# A High-Speed FPGA Using Programmable Mini-tiles

**Paul Chow[1], Soon Ong Seo, Kevin Chung, Gerard Paez[2], and Jonathan Rose**
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 1A4

## Abstract

Field-Programmable Gate Arrays (FPGAs) are now a recognized technology for the implementation of digital systems, but they suffer from reduced speed and logic density compared to mask-programmed gate arrays. Many studies have been performed concerning the effect of an FPGA's architecture on its speed and density. In this paper we describe how these studies are used in an actual implementation of a high-performance FPGA.

The architecture of the FPGA logic block was determined through an experimental process using custom-built CAD tools. The result is a logic block that is an asymmetric tree of four-input lookup tables that are hardwired together. A segmented routing architecture, also tuned using experiments, is employed to improve the speed of the interconnect.

To address the problems of full-custom design, a novel layout style for FPGAs is proposed. It can be likened to the technique used in PLAs, in which a 'mini-tile' contains a portion of most components in the logic tile. The mini-tile is optimized for layout density and speed, and placed into a 4x4 array, where it is then customized by adding vias to obtain the desired hardwired connections. As well as providing ease of layout, this technique gives the capability to easily change the hardwired connections in the logic block architecture, and the segmentation length distribution in the routing architecture.

## 1 Introduction

The generality of Field-Programmable Gate Array (FPGA) technology permits the design of reasonably complex digital circuits using a commodity part. The price of this generality is a reduced maximum clock rate for systems, and a reduction in the amount of logic that can be integrated onto one chip. A mask-programmed gate array (MPGA) implementation is roughly three times faster and requires about ten times less silicon area when compared to an FPGA implementation [1]. In this paper we describe the design

[1]Email: pc@eecg.toronto.edu
[2]Now with CEMISID, Universidad de Los Andes, Venezuela.

and implementation of an FPGA where the primary goal is high performance with a secondary goal of maintaining a "reasonable" logic density.

FPGAs were first introduced in 1986 by Xilinx using a RAM-based programming technology [2]. Since then there have been many new commercial architectures and many new programming technologies [3, 4, 5, 6]. All of these papers describe the architectures and their various features but very little discussion is devoted towards the design decisions that resulted in the architectures. Even less is mentioned about how the choices were made in circuit design.

A few non-commercial FPGA architectures have been described, and details about those designs are more readily available. The Triptych FPGA [7] matches the physical structure of the routing architecture to the fanin/fanout nature of the structure of digital logic by using short hardwired connections to the nearest neighbours. Segmented channels are used between the columns to provide for fanouts greater than one. No discussion is given about how the segmentation length distribution was selected. This routing architecture does not allow arbitrary point-to-point routing as provided 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 [8] 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.

Early work at the University of Toronto resulted in the implementation of an architecture (UTFPGA1) using three cascaded four-input logic blocks and segmented routing [9]. UTFPGA1 used information from our architectural studies but there was very little done to achieve a good design from the circuit and layout aspects. This was a first attempt that provided some insight into the problems faced in the design and layout of an FPGA.

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 [10]. This is also true in the design of an FPGA, where it is important that the CAD tools match well 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

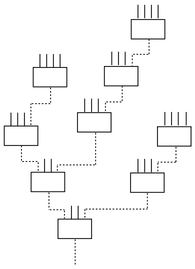

Figure 1: Circuit connections without hardwired connections.

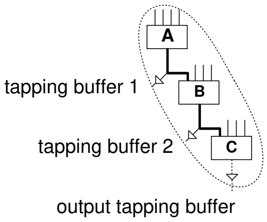


Figure 2: Hardwired logic block.

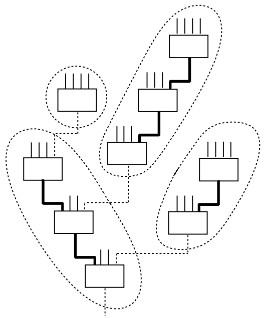

Figure 3: Circuit connections using hardwired logic block.

implemented in our test chip, called LEGO[3].

In the following section we summarize the results that led to the choice of our logic block architecture. Section 3 discusses the decision about the segmented routing architecture used. The circuit design and layout issues of the chip are presented in Section 4. The results are summarized in Section 5.

## 2 Logic Block Architecture

The speed and density gap between FPGAs and MPGAs is mostly caused by the routing structures used to connect logic components in each technology. In MPGAs, logic blocks are connected with mask-programmed metal wires, whereas in FPGAs, logic block pins are connected using field-programmed connections, which are wire segments linked by programmable switches. Regardless of the type of programmable switches used in the FPGA (e.g. static RAM-controlled pass transistors [3] or anti-fuses [4]), the capacitance, resistance, and size of programmable connections makes them much slower and larger than a simple metal wire.

One way to improve the speed and density of FPGAs is to replace some of the programmable connections between basic logic blocks with hardwired connections, which are simple metal wires. By using hardwired links to construct coarser-grained logic blocks (called Hardwired Logic Blocks, or HLBs) from several basic blocks, the delay and size of circuits can be reduced. For example, Figure 1 shows a network of basic blocks with five programmable connections in the routing along the critical path and nine programmable connections in total. Suppose that three of the basic blocks are hardwired together to create a hardwired logic block with a fast three-block path as shown in Figure 2. Now the resulting circuit in Figure 3 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 connec-

[3] **L**ogic that's **E**rasable and **G**reatly **O**ptimized

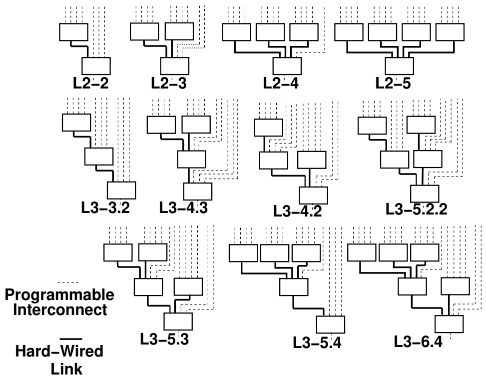


Figure 4: Example tree topologies.

tions has been reduced from nine to four and this may lead to a significant reduction in routing area.

### 2.1 The Hardwired Logic Block Design Space

There are many possible hardwired logic blocks and each possibility offers different delay and area for a given circuit. In general, the more hardwired links in a hardwired logic block, the faster will be the circuits implemented using that logic block. However, more hardwired connections leads to less connection flexibility and this may lead to lower density. There are many possible topologies for connecting basic logic blocks using hardwired links. Some of the tree topologies that were investigated for speed and area-efficiency are illustrated in Figure 4[4]. The granularity of the basic block also affects the delay and area of the implemented circuit. In FPGAs without hardwired connected blocks, smaller basic blocks tend to be better for logic density while coarse-grained basic blocks are better for speed. The results of previous studies [11, 12] have shown that for FPGA architectures without hardwired connections, a 4-input lookup table (4-LUT) is a good choice for both density and speed. The goal of the experiments described below is to select from among the many possible hardwired 4-LUT architectures one that produces high system speeds, at a reasonable cost in area.

An important architectural assumption is that each HLB features tapping buffers on the output of each LUT basic block to make the output accessible to the routing. Tapping buffers lead to faster HLB circuits since the output of one basic block can be accessed directly instead of propagating it through another basic block. Tapping buffers also improve logic density since unrelated pieces of logic can be packed together in the same HLB, with each piece using a separate tapped basic block output. In summary, with HLBs one gets

[4]The topology name is the letter "L" followed by the height of the HLB followed by "-" followed by a listing of sizes of the subtrees from a pre-order traversal separated by "_". Subtrees of size 0 and 1 are not listed.

a coarse-grained logic block, which leads to fast circuits, yet the HLB tapping buffers allow use of the relatively fine-grained basic blocks of an HLB, which leads to reasonable logic density.

To evaluate the various HLB topologies, a set of 15 MCNC combinational benchmark circuits are each implemented in 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 over all circuits are summarized for each HLB topology.

The area of the circuit consists of the logic area and the routing area. The logic area consists of the area of the LUT bits, tapping buffers, and the D flip-flops. For routing area estimates, we assume a routing architecture similar to Brown [13]. The basic architecture is illustrated in Figure 6, consisting of switching from pins to channel wires and between channel wires. We perform global routing to find the maximum number of routing tracks over all channel segments. This maximum number is assumed for every channel segment. The routing area is approximated by the area of the programmable switches in the connection blocks (C) and the switch blocks (S). The number of programmable switches in each connection block is the product of the number of HLB I/O pins and the number of routing tracks to which each pin can connect. The number of programmable switches in each switch block is determined by the number of tracks each track can connect to at a switch block, as well as the number of tracks in each channel segment.

The delay of the circuit is the sum of the LUT delays and the programmable connection delays along the critical path. We assume zero delay for hardwired connections. The delay for a 4-LUT was determined by SPICE simulations. The programmable connection delay is difficult to determine since it depends upon several circuit and routing architecture properties, such as connection fanout, number of switching stages in the connection and number of switches connected to each channel. We chose the simplest possible model for average programmable connection delay, a constant value parameter, since our studies concerned only logic block architecture. Note that using constant delay per programmable connection penalizes larger HLBs. With a larger HLB, the average number of switching stages per connection decreases in the circuit, and thus the average programmable connection delay should also decrease.

In the following, two studies are described: one in which the circuits are optimized purely for delay, and then purely for area. By comparing how the blocks perform in each domain, we can deduce a choice that performs well in both.

## 2.2 Experimental Results: Speed Study

In the first experiment, the circuits were optimized for the greatest possible speed, sometimes at significant area expense. The HLB topologies investigated were restricted to all possible tree topologies with nine or fewer 4-LUTs

| # LUTs | HLB Name | Speed Factor | | Area Factor | |
|---|---|---|---|---|---|
| 1 | L1 | 1 | (35 MHz) | 1 | (15.3 x $10^6$ $\mu m^2$) |
| 2 | L2-2 | 1.14 | | 1.19 | |
| 3 | L2-3 | 1.26 | | 1.21 | |
| 4 | L2-4 | 1.34 | | 1.32 | |
| 4 | L3-4_2 | 1.28 | | 1.31 | |
| 5 | L2-5 | 1.42 | | 1.40 | |
| 5 | L3-5_2 | 1.37 | | 1.37 | |
| 6 | L3-6_5 | 1.43 | | 1.65 | |
| 7 | L3-7_3 | 1.43 | | 1.66 | |
| 8 | L3-8_3_2 | 1.47 | | 1.61 | |
| 9 | L3-9_4_2 | 1.50 | | 1.70 | |

Table 1: HLB speedups with respect to no hardwired links.

and three or fewer LUT levels. HLBs with more than nine 4-LUTs would provide greater speedups but require too much area to be considered practical.

In Table 1 each row shows the HLBs with a given number of 4-LUTs that produce the greatest speedup. If there are two rows for a given number of LUTs, the second entry is an alternative HLB that offers a slower speed but at a lower area cost. The first column of Table 1 lists the number of 4-LUTs in the HLB, the second column shows the name of the HLB topology, the third column displays the normalized speedup of the HLB with respect to L1 (a 4-LUT FPGA with no hardwired links) and the last column shows the normalized area with respect to L1. The geometric mean of the areas[5] of the circuits for L1 is 15.3 x $10^6$ $\mu m^2$ and the mean of the delays is 28.5 ns or a mean system speed of 35.0 MHz, assuming a 1.2$\mu$m CMOS process.

The results in Table 1 show that the HLBs investigated can increase system speed from 14% up to 50% compared to L1. However, the area cost of these speedups is an increase of 19% to 70% in circuit size. The area increase is caused by the HLB mapper, which replicates LUTs during delay optimization to place as many hardwired links as possible in the HLBs along the critical paths. The addition of only one hardwired LUT (L2-2 HLB) yields a speedup of 14% with an area cost of 19%. Adding a second hardwired LUT (L2-3) gives a further 12% speedup (system speed of 44 MHz) but only an extra 2% area cost. The fastest investigated HLB (L3-9_4_2) yields a system speed of 52.5 MHz but costs 70% more in area compared to L1.

### 2.3 Experimental Results: Area-efficiency Study

The goal of this study is to find the minimum area cost of using each HLB topology to implement the benchmark set. The same HLB topologies as in

[5] Area measurements are in 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.

| # LUTs | HLB Name | Area Factor | | Speed Factor | |
|---|---|---|---|---|---|
| 1 | L1 | 1 | ($6.7 \times 10^6$ $\mu m^2$) | 1 | (22.1 MHz) |
| 2 | L2-2 | 0.93 | | 1.17 | |
| 3 | L2-3 | 0.96 | | 1.17 | |
| 4 | L2-4 | 1.00 | | 1.09 | |
| 4 | L3-4_2 | 0.93 | ($9.67 \times 10^6$ DSM$^2$) | 1.27 | (28 MHz) |
| 5 | L2-5 | 1.10 | | 1.03 | |
| 5 | L3-5_2 | 1.04 | | 1.24 | |
| 6 | L3-6_5 | 1.23 | | 1.04 | |
| 7 | L3-7_3 | 1.28 | | 1.19 | |
| 8 | L3-8_3_2 | 1.32 | | 1.25 | |
| 9 | L3-9_4_2 | 1.42 | | 1.24 | |

Table 2: HLB areas with respect to no hardwired links.

the speed study are investigated. It should be noted that when minimizing area, the HLB mapper does not replicate LUTs.

Table 2 presents the same HLB topologies as in the previous table but for area-optimized circuits. Note that even though the circuits are optimized for area the presence of hardwired links in the HLBs will cause a speedup with respect to L1, a 4-LUT FPGA without hardwired links. The first column lists the number of 4-LUTs in the HLB, the second column lists the HLB name, the third column lists the normalized area with respect to L1 and the fourth column lists the normalized speed with respect to L1. The geometric mean of the area of the area-optimized L1 circuits is $6.7 \times 10^6$ $\mu m^2$ and the mean of the delays is 45.3 ns or a mean system speed of 22.1 MHz.

From Table 2, there are three HLBs, L2-2, L2-3 and L3-4_2, that offer better logic density and speed than the 4-LUT FPGA without hardwired connections. The L3-4_2 HLB offers the best logic density and the best speedup among the HLBs in Table 2. In fact, the speed of the area-optimized circuits using L3-4_2 is only 20% slower than the delay-optimized L1 circuits, while using only 40% of the area of the delay-optimized L1 circuits. From Table 1, when optimizing purely for delay, the L3-4_2 HLB offers a 28% speedup over L1 with only a 31% area increase. The area-efficiency of L3-4_2 when optimizing purely for area coupled with the good speedup and area-efficiency when optimizing purely for delay led us to conclude that the L3-4_2 HLB was the best overall HLB.

# 3 Segmented Routing Architecture Design

The effect of an FPGA's routing architecture on system speed and integration logic density can be very significant. As discussed above, the main reason is that the programmable switches, which are pass transistors driven by SRAM cells, have significant resistance and capacitance and require large area. The *routing architecture*, which is the manner in which wire segments and switches are organized, can be designed to mitigate these effects.

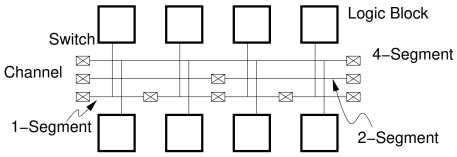


Figure 5: Segmented routing architecture example.

In the original Xilinx 2000 and 3000 architectures [2, 14], a very simple architecture was employed: most wire segments spanned only the length (or width) of one logic block, having switches at either 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 5. In this way, a segment with appropriate length for a connection can be selected, which results in less resistance along the path.

This leads to the architectural question: How many segments of each length should there be? That is, what is the *segmentation length distribution*? There is a tradeoff in this design question: the more longer segments there are, the greater the likelihood that long and otherwise time-consuming connections will be routed with fewer series switches, resulting in a faster circuit. On the other hand, an excessive number of longer segments will mean that some of the long segments will have to be used for short connections that do not span the entire segment length, 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 either to logic blocks or other routing segments. The advantage of un-populated segments is that there is less parasitic switch capacitance connected to the segment, which makes it faster. The disadvantage is that the reduction in routing flexibility may result in the need for more tracks and so a loss of logic density.

In this section we will summarize a study on segmented routing architecture distribution and population [15] 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 [13]. The number of tracks needed to complete the route using segments of length one is called the minimum. When the channel includes segments of other lengths, the number of tracks required above the minimum is measured.

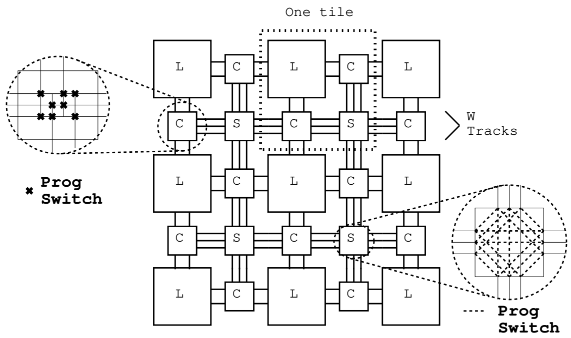


Figure 6: Architecture Definitions.

## 3.1 Definitions and Experiments

Figure 6 illustrates the broad architectural features of the LEGO FPGA that is being designed. 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. Thus 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.

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. Thus $D_1 + D_2 + D_3 = 1$ and the number of tracks containing segments of length $i$ is $D_i \times W$. Note that the longer segments pass through switch blocks without actually traversing a programmable switch.

As mentioned above, it is possible to consider the removal (or de-population) of the switches internal to a segment. We divide this population question 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 $SB_u$ 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.

## 3.2 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

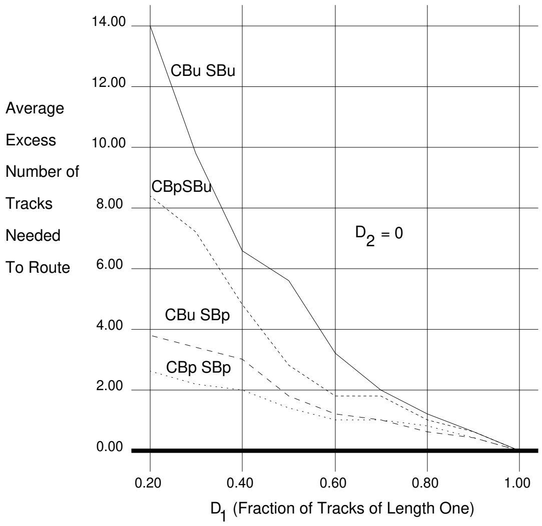


Figure 7: Population comparison.

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 the general results discussed below will hold for most logic blocks.

### 3.2.1 Population of Segments

The first question we address is that of population of the S and C blocks. Figure 7 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. 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.

The data show that both cases in which the switch block is un-populated (the upper two curves) result in significantly greater 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.

### 3.2.2 Segmentation Distribution

Table 3 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

| D2/D3 | 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 | | | | | | | | |

Table 3: Average number of excess tracks for each distribution $CB_u, SB_p$.

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 3 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 marked with a * in Table 3) would reflect such a choice. The faster architectures are the ones with higher values of $D_3$.

In Section 4.2 we give the reasons that lead to the choice of 16 tracks 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.

# 4 Chip Design

Given that the major architectural choices have been made there are still many decisions in the circuit-level implementation. The determination of the hardwired logic block and the segmented routing were done independent of the circuit design issues. The only implementation consideration was the area and timing models used to select the logic block. However, there were a few lower-level architectural questions that could only be answered after we considered the circuit and layout design. We will address those here, as well as describe a novel layout methodology.

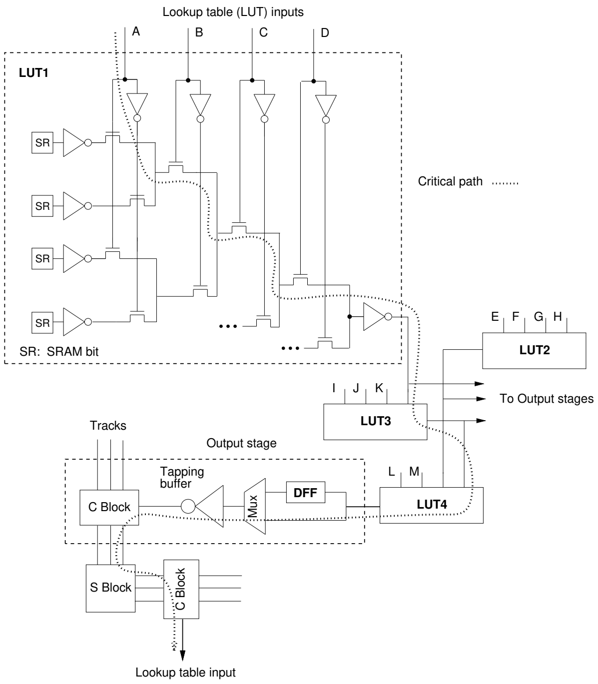


Figure 8: Critical Path

## 4.1 Circuit Design

The primary objective of this work is to achieve high speed by finding better architectures for FPGAs. Even with a good architecture, it is still necessary to carefully tune the circuit design and we address some of the important circuit issues here. The technology that we used is a 1.2$\mu$m two-level metal n-well CMOS process.

Recall that the logic block chosen is the one labelled L3-4_2 in Figure 4. Each 4-input LUT is implemented with a multiplexer tree, attached to static RAM cells. The critical path is shown in Figure 8. It starts at the most heavily loaded input of the first LUT multiplexer tree. Note that the outputs of the cascaded LUTs connect to the least loaded input of the next LUT. (Not only is this the least loaded input, but it also has the shortest distance that the signal can travel, which is a significant saving in delay compared to the other inputs). From the LUTs the path goes through a multiplexer that selects the combinational output or a latched output before going to the C block. In this case, the combinational path is used. The C block connects the signal to a track, which then goes through an S block (or number of S blocks) to another C block and then the input of another lookup table.

Extensive simulation with Hspice[16] was used to do the circuit design. Here we address the individual components separately.

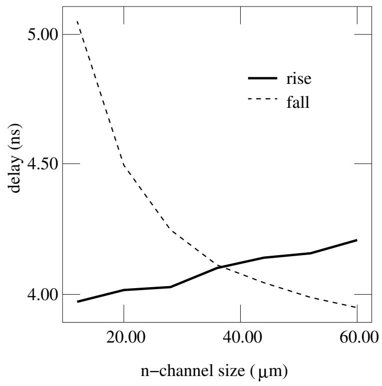


Figure 9: Transistor sizing for the logic block drivers.

### 4.1.1 Switch and Driver Transistor Sizing

Transistor sizing is a major concern for achieving high performance. In Triptych [7], the method of Logical Effort [17] 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, which were used for the switches, 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 find an initial driver size. We then tuned the design by simulating with different switch sizes. An example result is shown in Figure 9, 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 37$\mu$m for the size of the n-channel driver pulldown. In LEGO we actually used 34$\mu$m because in our initial simulations we used nine tracks in the channels, which was later changed to 16 tracks, and thus a higher load.

### 4.1.2 Muxes versus Switches

The C block, as illustrated in Figure 6, connects the track to the logic block inputs and the logic block outputs to the track. Consider the track to logic block connection. The connection can be done by means of switches, where there is a single switch for each track to pin connection, or by multiplexing $N$ tracks to each input pin. The output pins can be treated in a similar fashion using switches or de-multiplexers. In UTFPGA1 [9] we chose to use 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 saves in 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

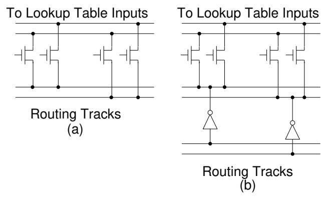


Figure 10: (a) C-block inputs with no receivers. (b) C-block inputs with isolation.

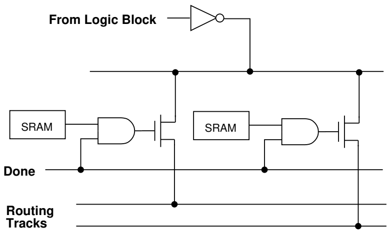


Figure 11: Power-up and programming protection without affecting the critical path.

track provides some fanout capability. Although using a multiplexer should result 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.

### 4.1.3 Routing Track Isolation

Another important issue is the capacitance at the C block inputs. With the switches connected as shown in Figure 10(a) the capacitance 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 [7] logic block, and their solution was to use gate logic instead of switch logic. The problems in LEGO can be reduced by adding a driver 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 10(b). This is the configuration used in LEGO.

### 4.1.4 Power-up and Programming Protection

FPGAs face a difficult problem when the chip first powers up: if the programming bits are unfavorable, then several buffers may be driving the same line in opposite directions, causing a current that may be enough to physically damage the chip. This problem has been previously addressed [9, 7]. One possibility is to make the driver a tri-state driver, but this has the disadvantage that the transistors that are used to disable the driver are in the critical path, either directly in series with the driver transistors or in the path that turns on the drive transistors. The solution used in LEGO is shown in Figure 11. We use the switch transistor to isolate the driver from the tracks.

## 4.2 Layout Style

Besides the desire to achieve a high-performance circuit design 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 6, 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 would be 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 smaller granularity.

The technique that we used was to divide the tile into 16 mini-tiles. Each mini-tile contains one sixteenth of the L block, S block, and C blocks. The tile[6] 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 8. The motivation for this topology comes from the topology used for PLAs [18], where the goal is to make the overall structure regular and have irregularity restricted to the final programming. Figure 12 shows a schematic of the mini-tile without details of the lookup table (LUT) logic included.

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 below. Figure 12 shows all the necessary wiring. Examples of via placement are shown by the Xs. The horizontal and vertical tracks of Figure 6 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 a $\frac{1}{4}$ 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 $\frac{1}{4}$ Mux-tree are used to connect between the mini-tiles to form the full 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 to make the full lookup table.

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

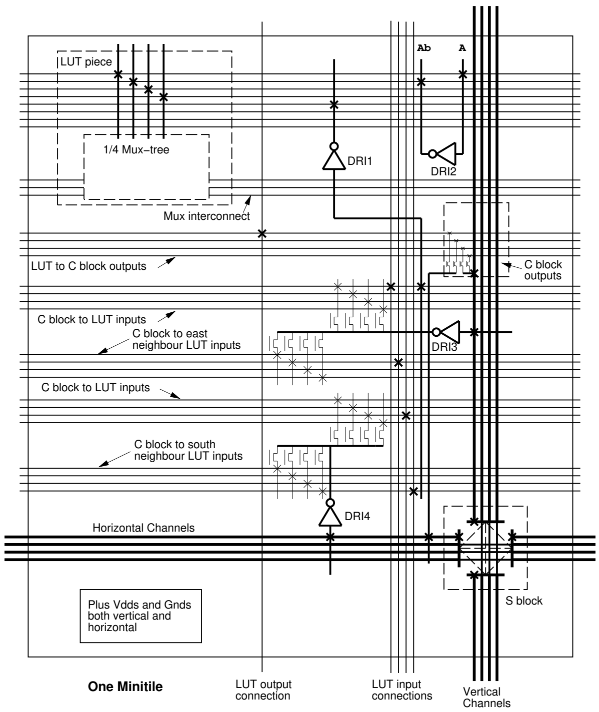


Figure 12: Mini-tile schematic.

In addition, the outputs of the LUTs are routed on the *mux interconnect* tracks to the edge of the tile for connection to the D flip-flops and the driver to the tracks. The output of the driver is routed back into the mini-tiles along a *LUT to C block outputs* track for connection to the C block outputs. 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 8. Driver *DRI1* drives the input from the C block to the LUT inputs.

The drivers marked *DRI3* and *DRI4* are the track isolation drivers shown in Figure 10(b). The transistors at the outputs of these drivers are the switches described in Section 4.1.2. In our 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 some time doing local optimization of the mini-tile. A plot of the mini-tile is shown in Figure 13 and the floorplan is shown in Figure 14. It can be seen that 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 hori-

Figure 13: Mini-tile plot.

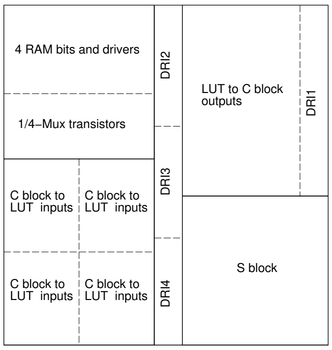


Figure 14: Mini-tile floorplan.

zontal dimension could be reduced by about 25%. With this area removed it can be seen that 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[7].

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 hardwired interconnections for the LUTs. This may be of benefit if our work in non-homogeneous logic blocks suggests that a mix of tiles with different types of hardwired connections is beneficial.

### 4.2.1 Other Layout Issues

The use of the mini-tiles made it easier to consider other issues such as access of the programming bits and the power distribution.

For programming, the top of the chip contains a shift register into which words of bits are serially loaded. Each mini-tile contains five word lines and seven bit lines. When the shift register is loaded, one of the word lines in the array is selected and data in the shift register are transferred into the array along the bit lines.

[7] Especially in a rapidly evolving industry.

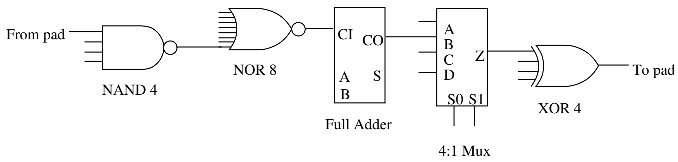


Figure 15: Benchmark combinational circuit.

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.

# 5 Results

The size of each tile, which contains four four-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 8. The equivalent path in a Xilinx 3000 FPGA would likely have to traverse at least two Configurable Logic Blocks (CLBs) with a delay of at least 11ns (in the -125 rated part). While this does not include inter-CLB routing time, the 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 15. The figure and some of the numbers in Table 4 are taken from some performance benchmarks published by Crosspoint Solutions [19]. 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[8]. 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.

[8]The Xilinx number takes this result and multiplies by a factor of 5/7 to approximate the speed of the XC4005-5 part. The *Published* number is actually for a XC4005-7 part.

| FPGA | Published | Adjusted |
|---|---|---|
| | *ns* | *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 | 58.0 | 23.6 |
| LEGO | N/A | 11.5 |

Table 4: Performance comparisons.

# 6 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. The focus on high-performance illustrated important issues in circuit design that affect the critical path delay. We also presented a layout methodology for FPGAs that reduces the amount of custom layout required without sacrificing too much density.

A prototype of the LEGO architecture, containing two tiles, went to fabrication in September 1992.

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 have not been addressed. In the case of LEGO, D flip-flops were added at the output of each tapping buffer in the hardwired logic block. The flip-flops are clocked from the system clock with an enable that can be selected from the routing channel.

# Acknowledgements

This work was supported by funding from MICRONET, a Network of Centres of Excellence on Microelectronic Devices, Circuits and Systems for Ultra Large Scale Integration. Fabrication is provided by the Canadian Microelectronics Corporation and Northern Telecom.

We would like to thank the referees for their comments, which helped to improve the presentation of this work.

# References

[1] Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic. *Field-Programmable Gate Arrays.* Kluwer Academic Publishers, 1992.

[2] William S. Carter, Khue Duong, Ross H. Freeman, Hung-Cheng Hsieh, Jason Y. Ja, John E. Mahoney, Luan T. Ngo, and Shelly L. Sze. A User Programmable Reconfigurable Logic Array. In *Custom Integrated Circuits Conference*, pages 233–235. IEEE, 1986.

[3] Hung-Cheng Hsieh, William S. Carter, Jason Ja, Ed Cheung, Steve Schreifels, Chuck Erickson, Philip Freidin, and Liane Tinkey. Third-Generation Architecture Boosts

Speed and Density of Field-Programmable Gate Arrays. In *Custom Integrated Circuits Conference*, pages 31.2.1–31.2.7. IEEE, 1990.

[4] Mike Ahrens, Abbas El Gamal, Doug Galbraith, Jonathan Greene, Sinan Kaptanoglu, Lynn Hutchings K.R. Dharmarajan, Sifuei Ku, Phil McGibney, John McGowan, Amer Samie, Kitty Shaw, Norma Stiawalt, Telle Whitney, Tom Wong, Wayne Wong, and Bortay Wu. An FPGA Family Optimized for High Densities and Reduced Routing Delay. In *Custom Integrated Circuits Conference*, pages 31.5.1–31.5.4. IEEE, 1990.

[5] David Marple and Larry Cooke. An MPGA Compatible FPGA Architecture. In *Custom Integrated Circuits Conference*, pages 4.2.1–4.2.4. IEEE, 1992.

[6] Algotronix Ltd., Edinburgh, Scotland. *CAL 1024 Datasheet*, 1989.

[7] Carl Ebeling, Gaetano Borriello, Scott A. Hauck, David Song, and Elizabeth A. Walkup. TRIPTYCH: a New FPGA Architecture. In Will Moore and Wayne Luk, editors, *FPGAs*, chapter 3.1, pages 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.

[8] 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, September 1992.

[9] Paul Chow, Soon Ong Seo, Dennis Au, Terrence Choy, Bahram Fallah, David Lewis, Cherry Li, and Jonathan Rose. A 1.2$\mu$m CMOS FPGA Using Cascaded Logic Blocks and Segmented Routing. In Will Moore and Wayne Luk, editors, *FPGAs*, chapter 3.2, pages 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.

[10] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[11] 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*, 25(5):1217–1225, October 1990.

[12] Satwant Singh, Jonathan Rose, Paul Chow, and David Lewis. The Effect of Logic Block Architecture on FPGA Performance. *IEEE Journal of Solid-State Circuits*, 27(3):281–287, March 1992.

[13] Stephen Brown, Jonathan Rose, and Zvonko G. Vranesic. A Detailed Router for Field-Programmable Gate Arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(5):620–628, May 1992.

[14] Hung-Cheng Hsieh, Khue Dong, Jason Y. Ja Roy Kanazawa, Luan T. Ngo, Liane G. Tinkey, William S. Carter, and Ross H. Freeman. A 9000-Gate User-Programmable Gate Array. In *Custom Integrated Circuits Conference*, pages 15.3.1–15.3.7. IEEE, 1988.

[15] Gerad Paez, Immanuel Rahardja, and Jonathan Rose. On the Design of Area-Efficient Segmented Routing Architectures for Field-Programmable Gate Arrays. In preparation, 1992.

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

[17] 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*, pages 1–16, March 1991.

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

[19] Crosspoint Solutions. Performance benchmarks, rev 1.01 8/92, 1992.