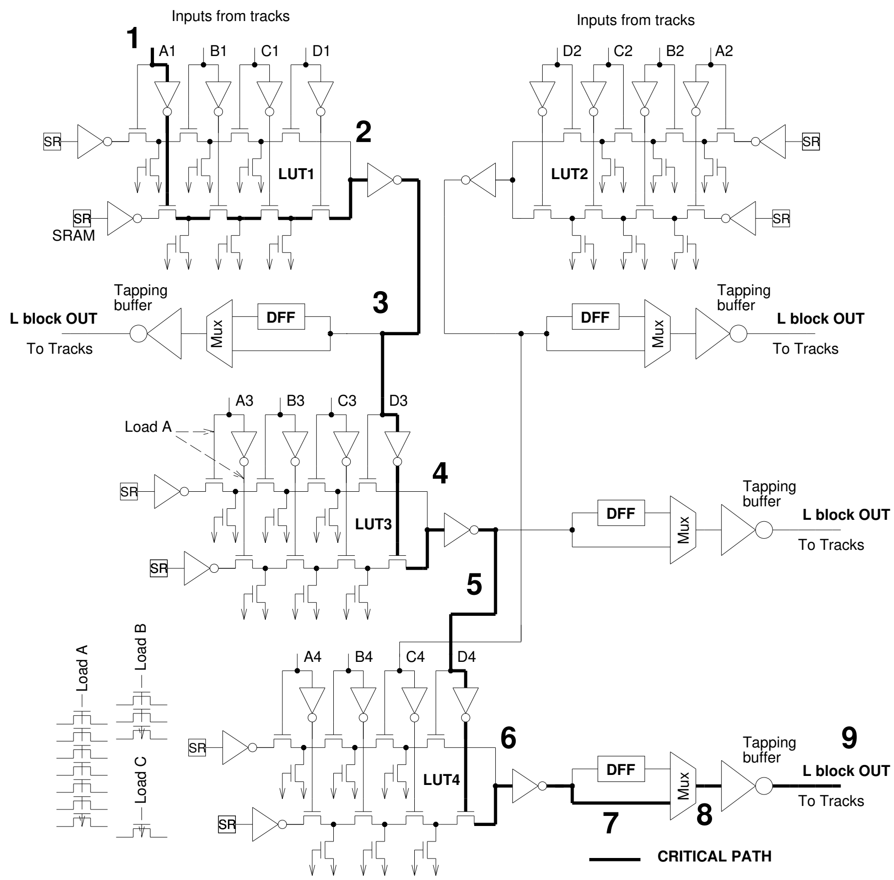

Figure 15 - *Details of critical path through the logic block.*

A compromise of the serial and parallel techniques was used in LEGO and is shown in Figure 2. This serial-parallel technique uses a master-slave shift register to serially load a row of bits. The vertical shift register is initialized with the top bit on and the remainder off. This bit is shifted after each row is programmed to select the next row. After a row of bits has been loaded in the top shift register, the whole row of data is copied into the selected row in one clock pulse. The static RAMs that store the data bits do not have to be master-slave because the data is not shifted beyond this point. This halves the size of a programming bit as compared to the scheme that uses a single shift chain. This technique still leaves the option of programming any row, as in the fully parallel method, but it requires fewer programming pins.

The regular structure of the serial-parallel architecture also fits well with the layout methodology used for LEGO, which is described next.

4.2 The Mini-tile Layout Style

We wanted to find a better layout style than just trying to attack the problem purely as a full-custom design. When doing VLSI layout, it is desirable to find some regularity in the topology to make the task easier. In an FPGA, the first level of regularity is the arraying of the tiles as shown in Figure 1, where a tile includes an L block, an S block, and two C blocks. This partitioning was used for UTFPGA1 and in LEGO. However, even the full custom layout of a tile is a significant task, and it is desirable to find even more regularity. In UTFPGA1 we attempted to use the same C block layout for both the vertical and horizontal channels but this resulted in a very poor tile floorplan because it was difficult to pack the pieces together. For LEGO we wanted a solution that had a more fine granularity.

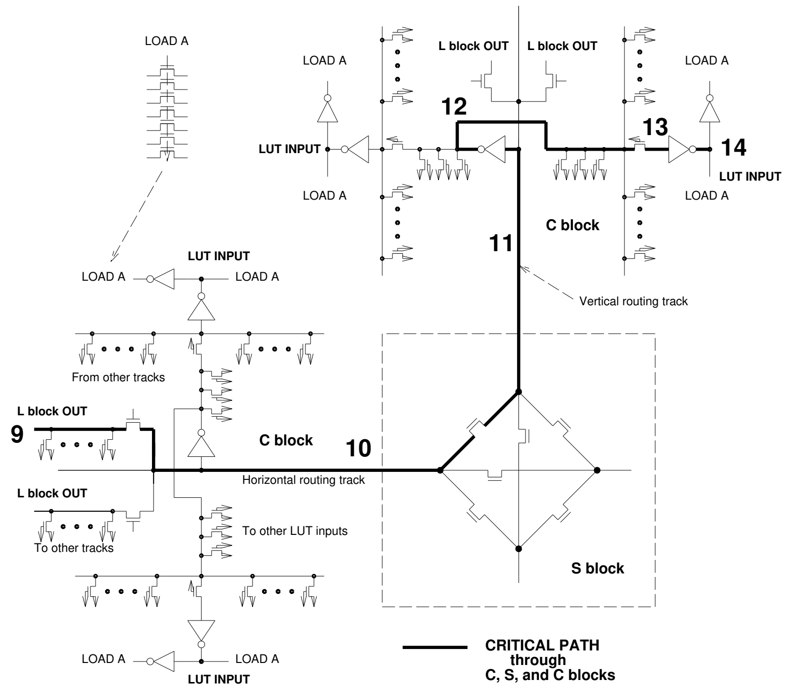

Figure 16 - *Details of critical path through the routing network.*

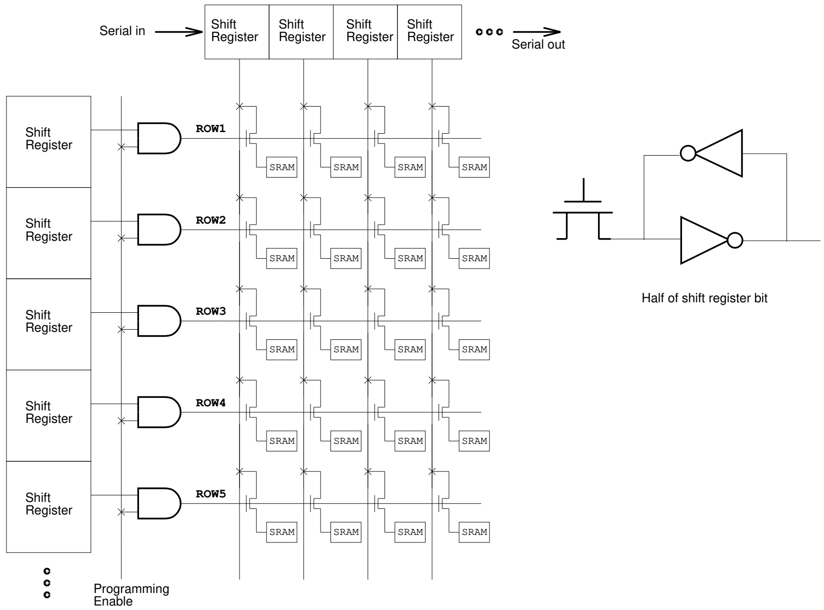

Figure 17 - *Serial-parallel programming of the configuration bits.*

Instead, we divided the tile into 16 *mini-tiles*. Each mini-tile contains one sixteenth of the L block, S block, and C blocks. The tile[1] is implemented by creating a 4x4 array of mini-tiles, and adding vias at the appropriate locations to customize the local function of each mini-tile. The flip-flops, muxes to select the latched or combinational outputs, and output tapping buffers are at the edge of the array, since there are only four of each. These are shown as part of the output stage in Figure 2. The motivation for this topology comes from the topology used for PLAs [Mead80], where the goal is to make the overall structure regular and have irregularity restricted to the final programming. Figure 2 shows the details of the mini-tile.

The major problem with dividing the functionality of the tile across 16 mini-tiles is the routing needed to interconnect the mini-tiles. It is difficult to organize the routes correctly, and it is likely that the extra wires will cost more in area than a non-mini-tile approach. The area issue is addressed further in Section 5.0. Figure 2 shows all the necessary wiring. Examples of via placement are shown by the Xs. The horizontal and vertical tracks of Figure 1 are shown in heavier lines intersecting at the switch block (S block).

The choice of how many tracks to use depends on the actual size of the overall array of tiles in a chip and on the number of mini-tiles. For larger FPGAs more tracks would be needed, and in the case of LEGO, the number will be a multiple of four because of the mini-tile array size.

Each mini-tile contains one quarter of a mux-tree, which is really a 4-to-1 mux that has the SRAM bits at its inputs and a separate 2-to-1 mux. The three mux interconnect tracks going through the quarter mux-tree are used to connect between the mini-tiles to form the full 4-input lookup tables. These tracks are segmented in the mini-tile to allow sharing of the tracks and can be connected

1. We will use *tile* to denote the complete tile, and *mini-tile* to denote the smaller building block piece.

with a piece of metal as needed during final configuration. One of the 2-to-1 muxes is not required when hooking up the full lookup table.

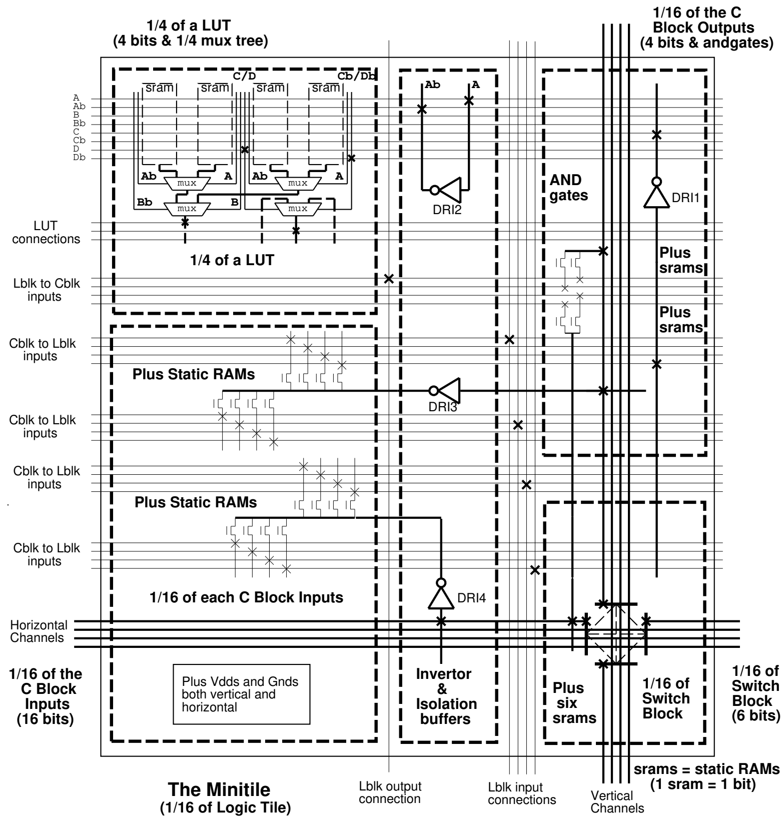

Figure 18 - *Details of the minitile.*

In addition, the outputs of the LUTs are routed on the *LUT connection* tracks to the edge of the tile for connection to the D flip-flops and to the driver to the tracks. The output of the driver is routed back into the mini-tiles along a *Lblk to C blk inputs* track for connection to the C block for output. The remaining vertical wires are to distribute signals to the mini-tiles along the vertical dimension of the array.

Driver *DRI2* is the inverter shown at the inputs to the LUTs in Figure 2. Driver *DRI1* drives the input from the C block to the LUT inputs.

The drivers marked *DRI3* and *DRI4* are the track isolation drivers as shown in Figure 2. The transistors at the outputs of these drivers are the switches described in Section 3.1. In this partitioning, the C block includes connections to the south- and east-neighbour tiles so some of the wires in each mini-tile are used to route signals to the LUT inputs in those neighbours.

Using the mini-tile methodology we sacrifice the potential for global layout optimization across the whole tile in exchange for a more simple full-custom layout task. However, we could afford to spend significant effort on the layout of the mini-tile. The floorplan of the minitile is shown in Figure 2. The vertical dimension is limited in pitch by the number of horizontal metal 2 tracks. In the horizontal dimension, there are still some visible internal vertical routing channels that do not go over any active area. If we had a third metal routing layer so that the vertical routing could go over active area, then the horizontal dimension could be reduced by about 25%. With this area removed, the layout efficiency would be quite good as most of the metal routing would be on top of active area, which cannot be compacted much. Since we did not attempt a full custom layout of the tile we do not have a quantitative measure of our layout efficiency. However, we feel that by using this methodology there was not a great sacrifice in overall density, and shortening the time to do the layout was more important.

An added benefit that has not been explored at this point is that with this ability to configure the

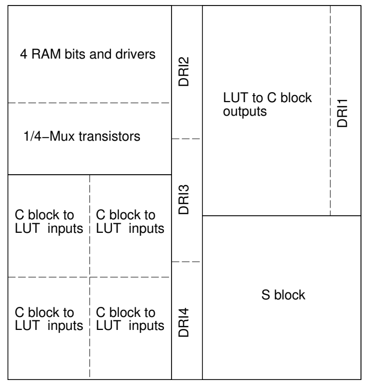

Figure 19 - *Floorplan of mini-tile.*

tile after the hard layout work is done, it is possible to try other hard-wired interconnections for the LUTs and other channel segmentation distributions. This may be of benefit if our work in non-homogeneous logic blocks suggests that a mix of tiles with different types of hard-wired connections is beneficial.

4.3 Other Layout Issues

Our experience with UTFPGA1 showed that doing a full-custom layout of a complete tile is extremely difficult, and distributing power was one of the major problems.With the mini-tile methodology, it is easy to have a rectangular floorplan for the tile, and this makes it easier to distribute power throughout the array.

As described in Section 2.2, LEGO uses a segmented routing scheme. The segments repeat at intervals of one, two, and three tiles. To achieve this with a single tile, and without requiring customization of the interconnect between the tiles, the scheme shown in Figure 2 was used. This scheme will work as long as the number of tracks for a repeat interval is the same as the repeat interval. In LEGO, there are four tracks of length two and three tracks of length three.

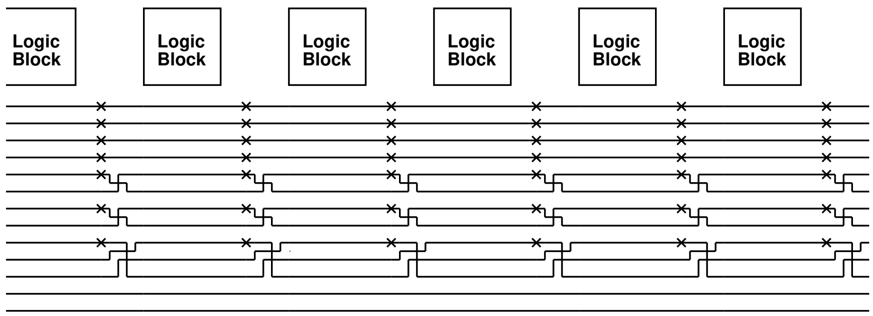

Figure 20 - *Segmentation scheme for use in a tiled layout style.*

5.0 Results and Performance

The goal of this project was to design an FPGA with high performance without greatly sacrificing area. In this section we analyze how well this goal was met. We first provide area measurements and speed measurements from simulation. Subsequently, measured results from a fabricated test chip are provided.

5.1 Area and Speed

The size of each tile, which contains four 4-input lookup tables, four D flip-flops, and connections for 16 tracks, is 1240µm by 1184µm. By way of comparison, the Xilinx 3000 tile in the same feature size technology (1.2µm CMOS), which contains the equivalent of two four-input lookup tables, two D flip-flops and connections for eight tracks (about half the functionality of the LEGO tile) is roughly 500µm by 400µm. This is about a factor of four smaller, which comes from the fact that there is twice as much logic and routing in the LEGO tile, the mini-tile strategy, and design decisions that sacrificed area for speed. In particular, the large size drivers and the switches (rather than a multiplexer) used in the C block take up significant area. Each LEGO tile requires 492 programming bits.

From simulation, we measure a delay of 4.1ns for the critical path shown in Figure 2 and Figure 2. The measured delay for LEGO is simulated time for nominal conditions, not worst-case.

An attempt to compare LEGO with commercial FPGAs is shown in Table 4. The circuit used is shown in Figure 2. The figure and some of the numbers in Table 4 are taken from some performance benchmarks published by Crosspoint Solutions [Cross92]. The numbers in the *published* column are the reported numbers assuming that the output goes to a pad driving a 50*pf* load. The *adjusted* numbers subtract 25*ns* as an attempt to take out the effect of the output pad[2]. This is a gross approximation as we do not have the numbers for the delays due to only the logic. However, it can be seen that LEGO compares well even though the others are implemented in higher performance technologies. Much of this is because of the larger functionality of the LEGO logic block. The LEGO number is from an HSPICE simulation under nominal conditions. The

2. The Crosspoint benchmarks erroneously used data for a Xilinx 4005-7 part, which does not exist. The Xilinx data has been multiplied by 5/7 to reflect the speed of a 4005-5

other numbers are for worst case process conditions.

FPGA	Published (ns)	Adjusted (ns)
Crosspoint CP20420-1	37.4	12.4
Crosspoint CP20420-0	54.2	29.2
Actel A1280-1	70.0	45.0
Xilinx XC4005-5	41.4	16.4
LEGO	n/a	11.5

Table 4 - *Performance Comparisons for the ATN Benchmark*

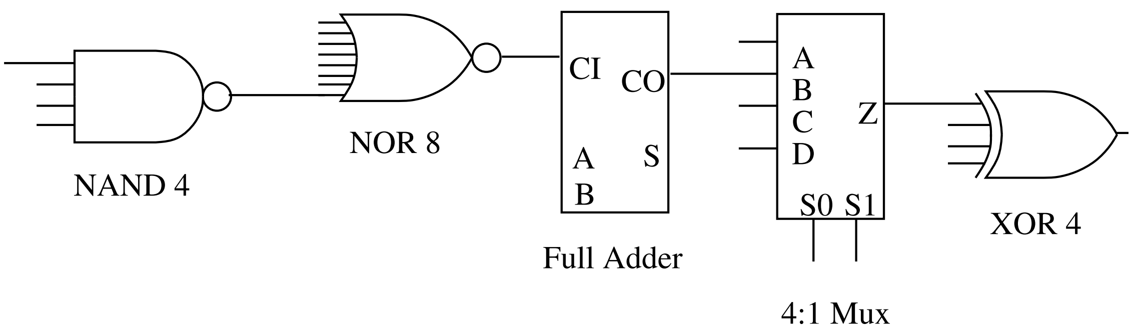

Figure 21 - *The ATN Benchmark Circuit*

5.2 Fabrication Results

A chip comprising two LEGO tiles was fabricated for which the photomicrograph is shown in Figure 2. The extra wires that exit the column and rows were either attached to pads or “wrapped” around the tiles to connect to the other side of the channel. This chip has 50 pads, including 12 power and ground, 23 signal pins into the FPGA core, eight programming pins, five control pins, and two test pins.

The testing of the chip showed that the logic blocks were not fully functional, and the cause has not been determined, though we suspect that having only nominal parameters to simulate the pass transistor logic may have lead to some threshold problems. However, we were able to make reliable connections using the programmable routing.

A smaller test chip was also fabricated. It contained only four of the mini tiles, which was enough to build one of the four-input lookup tables. From this chip, we were able to make delay measurements that included one part of the logic block. Based on these measurements, we observed a pad to pad delay through one S block switch transistor of 19 ns, where the output was unloaded. The pad to pad delay through part of the critical path shown in Figure 2 and Figure 2 was measured. The measured path exits the logic block through the tapping buffer at node 3 of

Figure 2, rather than node 6. The measured delay was 26 ns, giving a delay of about 7 ns when the pad delay is eliminated. The simulated delay for this path is about 3 ns. Speed measurements were done using a tester and have an error of ±1 ns. We attribute most of the difference to the fact that the simulations are based on nominal parameters.

Figure 22 - *Die Photo*

6.0 Conclusions

We have described the design of a high-performance FPGA from the architectural design decisions used to select the logic block architecture and the segmentation of the routing tracks, to the circuit design and layout methodology. This project has illustrated important issues in circuit design that affect the critical path delay. We have also presented a layout methodology for FPGAs that reduces the amount of custom layout required without sacrificing excessive density. The LEGO design compares favorably with existing commercial FPGAs. The most important contribution of our work is that we have gained a greater understanding of the real design issues in an FPGA circuit.

Two areas where further work is definitely merited are in clocking and the use of the D flip-flops. All of the architectural studies to date focus on the combinational logic aspects. Questions about how many flip-flops are required, and how they should be clocked need to be addressed.

7.0 Acknowledgments

This research was supported by a grant from the MICRONET network of excellence. The authors would like to thank Jaro Pristupa for his help with the chip fabrication and Paul Chow, Jr., (no relation) for help with the testing. The Canadian Microelectronic Corporation provided fabrication access using the facilities of Northern Telecom. Their help is always appreciated.

8.0 References

[Ahre90] M. Ahrens, A. El Gamal, D. Galbraith, J. Greene, S. Kaptanoglu, et. al., "An FPGA Family Optimized for High Densities and Reduced Routing Delay," Proc. 1990 Custom Integrated Circuits Conference, May 1990, pp. 31.5.1 - 31.5.4.

[Birk92] J. Birkner et al, "A very-high-speed field-programmable gate array using metal-to-metal antifuse programmable elements," Microelectronics Journal, v. 23, pp. 561-568.

[Brit93] Barry K. Britton, Dwight D. Hill, William Oswald, Nam-Sung Woo and Satwant Singh, "Optimized Reconfigurable Cell Array Architecture for High-Performance Field-Programmable Gate Arrays", Proc. 1993 Custom Integrated Circuits Conference, March 1993, pp. 7.2.1- 7.2.5.

[Brow91] S. Brown, J.S. Rose, Z. Vranesic, "A Detailed Router for Field Programmable Gate Arrays" IEEE Transactions on Computer-Aided Design of Circuits and Systems, Vol. 11, No. 5, May 1992, pp. 620-628.

[Brow92] S. Brown, R. Francis, J. Rose, Z. Vranesic, **Field-Programmable Gate Arrays**, Kluwer Academic Publishers, May 1992.

[Cart86] W. Carter, K. Duong, R. H. Freeman, H. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo and S. L. Sze, "A User Programmable Reconfigurable Gate Array," Proceedings of the 1986 Custom Integrated Circuits Conference (CICC-86), May 1986, pp. 233-235.

[Clif93] Richard Cliff et al., "A Dual Granularity and Globally Interconnected Architecture for a Programmable Logic Device", Proc. 1993 Custom Integrated Circuits Conference, March 1993, pp. 7.3.1- 7.3.5.

[Chow91] P. Chow, S.O. Seo, D. Au, B. Fallah, C. Li, J.Rose, "A 1.2um CMOS FPGA Using Cascaded Logic Blocks and Segmented Routing," International Workshop on Field Programmable Logic and Applications, Sept 1991, Oxford, UK.

[Chow93] P. Chow, S. O. Seo, K. Chung, G. Paez and J. Rose, "A High-Speed FPGA Using Programmable Mini-tiles", **Research on Integrated Systems** - *Proceedings of the 1993 Symposium on Integrated Systems*, University of Washington, March 1993, pp. 103-122.

[Chun91] K. Chung, S. Singh, J. Rose and P. Chow, "Using Hierarchical Logic Blocks to Improve the Speed of Field-Programmable Gate Arrays", in *FPGAs - Proceedings of the First International Workshop on Field Programmable Logic and Applications*, Oxford, Sept. 1991, pp. 103-113.

[Chun92] K. Chung and J. Rose, "TEMPT: Technology Mapping for the Exploration of FPGA Architectures with Hard-Wired Connections", *Proceedings of the 29th Design Automation Conference (DAC-29),* June 1992, pp. 361-367.

[Chun94] K. Chung, "Architecture and Synthesis of Field-Programmable Gate Arrays with Hard-wired Connections", *Ph. D. Thesis*, Department of Electrical and Computer Engineering, University of Toronto, 1994.

[Cross92] Crosspoint Solutions, Performance benchmarks, Rev 1.01 8/92, 1992.

[Ebel91] C. Ebeling, G. Borriello, S. Hauck D. Song and E. Walkup, “TRIPTYCH: a New FPGA Architecture,” in FPGAs, Abingdon EE&CS Books 1991 Will Moore and Wayne Luk, Ed., pp. 75-90.

[ElGa89] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat and A. Mohsen, “An Architecture for Electrically Configurable Gate Arrays,” *IEEE Journal of Solid State Circuits (JSSC),* Vol. 24, No. 2, April 1989, pp. 394-398.

[Gupt90] A. Gupta, V. Aggarwal, R. Patel, P. Chalasani, D. Chu, P. Seeni, P. Liu, J. Wu and G. Kaat, “A User Configurable Gate Array Using CMOS-EPROM Technology”, *CICC-90,* May 1990, pp. 31.7.1-31.7.4.

[Hauc92] S. Hauck, G. Borriello, S. Burns and C. Ebeling, “MONTAGE: An FPGA for Synchronous and Asynchronous Circuits,” Proceedings of the Second International Workshop on Field-Programmable Logic and Applications, 1992, Vienna.

[Henn90] J. Hennessy and D. Patterson, **Computer Architecture: A Quantitative Approach**, Morgan Kaufmann, 1990.

[Hsie88] H. Hsieh, K. Duong, J. Ja, R. Kanazawa, L. Ngo, L. Tinkey, W. Carter, and R. Freeman, “A Second Generation User-Programmable Gate Array,” Proc. 1987 Custom Integrated Circuits Conference, May 1987, pp. 515 - 521.

[Hsie90] H. Hsieh, W. Carter, J. Y. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin and L. Tinkey, “Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays,” *CICC-90,* pp. 31.2.1-31.2.7.

[Hspic90] Meta-Software, Inc., 1300 White Oaks Road, Campbell, CA 95008, HSPICE User's Manual.

[Koul92] J. Kouloheris and A. El Gamal, “FPGA Area vs. Cell Granularity - PLA Cells,” in Proc. of Custom Integrated Circuits Conference, May 1992.

[Marp92] David Marple and Larry Cooke, “An MPGA Compatible FPGA Architecture,” In Custom Integrated Circuits Conference, IEEE, 1992, pp. 4.2.1-4.2.4

[Mead80] C. Mead and L. Conway, **Introduction to VLSI Systems**, Addison-Wesley, 1980.

[Paez95] G. Paez, I. Rahardja, and J. Rose, “On the Design of Area-Efficient Segmented Routing Architectures for Field-Programmable Gate Arrays,” in preparation, 1995.

[Pate95] R. Patel, et. al, “A 90.7MHz - 2.5 Million Transistors CMOS PLD with JTAG Boundary Scan and In-System Programmability,” CICC-95, May 1995, pp 507-510.

[Rose89] J.S. Rose, R.J. Francis, P. Chow, and D. Lewis, “The Effect of Logic Block Complexity on

Area of Programmable Gate Arrays," Proc. IEEE Custom Integrated Circuits Conference, (CICC), San Diego, May 1989, pp. 5.3.1 - 5.3.5.

[Rose90] J. Rose, R. J. Francis, D. Lewis, P. Chow, "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency", *IEEE JSSC*, Vol. 25, No. 5, Oct. 1990, pp. 1217-1225.

[Sing91] S. Singh, "The Effect of Logic Block Architecture on FPGA Performance", *M.A.Sc. Thesis*, Department of Electrical and Computer Engineering, University of Toronto, 1991.

[Sing92] S. Singh, J. Rose, P. Chow and D. Lewis, "The Effect of Logic Block Architecture on FPGA Performance," *JSSC,* Vol. 27, No. 3, March 1992, pp. 281-287.

[Smit93] D.E. Smith, "Intel's FLEXlogic FPGA Architecture," Compcon Spring '93, pp. 378-384.

[Suth91] Ivan E. Sutherland and Robert F. Sproull, "Logical Effort: Designing for Speed on the Back of an Envelope," Proc. 1991 UC Santa Cruz Conference on Advanced Research in VLSI, March 1991, pp. 1-16.

[Tava95] D. Tavana, W. Yee, S. Young, and B. Fawcett, "Logic Block and Routing Considerations for a New SRAM-Based FPGA Architecture," CICC '95, May 1995, pp. 511-514.

[Wong89] S.C. Wong, H.C. So, J.H. Ou and J. Costello, "A 5000-Gate CMOS EPLD with Multiple Logic and Interconnect Arrays," Proc. 1989 Custom Integrated Circuits Conference, May 1989, pp. 5.8.1 - 5.8.4.

[Yang91] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide," Version 3.0, Microelectronics Centre of North Carolina (MCNC), January 1991.