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

Paul Chow, Soon Ong Seo, Jonathan Rose, Kevin Chung, Gerard Páez-Monzón and Immanuel Rahardja

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 experimentation 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 using full-custom hand layout to obtain absolute minimum die sizes. This is both labor and time-intensive. We propose a design style with a *mini-tile* that contains a portion of all the components in the logic tile, resulting in less full-custom effort. The mini-tile is replicated in a 4×4 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.

Keywords— **field-programmable gate arrays, FPGA, FPGA architecture, SRAM programmable**

I. Introduction

FIELD -Programmable Gate Array (FPGA) technology permits the design of many different complex digital circuits using a single off-the-shelf device [1]. 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.

Paul Chow and Jonathan Rose are with the Department of Electrical and Computer Engineering at the University of Toronto. Soon Seo is now with ATI Technologies. Kevin Chung is now with Xilinx, Inc. Gerard Páez-Monzón is with CEMISID-Universidad de Los Andes-Venezuela and Immanuel Rahardja is now with Aristo Technology Inc.

Paul Chow can be reached at pc@eecg.toronto.edu

FPGAs were first introduced in 1986 by Xilinx using a memory-based programming technology [2]. Since then there have been many new commercial architectures [3], [4], [5], [6], [7] and several new programming technologies including two types of anti-fuse [8], [9], [10] and floating gate transistors, which are UV and electrically erasable [11], [12], [13].

A few non-commercial FPGA architectures have been reported for which the design details are more readily available. The Triptych FPGA [14], [15] 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 [16], [15] 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 [17]. 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 [18].

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 [19]. 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 per-

form *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 III discusses the circuit design issues and the layout of the chip is presented in Section IV. The performance of the chip and the design summary are provided in Section V.

II. Architecture

The speed and density gap between FPGAs and mask-programmable gate arrays (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 [3] or anti-fuses [20], or floating-gate based switching [13]), 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.

A. 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 [21]. 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 block 5 in Figure 1(a) 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.

A.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 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 [22], 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 [23], [24], [25], [26], [27] and as such has been adopted by several commercial vendors [3], [4], [5].

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.

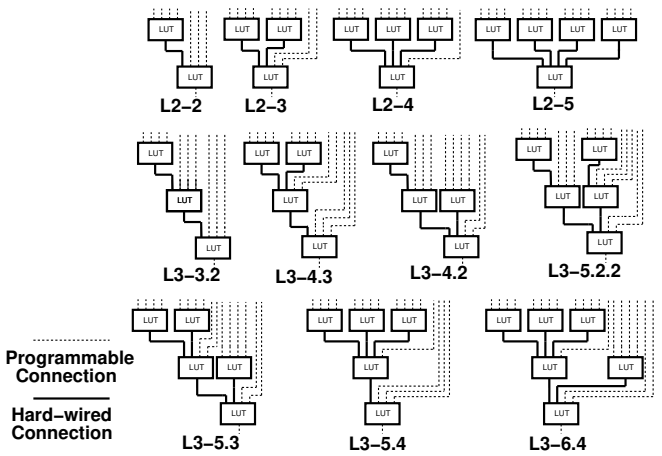
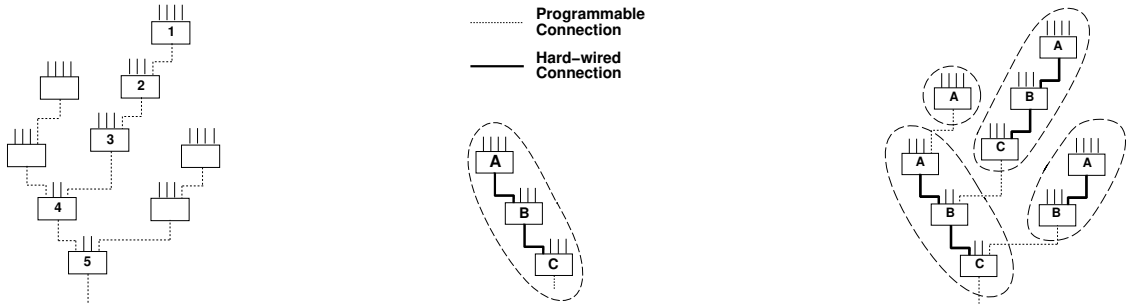


Fig. 2. Example of some of the tree topologies investigated.

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



(a) Network of Basic Blocks (b) An HLB (c) Faster HLB Network

Fig. 1. Hard-wired connections and logic blocks.

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.

A.2 Experimental Method

To evaluate the HLB topologies, a set of 15 benchmark combinational circuits from the MCNC suite [28] 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 [29].

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 [22]. 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.

A.3 Experimental Results

Table I provides the summarized results of the implementation of the benchmark circuits for those HLB topologies that appear the most promising. Table I 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 I 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 shows a slower 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 I give the normalized area with respect to L1, for both the speed and area cases. The un-normalized absolute values for speed and area are given in the L1 row, assuming a $1.2\mu m$ 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.

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 than 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 I, we reject those blocks that have too high an area penalty: those in the last four rows. Of the remaining blocks, L3-5.2, L2-5, L3-4.2, and L2-4 provide the best

TABLE I
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
1	L1	1 (35 MHz)	1 (15.3 × 106 μm^2)	1 (22 MHz)	1 (6.7 × 106 μm^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
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

speed up. However, the L3-4.2 block proved superior to the others in the area-optimized case, 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.

It is tempting to use programmable multiplexers in place of the hard-wired connections to allow either the direct link or to access an input. We do not consider their use in this study, for two reasons: First, we are interested in finding the fastest topologies, and the multiplexers do not improve the speed; Secondly, each extra input to the logic block is expensive in terms of area, and part of the purpose of the hard-wired connection is to save that area.

B. 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 [2], [30] 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 3. 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 [8].

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.

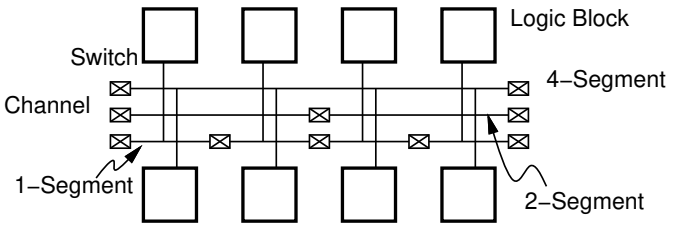


Fig. 3. Segmented Routing Architecture

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 unpopulated 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 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 [31]. 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.

B.1 Definitions and Experimental Method

Figure 4 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.

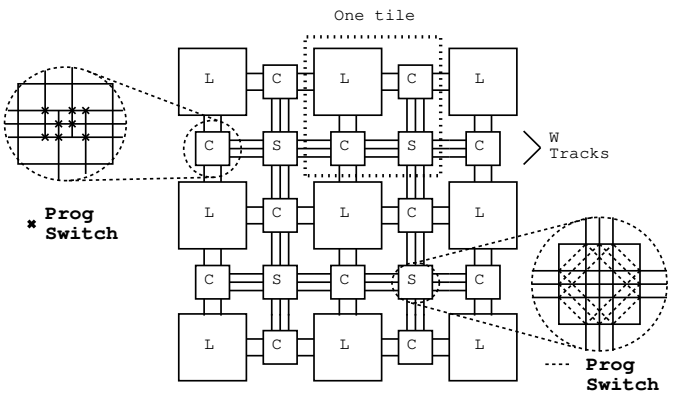


Fig. 4. Architectural Definitions

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 unpopulated, 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 depopulated, and for 3-segments, there are two switch blocks and one connection block that can be de-populated.

B.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.

B.2.a Population of Segments. The first question we address is that of population of the S and C blocks. Figure 5 is a plot, for $D_2 = 0$, of the average number of excess tracks versus D1 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.

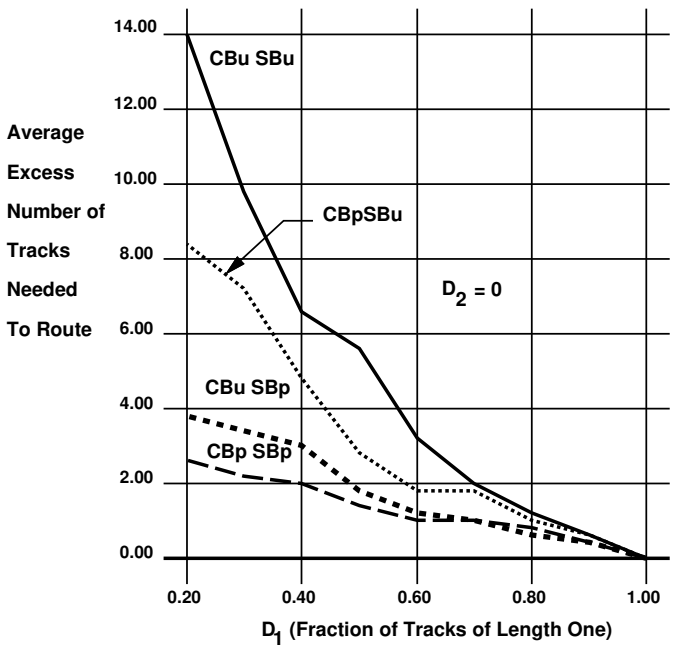


Fig. 5. Population Comparison

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.

B.2.b Segmentation Distribution. Table II 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 II 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 D2 in the range of 0.0 to 0.6 and D3 in the range of 0.1 to 0.4 (those shown in boxes in Table II) would reflect such a choice. The faster architectures are those with higher values of D_3.

In Section IV-C 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.

C. 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 4 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.

III. 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 $1.2\mu m$ two-level metal n-well CMOS process.

Figure 6 illustrates the signal path starting at the input of an L3-4.2 lookup table, which was selected in Section II-A. 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.

In the following sections we discuss issues that arose in the design of this path. Simulation with HSPICE [32] was used to evaluate different circuit options.

A. 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 7 illustrates the use of a multiplexer and Figure 8 shows a connection block using one switch for each track-to-pin connection. The boxes labelled SRAM are the programming bits. In UTFPGA1 [17] we used multiplexers for the inputs and switches for the outputs. The reasoning was that since only one wire in the channel can be

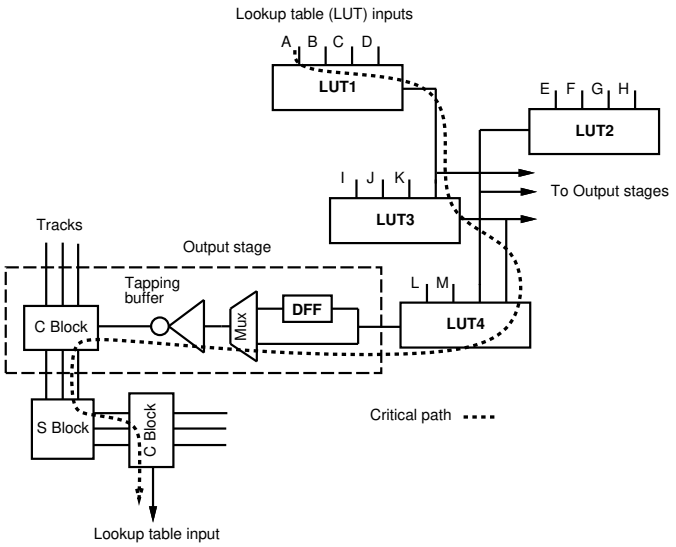


Fig. 6. Architectural Definitions

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.

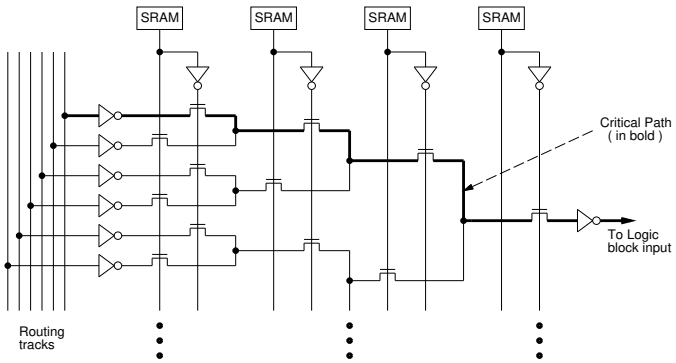


Fig. 7. Connection block input using a multiplexer.

Table III 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 $\times$ 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 IV-A. Unit-sized transistors are used in the programming cells and the transistors that would be used in

TABLE II

AVERAGE EXCESS TRACKS FOR ALL DISTRIBUTIONS, POPULATION CB_uSB_p. ABSOLUTE AVERAGE NUMBER OF TRACKS = 12, STANDARD DEVIATION = 2.2.

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								

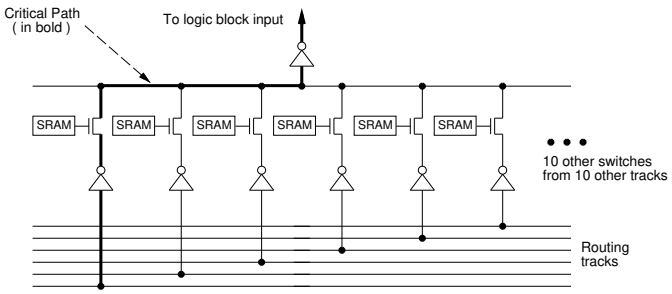


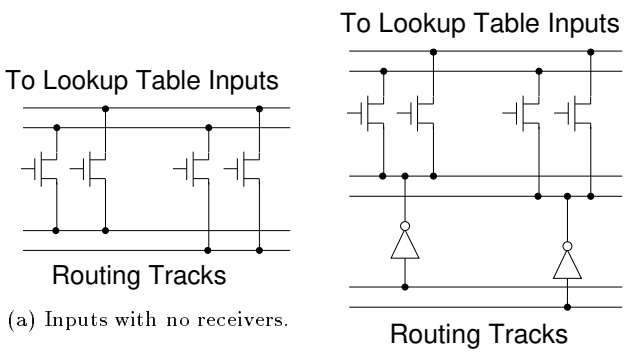
Fig. 8. Connection block input using pass transistors.

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.

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.

B. Isolation

Another important issue is the capacitance at the C block inputs. With switches connected as shown in Figure 9(a), the capacitance seen from the routing track depends on whether the switch is on or off. If the 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 [14] 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 9(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 10, which reduces the number of buffers required by one half, also reducing the capacitance on the track.



(a) Inputs with no receivers.

(b) Inputs with isolation.

Fig. 9. C-block input isolation options.

C. Lookup Table Design

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

Figure 11 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 6 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 11), while the

TABLE III
AREA COMPARISON FOR THE TWO POSSIBLE C BLOCK TECHNIQUES.

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

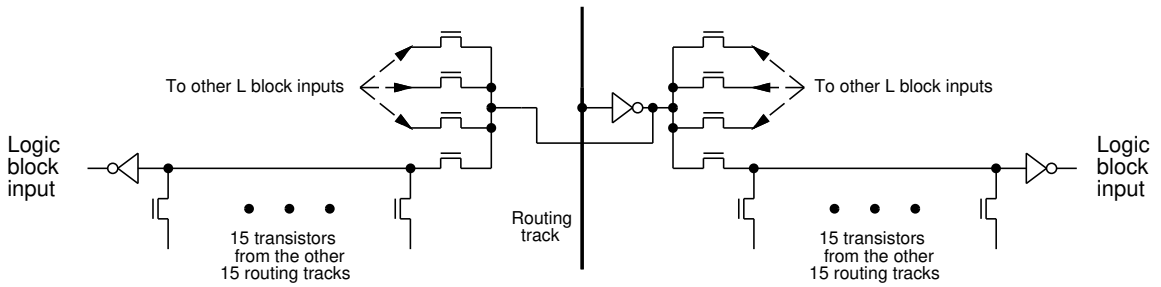


Fig. 10. Track isolation using shared buffer.

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.

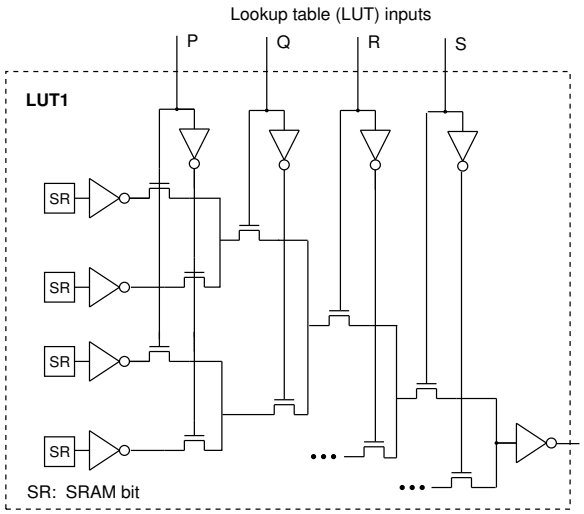


Fig. 11. Top quarter of LUT multiplexer tree.

D. Full Complementary Versus n-channel Only Switches

Throughout the array there are switches in the LUT, the C block, and the S block, so the design of the switches will have a significant impact on the area and speed of the array. The most obvious area gain can be made by using a single n-channel pass transistor instead of a full transmission gate. In this section, we summarize the important considerations when deciding between the two alternatives.

By using a pass transistor, the area benefit is not just because there are half the transistors, but also because of a savings in well areas, which can be significant depending on whether the switches are scattered enough that individual wells must be provided for each transmission gate. As well, internal nodes of n-channel pass transistor circuits may not require contacts, saving area and reducing junction capacitance. There is also a savings in routing for the complementary clock or providing the local clock inverter if that technique is used.

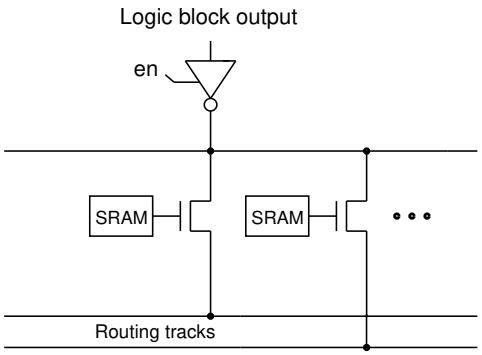
For signal integrity, the pass transistor causes a V_t voltage drop when passing highs and has a slower rise time towards the end of a low to high transition. These effects can be partially compensated by lowering the switching threshold of the succeeding gates. Unfortunately, the reduced high also means that the pullup transistors in succeeding gates will not be fully turned off, resulting in static power dissipation. The problems become even greater as supply voltages move towards $3V$ because the worst case V_t voltage drop could still be around $1.2V$ to $1.3V$ ($V_t = 0.7V$ nominal at room temperature) significantly degrading the noise margin. This means that pass transistors are not viable unless the gate voltage of the pass transistor is bootstrapped to above the supply.

As for speed, consider the construction of a transmission gate by the addition of a p-channel transistor that is 2.5 times the size of the n-channel transistor. The resistance changes by 0.5 but the junction capacitance increases by 3.5 or more, depending on the use of contacts. This means that the RC delay is at least $0.5 \times 3.5 = 1.75$ times larger so that the n-channel pass transistor would normally be faster, although careful simulations would always be required to make sure that this is true.

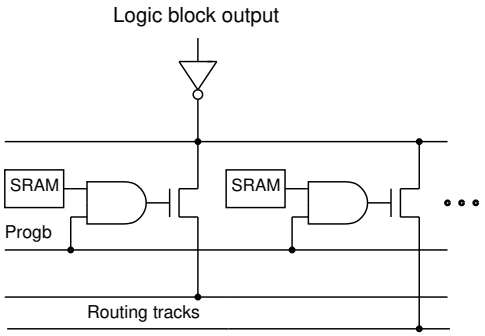
In LEGO, which was designed for $5V$ operation, we chose to use pass transistors because we could get a better propagation delay and reduced area. To compensate for the V_t drop we used simulations to adjust the switching point of the gates following the switches and to make sure that the propagation delay for rising and falling signals was balanced.

E. 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 [17] the output drivers are disabled using a tristate buffer as shown in Figure 12(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 12(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.



(a) Using a tristate buffer.



(b) Using a buffer and the **progb** signal.

Fig. 12. Hazard prevention alternatives.

F. 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 [14], the method of Logical Effort [33] 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 13, 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 $37mm$ for the size of the n-channel driver pulldown. In LEGO we actually used $34mm$ because in our initial simulations we used nine tracks in the channels, which was later changed to 16 tracks, and thus a higher load.

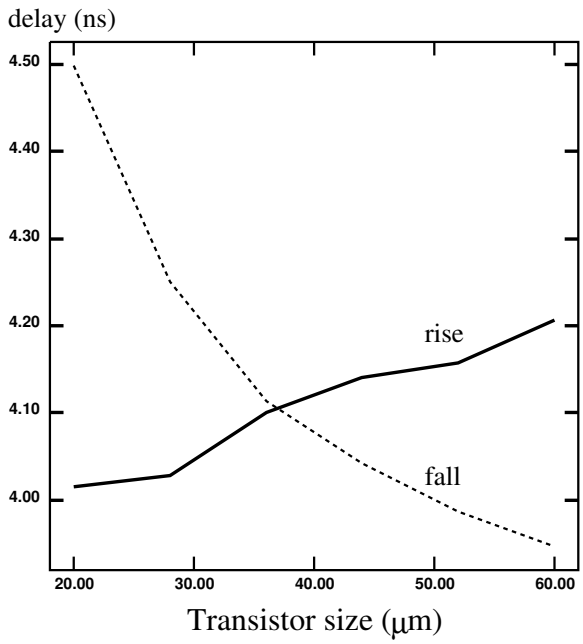


Fig. 13. An example of varying switch sizes for transistor sizing.

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 14 shows the critical path through the logic block section, and Figure 15 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.

IV. 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.

A. Programming Architecture

Three techniques for programming the configuration bits were identified. The simplest is to connect all of the bits

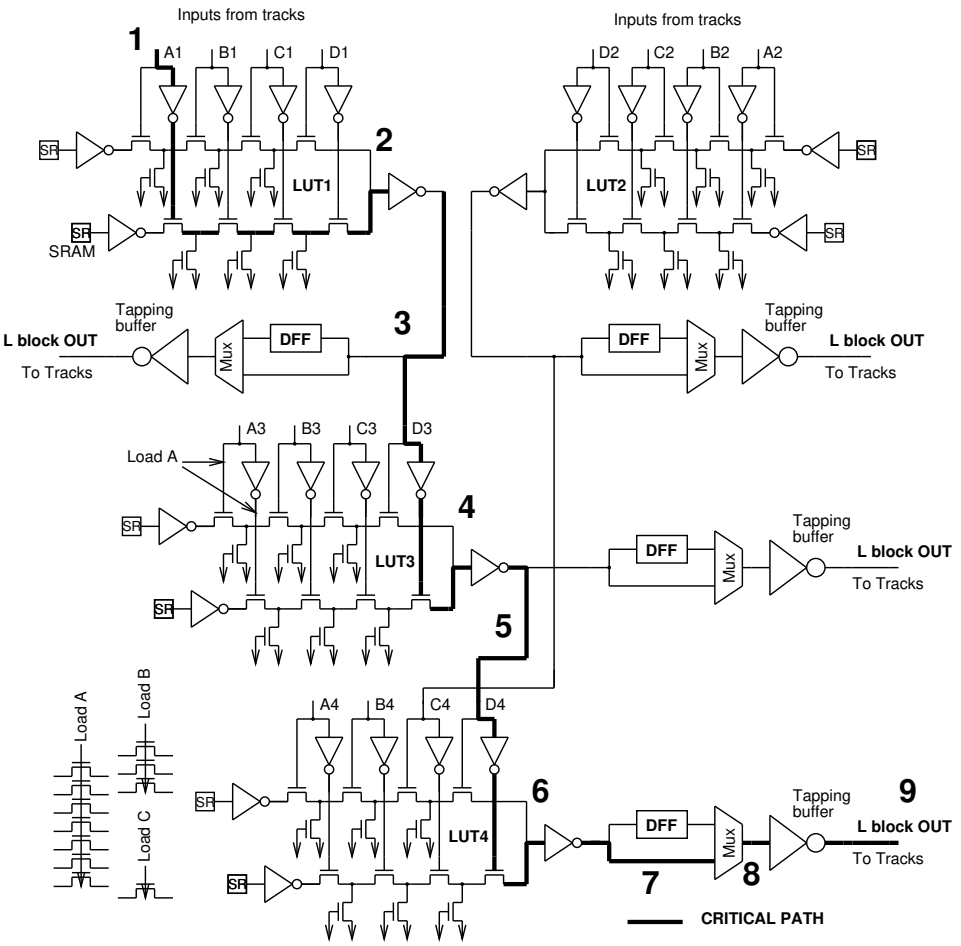


Fig. 14. Details of critical path through the logic block.

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.

A compromise of the serial and parallel techniques was used in LEGO and is shown in Figure 16. 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.

B. 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 4, 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 finer granularity.

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 4×4 array of mini-tiles, and adding vias at the appropriate locations

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

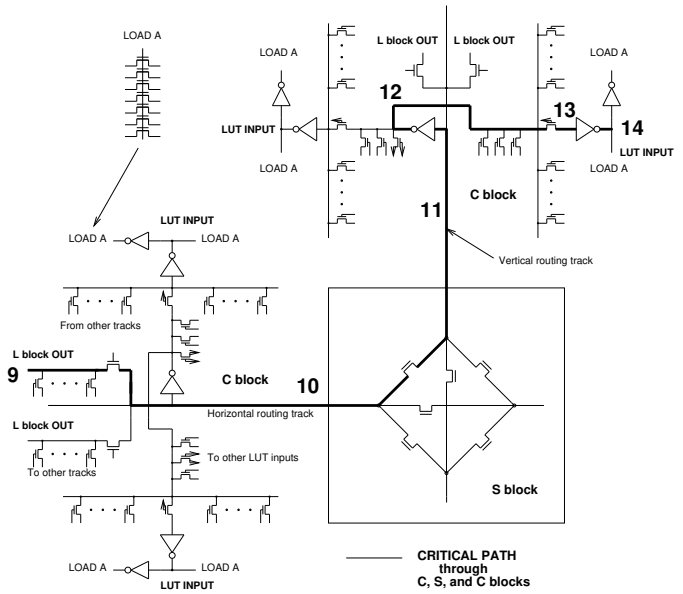


Fig. 15. Details of critical path through the routing network.

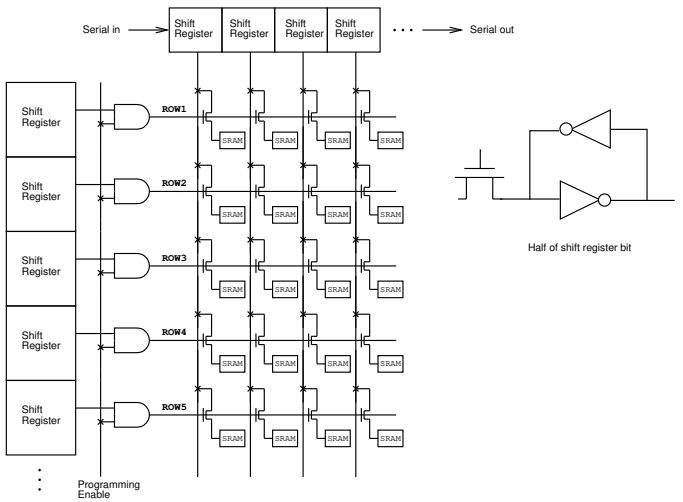


Fig. 16. Serial-parallel programming of the configuration bits.

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 6. The motivation for this topology comes from the topology used for PLAs [34], where the goal is to make the overall structure regular and have irregularity restricted to the final programming. Figure 17 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 V. Figure 17 shows all the necessary wiring. Examples of via placement are shown by the Xs. The horizontal and vertical tracks of Figure 4 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 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.

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 11. 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 10. The transistors at the outputs of these drivers are the switches described in Section III-A. 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 mini-tile is shown in Figure 18. 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 tile after the hard layout work is done, it is possible to try other hard-wired interconnections for the LUTs and other channel segmen-

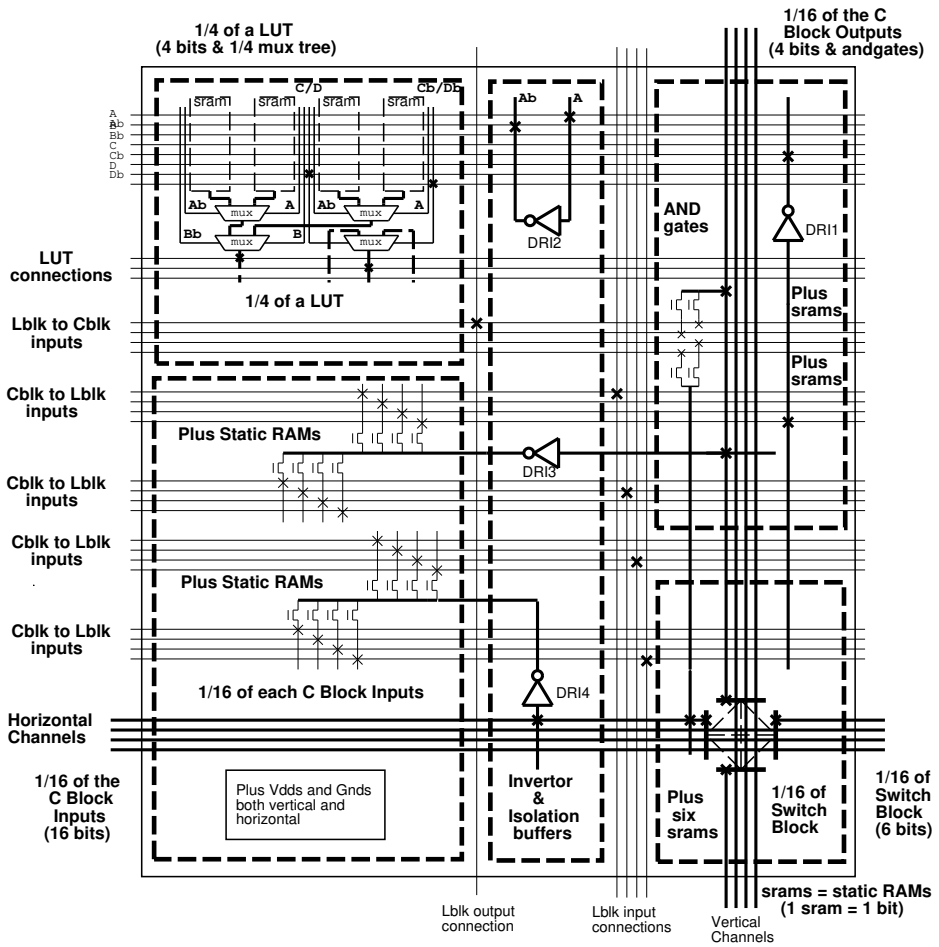


Fig. 17. Details of the mini-tile (1/16 of a logic tile).

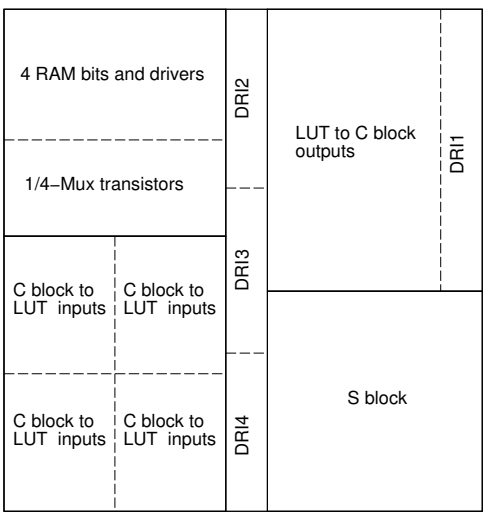


Fig. 18. Floorplan of mini-tile.

tation 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.

C. 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 II-B 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 19 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.

V. 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.

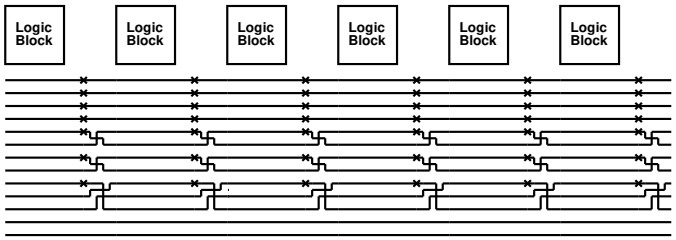


Fig. 19. Segmentation scheme for use in a tile layout style.

A. 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 $1240\mu m$ by $1184\mu m$. By way of comparison, the Xilinx 3000 tile in the same feature size technology ($1.2\mu m$ 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 $500\mu m$ by $400\mu m$. 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 14 and Figure 15. 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 IV. It is only provided as a point of reference since doing more complicated circuits across a number of architectures would be difficult. The circuit used is shown in Figure 20. The figure and some of the numbers in Table IV are taken from some performance benchmarks published by Crosspoint Solutions [35]. The numbers in the *published* column are the reported numbers assuming that the output goes to a pad driving a $50pf$ load. The *adjusted* numbers subtract $25ns$ 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 other numbers are for worst case process conditions.

B. Fabrication Results

A chip comprising two LEGO tiles was fabricated for which the photomicrograph is shown in Figure 21. 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.

[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.

TABLE IV
PERFORMANCE COMPARISONS FOR THE ATN BENCHMARK.

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

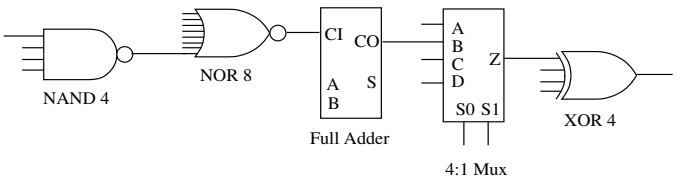


Fig. 20. The ATN benchmark circuit.

Fig. 21. Photomicrograph of the LEGO test chip.

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 $19ns$, where the output was unloaded. The pad to pad

delay through part of the critical path shown in Figure 14 and Figure 15 was measured. The measured path exits the logic block through the tapping buffer at node 3 of Figure 15, rather than node 6. The measured delay was $26ns$, giving a delay of about $7ns$ when the pad delay is eliminated. The simulated delay for this path is about $3ns$. Speed measurements were done using a tester and have an error of $\pm 1ns$. We attribute most of the difference to the fact that the simulations are based on nominal parameters.

VI. Conclusions

We have described the design of a high-performance FPGA from the high-level architectural decisions used to select the logic block and routing architecture to the low-level circuit design and layout methodology.

For the architectural design, we developed the architecture using an experimental process in which custom-built CAD tools are used to implement benchmark circuits on candidate architectures. This method has the danger that the tools may have unfair algorithmic biases towards certain architectures, resulting in "incorrect" architectural decisions. We have made an honest effort, however, to use the highest quality algorithms possible. This method is the only practical way of dealing with the enormous complexity of FPGA architecture development because theoretical analysis can never account for all of the conflicting constraints that must be considered in a real design. In addition, this kind of approach produces an architecture that is tuned to the capabilities of the tools, which is key to the success of an FPGA.

This project has also illustrated the important issues in circuit design that affect the critical path delay. We have focused on achieving a high-speed design while keeping in mind the area tradeoffs. The most critical issues are the design of the switches and minimizing the capacitance of the routing network. As well as the circuit issues, we have also presented a layout methodology for FPGAs that reduces the amount of custom layout required without sacrificing excessive density.

Our results have shown that 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.

As technology continues to scale, the area of the logic will decrease but the relative effect of the delays due to wiring will increase. This will create further demands on the routing architecture, especially as the amount of logic available on a single FPGA will also increase resulting in larger systems being built. Architecturally, there will continue to be a need for innovation in new logic and routing structures, and any new studies must always match the capabilities of the design and implementation tools to the architecture. It is especially important that a good set of benchmark circuits be available, especially to the academic community. With the trend towards larger systems-on-a-chip, studies like ours can only have merit when they use circuits that reflect this trend. We hope that the community will work towards improving the publicly available benchmark suites such as the one provided by MCNC [28]. It is especially important that the industrial community participate in this effort as they possess such circuits and stand to gain the most from the research that uses them.

In the future, we will explore other aspects in architecture and design, including the incorporation of large blocks of memory, more complex routing structures for high-speed interconnect, and the global distribution of routing tracks.

VII. Acknowledgments

This research was supported by a grant from the MICRONET Network of Excellence. The authors would like to thank Jaro Pristupa for his support of our CAD and VLSI labs, Paul Chow, (no relation) for help with the testing, and Ken Martin for his insights into circuit design using pass transistors. The Canadian Microelectronic Corporation provided fabrication access using the facilities of Nortel. Their help is always appreciated.

We also acknowledge the valuable feedback provided by the referees.

References

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

[2] 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," in *Proceedings of the 1986 Custom Integrated Circuits Conference (CICC-86)*, May 1986, pp. 233–235.

[3] 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," in *Proceedings of the 1990 Custom Integrated Circuits Conference (CICC-90)*, 1990, pp. 31.2.1–31.2.7.

[4] 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," in *Proceedings of the 1993 Custom Integrated Circuits Conference*, 1993, pp. 7.2.1– 7.2.5.

[5] Richard Cliff et al., "A dual granularity and globally interconnected architecture for a programmable logic device," in *Proceedings of the 1993 Custom Integrated Circuits Conference*, 1993, pp. 7.3.1– 7.3.5.

[6] D.E. Smith, "Intel's FLEXlogic FPGA architecture," in *Compcon Spring '93*, 1993, pp. 378–384.

[7] D. Tavana, W. Yee, S. Young, and B. Fawcett, "Logic block and routing considerations for a new SRAM-based FPGA architecture," in *Proceedings of the 1995 Custom Integrated Circuits Conference*, 1995, pp. 511–514.

[8] Abbas El Gamal, Jonathan Greene, Justin Reyneri, Eric Rogoyski, Khaled A. El-ayat, and Amr Mohsen, "An architecture for electrically configurable gate arrays," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 394–398, Apr. 1989.

[9] J. Birkner et al., "A very-high-speed field-programmable gate array using metal-to-metal antifuse programmable elements," *Microelectronics Journal*, vol. 23, pp. 561–568, 1992.

[10] David Marple and Larry Cooke, "An MPGA compatible FPGA architecture," in *Proceedings of the 1992 Custom Integrated Circuits Conference*. IEEE, 1992, pp. 4.2.1–4.2.4.

[11] S.C. Wong, H.C. So, J.H. Ou, and J. Costello, "A 5000-gate CMOS EPLD with multiple logic and interconnect arrays," in *Proceedings of the 1989 Custom Integrated Circuits Conference*, 1989, pp. 5.8.1 – 5.8.4.

[12] 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," in *Proceedings of the 1990 Custom Integrated Circuits Conference (CICC-90)*, 1990, pp. 31.7.1–31.7.4.

[13] R. Patel et al., "A 90.7MHz - 2.5 million transistors CMOS PLD with JTAG boundary scan and in-system programmability," in *Proceedings of the 1995 Custom Integrated Circuits Conference*, 1995, pp. 507–510.

[14] Carl Ebeling, Gaetano Borriello, Scott A. Hauck, David Song, and Elizabeth A. Walkup, "TRIPTYCH: a new FPGA architecture," in *FPGAs*, Will Moore and Wayne Luk, Eds., chapter 3.1, pp. 75–90. Abingdon EE&CS Books, 15 Harcourt Way, Abingdon OX14 1NV, England, 1991, Presented at the Oxford 1991 International Workshop on Field Programmable Logic and Applications.

[15] G. Borriello, C. Ebeling, S. A. Hauck, and S. Burns, "The triptych FPGA architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 4, pp. 491–500, Dec. 1995.

[16] Scott Hauck, Gaetano Borriello, Steven Burns, and Carl Ebeling, "MONTAGE: An FPGA for synchronous and asynchronous circuits," in *Proceedings of the Second International Workshop on Field-Programmable Logic and Applications*, Vienna, Austria, Sept. 1992.

[17] Paul Chow, Soon Ong Seo, Dennis Au, Terrence Choy, Bahram Fallah, David Lewis, Cherry Li, and Jonathan Rose, "A 1.2μm CMOS FPGA using cascaded logic blocks and segmented routing," in *FPGAs*, Will Moore and Wayne Luk, Eds., chapter 3.2, pp. 91–102. Abingdon EE&CS Books, 15 Harcourt Way, Abingdon OX14 1NV, England, 1991, Presented at the Oxford 1991 International Workshop on Field Programmable Logic and Applications.

[18] Paul Chow, Soon Ong Seo, Kevin Chung, Gerard Paez, and Jonathan Rose, "A high-speed FPGA using programmable minitiles," in *Symposium on Integrated Systems, previously the Conference on Advanced Research in VLSI*, Mar. 1993, pp. 103–122.

[19] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach, Second Edition*, Morgan Kaufmann Publishers, Inc., 1995.

[20] M. Ahrens, A. El Gamal, D. Galbraith, J. Greene, S. Kaptanoglu, et al., "An FPGA family optimized for high densities and reduced routing delay," in *Proceedings of the 1990 Custom Integrated Circuits Conference (CICC-90)*, 1990, pp. 31.5.1 – 31.5.4.

[21] Kevin Chung, Satwant Singh, Jonathan Rose, and Paul Chow, "Using hierarchical logic blocks to improve the speed of field-programmable gate arrays," in *FPGAs*, Will Moore and Wayne Luk, Eds., chapter 3.3, pp. 103–113. Abingdon EE&CS Books, 15 Harcourt Way, Abingdon OX14 1NV, England, 1991, Presented at the Oxford 1991 International Workshop on Field Programmable Logic and Applications.

[22] 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.

[23] Jonathan Rose, Robert J. Francis, Paul Chow, and David Lewis, "The effect of logic block complexity on area of programmable gate arrays," in *Custom Integrated Circuits Conference*. IEEE, May 1989, pp. 5.3.1–5.3.5.

[24] Jonathan Rose, Robert J. Francis, David Lewis, and Paul Chow, "Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1217–1225, Oct. 1990.

[25] Satwant Singh, "The effect of logic block architecture on the speed of field-programmable gate arrays," M.A.Sc. thesis, University of Toronto, Department of Electrical Engineering, Toronto, Ontario, M5S 3G4, 1991.

[26] Satwant Singh, Jonathan Rose, Paul Chow, and David Lewis, "The effect of logic block architecture on FPGA performance," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 3, pp. 281–287, Mar. 1992.

[27] Jack L. Kouloheris and Abbas El Gamal, "PLA-based FPGA area versus cell granularity," in *Proceedings of the 1992 Custom Integrated Circuits Conference*. IEEE, 1992, pp. 4.3.1–4.3.4.

[28] Microelectronics Centre of North Carolina (MCNC), *Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0*, Jan. 1991.

[29] K. Chung and J. Rose, "TEMPT: Technology mapping for the exploration of FPGA architectures with hard-wired connections," in *Proceedings of the 29th Design Automation Conference (DAC-29)*, 1992, pp. 361–367.

[30] Hung-Cheng Hsieh, Khue Duong, Jason Y. Ja, Roy Kanazawa, Luan T. Ngo, Liane G. Tinkey, William S. Carter, and Ross H. Freeman, "A second generation user-programmable gate array," in *Custom Integrated Circuits Conference*. IEEE, 1987, pp. 515–521.

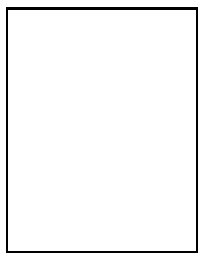
[31] S. Brown, J.S. Rose, and Z. Vranesic, "A detailed router for field programmable gate arrays," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, vol. 11, no. 5, pp. 620–628, May 1992.

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

[33] Ivan E. Sutherland and Robert F. Sproull, "Logical effort: Designing for speed on the back of an envelope," in *Proceedings of the 1991 University of California at Santa Cruz Conference on Advanced Research in VLSI*, Mar. 1991, pp. 1–16.

[34] Carver Mead and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

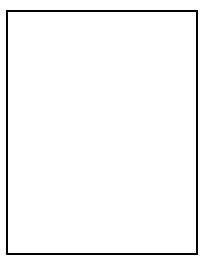
[35] "Crosspoint solutions, performance benchmarks, rev 1.01," Aug. 1992.



Paul Chow (S'79–M'83) received the B.A.Sc. degree with honours in Engineering Science, and the M.A.Sc. and Ph.D. degrees in Electrical Engineering from the University of Toronto, Toronto, Ont., in 1977, 1979 and 1984, respectively.

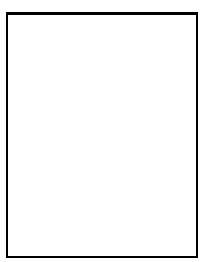
In 1984 he joined the Computer Systems Laboratory at Stanford University, Stanford, CA, as a Research Associate, where he was a major contributor to the MIPS-X project. In January 1988, he joined the Department of Electrical and Computer Engineering at the University of Toronto where he is now an Associate Professor. He spent the 1995-1996 year at ATI Technologies Inc., in Thornhill, Ont., doing IC Design and working on new CAD flows. His research interests include high performance computer architectures, architectures for programmable digital signal processors, VLSI systems design, and field-programmable gate array architectures and applications. More details about his research can be found at www.eecg.toronto.edu/~pc.

He is currently the Chair of the Technical Advisory Committee for the Canadian Microelectronics Corporation.



Soon Ong Seo received the B.A.Sc. degree with honours in Electrical Engineering from the University of Ottawa, Ottawa, Ont. in 1985, and the M.A.Sc. degree from the University of Toronto, Toronto, Ont., in 1994.

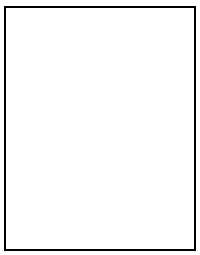
In 1992 he joined ATI Technologies Inc., Thornhill, Ont., as a Design Engineer, where he is involved in the design and support of many graphics chips. His interests include high performance architecture and design, graphics, and field-programmable gate array design and architecture.



Jonathan Rose is an Associate Professor of Electrical and Computer Engineering at the University of Toronto, and an NSERC University Research Fellow.

He received the Ph.D. degree in Electrical Engineering in 1986 from the University of Toronto. From 1986 to 1989, he was a Research Associate in the Computer Systems Laboratory at Stanford University. In 1989, he joined the faculty of the University of Toronto. He spent the 1995-1996 year as a Senior Research Scientist at Xilinx, Inc., in San Jose, CA, working on a next-generation FPGA architecture.

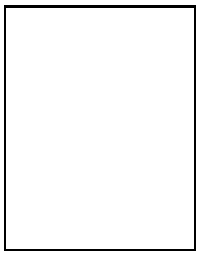
He is the co-founder of the ACM FPGA Symposium, and remains part of that Symposium on its steering and program committees. He has worked for Bell-Northern Research and a number of FPGA companies on a consulting basis. A paper co-authored with Steve Brown won a distinguished paper award at the 1990 ICCAD Conference. His research covers all aspects of FPGAs including architecture, CAD, Field-Programmable Systems, and graphics and vision applications of rapid prototyping systems.



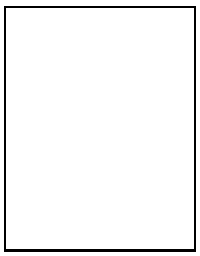
Kevin Chung received the B.A.Sc., M.A.Sc., and Ph.D. degree at the Department of Electrical and Computer Engineering of the University of Toronto in 1986, 1988 and 1994 respectively.

He is currently working as a Senior Software Engineer at Xilinx, Inc. at the Xilinx Toronto Development Centre in Toronto and has been with Xilinx since 1994. Previously, he was a Software Engineer at Data I/O Corporation from 1993 to 1994.

His main interests are in Field-Programmable Gate Array (FPGA) architectures and CAD algorithms for FPGAs.



Gerard Páez-Monzón (BEE,Villanova University, PA-USA '79; DEA in Computer Systems, Université Pierre et Marie Curie, Paris-France '83; Ph.D. in Computer Science, UPMC, Paris-France '86). He is a Professor with the Engineering School of the Universidad de Los Andes-Venezuela. He was a visiting scholar at the University of Toronto, Canada '91-92 where he worked with the FPGA group. Páez-Monzón is a principal member of the Research Center CEMISID of the Universidad de Los Andes in the areas of Microprocessor Architecture, Microprogramming, Floating-Point Units Design, FPGA, and VLSI Design. He is the author of a Computer Architecture book and a number of Computer Science research papers. He is currently on leave at National Semiconductor Corporation/Cyrix-West as a Microprocessor Architect working with the Architecture Group in x86 architecture designs in Santa Clara, CA. USA.



Immanuel Rahardja obtained his Electrical Engineering Bachelor degree from the University of Toronto in 1992. He worked at Xilinx from 1992 to 1996 on various projects exploring and evaluating different FPGA architectures. He currently works at a startup company, Aristo Technology, developing the graphical user interface software for a set of block-level system IC planning and implementation design tools.