Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference, pp. 26-43, March, 1992.

TRIPTYCH: An FPGA Architecture with Integrated Logic and Routing

Scott Hauck, Gaetano Borriello, Carl Ebeling
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Abstract

We describe Triptych, a new FPGA architecture, that blends logic and routing resources to achieve efficient implementation of a wide range of circuits in both area and speed. The physical structure of Triptych attempts to match the structure of factored logic functions, thus providing an efficient substrate in which to implement these circuits. This approach both requires and takes advantage of an integrated approach to the mapping, placement and routing process. We first describe the Triptych architecture in detail. This is followed by the development of a new method for architectural comparison of FPGAs that is free of irrelevant implementation effects. Then the Triptych, Xilinx, Algotronix, and Concurrent Logic architectures are compared using this method to obtain normalized area and performance figures for a wide range of circuits, including both datapath elements and control logic. Our results indicate that Triptych is more area-efficient (Xilinx mappings average 3.5 times larger than Triptych mappings) and has at least comparable delay characteristics.

1 Introduction

Field-programmable gate arrays (FPGAs) are chips that can be programmed to implement arbitrary circuit structures. This is achieved by using a combination of logic and interconnection resources, which are generally treated separately. That is, the FPGA is partitioned into a set of logic blocks interconnected by programmable wiring in the form of routing channels or a cross-bar matrix. The target circuit is mapped to an FPGA by dividing it into a set of logic functions and registers, each of which can be implemented by one logic block, and a netlist indicating how these elements are connected. The circuit is then realized by placing each logic function and register in a logic block, and routing the logic blocks together according to the netlist. This clear separation between the logic and interconnection resources is attractive because the mapping, placement and routing decisions are decoupled. The price for this separation is the large area and delay costs incurred for the flexible interconnection needed to support arbitrary routing requirements. This leads to architectures like Xilinx [4], where the routing resources consume more than 90% of the chip area . Even so, it is well known that the largest Xilinx FPGA (3090) seldom achieves more than 50% logic block utilization for random logic. A lack of interconnect resources also leads to decreased performance as critical paths are forced into more indirect routes.

Domain-specific FPGAs like the Algotronix CAL1024 [2] and the Concurrent Logic CFA6000 [5] increase the chip area devoted to logic by reducing the routing to nearest-neighbor communication (this categorization is not completely accurate since Xilinx FPGAs have some nearest-neighbor connections and the CFA 6000 has introduced local busses). The result is that these architectures are restricted to highly pipelined dataflow applications, for which they are more efficient than the general-purpose FPGAs. Implementing circuits with these FPGAs requires close attention to routing during placement, which is generally accomplished by manual placement and routing of regular structures, or using specialized tools optimized to specific circuit domains [9]. Some research has been done on integrating placement and routing, but this work is very preliminary [3].

It is clear that domain-specific FPGAs can benefit greatly from an integrated approach to mapping, placement and routing. It is also the case that general-purpose FPGAs can take advantage of the increased flexibility offered by this approach. While it is conceptually appealing to view routing as a separate process which can always be completed, this does not coincide with reality, and attempting to achieve this extracts a large cost in terms of flexible routing resources. We have taken this argument one step further to establish that the FPGA architecture itself can take advantage of an integrated approach to mapping. In this paper, we present a new FPGA architecture called Triptych, that blends logic and routing resources. That is, each routing and logic block (RLB) in the Triptych array can be used both to compute a logic function and to route signals. More importantly, the array is structured to match the inherent fanin/fanout tree structure of circuit graphs. This allows the physical layout of a mapped circuit to follow its logical structure, reducing the need for extensive routing resources. It also turns routing resources into a much more optimizable resource. Taking full advantage of this structure will require integrating mapping, placement and routing tools, but we show in Section 3 that this structure has significant potential across a wide range of circuit types when compared to existing FPGA architectures.

Triptych defines a structure that is amenable to both highly regular circuits with local communication and more general circuits including the random logic circuits associated with finite state machines. As such, Triptych falls somewhere between general-purpose and domain-specific FPGAs. For applications with local communication, RLBs are used primarily for logic, allowing implementations that are competitive with domain-specific FPGAs. For more general applications such as FSMs, a higher percentage of RLBs are used for signal routing. Thus, although logic density decreases with greater global routing demands, a tradeoff between routing and logic resources can be reached on a per-circuit basis instead of as an architecturally dictated percentage.

We begin this paper by describing the Triptych architecture in detail, giving some implementation examples to illustrate its use. We then compare the Triptych architecture to that of the Xilinx, Algotronix and Concurrent Logic FPGAs by measuring the implementation cost and performance of each for a variety of interesting circuits. We have not included Actel in this study because it uses an anti-fuse technology, which makes architectural comparison difficult [1, 6]. We conclude by summarizing the results of this research and discussing possible future directions.

2 The Triptych Architecture

The FPGA architecture we present in this paper differs from other FPGAs by matching the structure of the logic array to that of the target circuits, rather than providing an array of logic cells embedded in a general routing structure. By matching the physical structure to the logical structure, we reduce the amount of “random” routing that is otherwise required.

Figure 1 shows a high-level view of a typical multi-level combinational logic circuit. The flow is shown as unidirectional, from inputs to outputs. From the point of view of each input, the data flow forms a fanout tree (shown with solid arrows) to those outputs that the input affects. From the point of view of each output, the data flow forms a fanin tree (shown with dashed arrows) from those inputs it depends upon. It is this fanin/fanout tree form that Triptych emulates architecturally by arranging RLBs (routing and logic blocks) into columns, with each RLB having a short, hard-wired connection to its nearest neighbors in adjacent columns (see Figure 2).

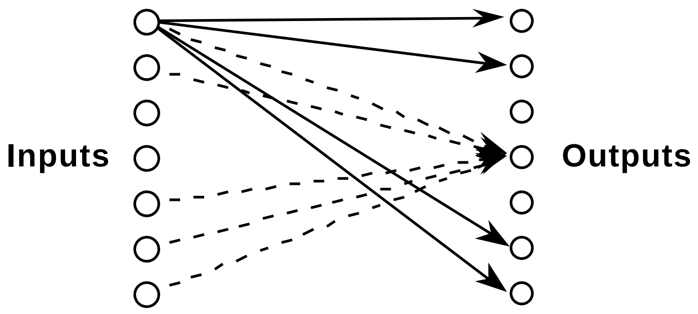

Figure 1. View of a multi-level combinational logic circuit as interleaved fanin/fanout trees.

The basic structure is augmented with segmented routing channels between the columns that facilitate larger fanout structures than is possible in the basic array. Finally, two copies of the array, flowing in opposite directions, are overlaid. Connections between the planes exist at the crossover points of the short diagonal wires. It is clear that this array does not allow arbitrary point-to-point routing like that associated with the Xilinx FPGA. However, we claim that this array matches the form of a large class of circuits, and that a mapping strategy that takes this structure into account can produce routable implementations.

Each RLB in the array has three inputs and three outputs, and may perform an arbitrary logic function of the three inputs, with the result optionally held by a master/slave D-latch. Routing in the Triptych array is in three forms: horizontally through the RLBs (by selecting an input to be routed to an output), diagonally through short wires to neighbors, and vertically through the segmented channels between columns of RLBs. Only one input and one output can be connected to the vertical wires; the other two must be on the local diagonal interconnect.

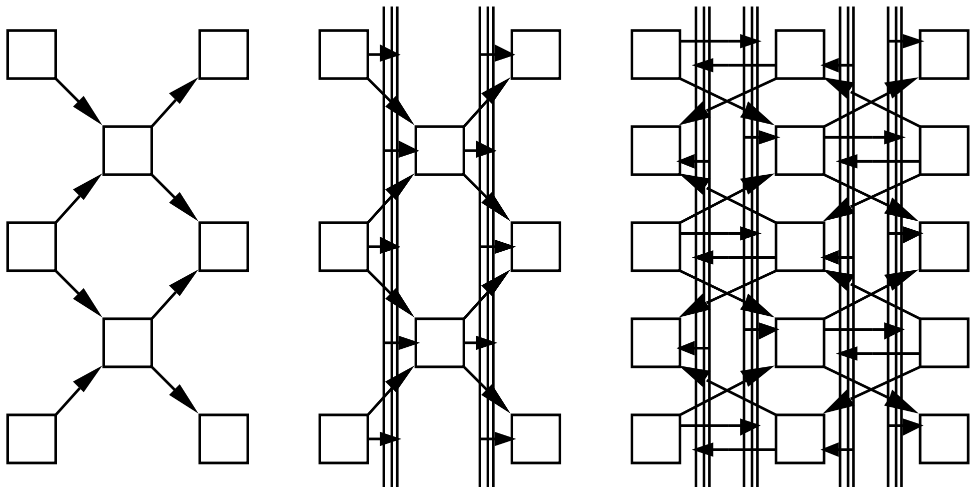

Figure 2. The overall structure of the Triptych FPGA shown in a progression of steps. The basic fanin/fanout structure on the left is augmented with segmented routing channels attached to a third RLB input and output. The structure on the right is obtained by merging two copies of the middle structure, with data flowing in opposite directions in the two copies. Not shown are the connections between the two copies, which permit internal feedback.

Circuits can be mapped onto this array by partitioning the logic into circuit DAGs containing nodes with at most three inputs. These DAGs are then mapped to the physical structure, with the inputs taken on one side and the outputs generated at the other. The nodes of the DAGs are placed such that input signals are available from the neighbor nodes or along a vertical connection. As suggested in [13], delay can be minimized by using mostly direct, hard-wired connections for the critical path. It is not the goal of Triptych implementations to achieve 100% logic utilization. Many RLBs are used to provide routing, either to fanout a signal or to pass it forward to the next level. Sometimes a mapping will even leave some cells unused to achieve a more routable placement of nodes. Examples of mappings are provided below.

2.1 RLB structure

A logical schematic of the Triptych RLB is show in Figure 3. As can be seen, the cell is designed to handle both function calculation and signal routing simultaneously (hence the name *routing and logic block*, RLB). It takes input from three sources and feeds them into a function block capable of computing any function of these three inputs, and the output can then be used in latched or unlatched form. The RLB's three outputs can choose from any of the three inputs and either the latched or unlatched version of the function block output. One last feature is the loopback from the master/slave D-latch, which enables the function to be dependent on its previous value. This is included for state machine implementation, although it may be used to output both the latched and unlatched versions of the function block. Again, only one of the inputs and one of the outputs can be connected to the vertical wires; the other two of each type are connected to the local diagonal wires.

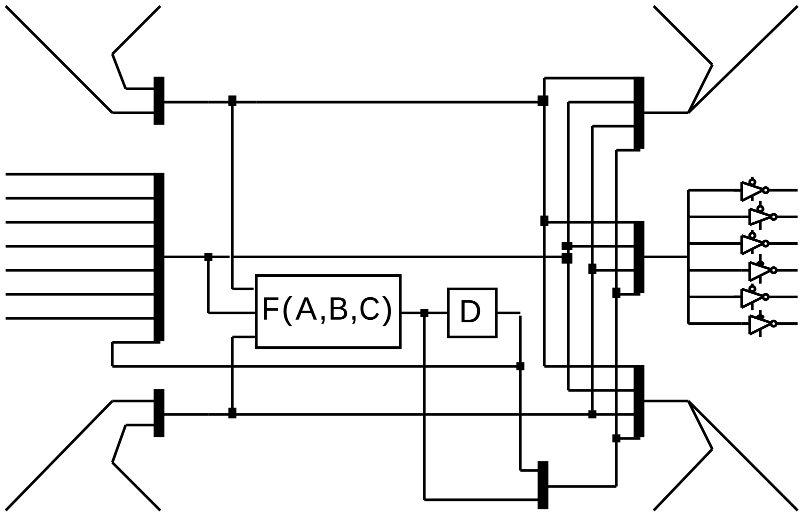

Figure 3. Triptych routing and logic block (RLB) design. The RLB consists of: 3 multiplexers for the inputs, a 3-input function block, a master/slave D-latch, a selector for the latched or unlatched result of the function, and 3 multiplexers for the outputs.

2.2 Typical RLB utilization

A Triptych RLB is capable of performing both function calculation and routing tasks simultaneously, which leads to several different uses of the RLB (see Figure 4). The three most obvious are: (a) a routing block with each input connected to one of the outputs; (b) a splitter with one of the inputs going to two or three of the outputs; and (c) as a function calculator with the three inputs going to the function block and the function going out the outputs. However, there are two important classes of hybrids that help produce more compact designs. The first comes from the observation that in blocks used to calculate a three-input function, the function block will most likely not go out all three outputs, and one or two of the input signals could be sent out the unused output connection(s), as in (d). Secondly, a function of two inputs can be implemented by making the function insensitive to the third input, thus allowing the unused input to be used to route an arbitrary signal, as in (e). An important observation is that the RLBs will never need to be used for one-input functions (i.e., an inverter), since any output signal will only be used either as an input to another arbitrary function block, where the inverter could be merged into the function computed, or to an output pin, where an optional inversion can be applied.

As was shown earlier, the Triptych FPGA has no global routing for moving signals horizontally. Instead, there is a heavy reliance on unused RLBs and unused portions of RLBs to perform these routing tasks.

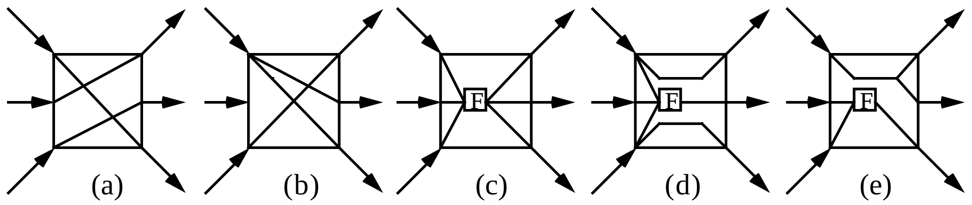

Figure 4. Five typical uses of Triptych routing and logic blocks (RLBs).

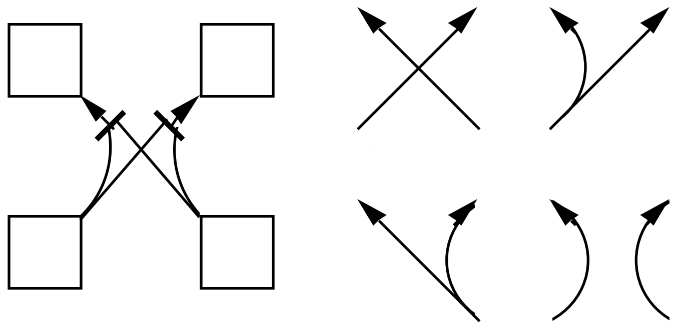

Figure 5. Schematic view of a pair of diagonals and the routing combinations they allow (implemented by a multiplexer at each diagonal input).

2.3 Interconnection

The Triptych RLBs are connected by three separate interconnection schemes. The first is for horizontal interconnect, and is accomplished through the RLBs as described above. The second is for local high-speed communication between neighboring RLBs and is achieved through “diagonals”. The detailed structure of the diagonals is shown in Figure 5. They allow outputs to be sent to the four RLBs immediately above and below them; two in the next column, which flow in the same direction, and two in the same column, which flow in the opposite direction. Diagonals are important for two reasons. Diagonals permit the construction of multilevel functions of more than three inputs without the speed penalty of general-purpose interconnect. They also allow signal flow to change direction both so that circuits can be more tightly packed and feedback can be provided for the implementation of sequential logic.

The third type of interconnect is used for longer range connections and large fanout nodes. It is implemented as a set of segmented “channel wires” between adjacent columns (see Figure 6) that connect middle outputs of RLBs to the middle inputs of RLBs flowing in the same direction in the next column. Needless to say, this flexibility leads to slower signal propagation, and speed-critical designs will avoid using the vertical channels for critical paths. There are 7 tracks in a vertical channel, with 6 handling inter-cell RLB routing and a seventh to carry a pin input. The 6 inter-cell tracks are broken up into two tracks each of 8, 16, and 32 RLB high segments.

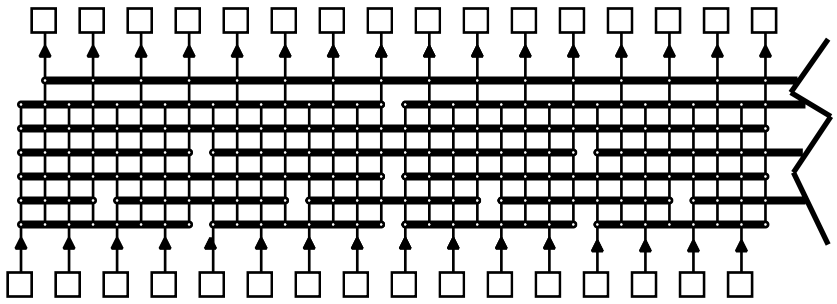

Figure 6. Top half of a segmented channel (on its side). The bottom half is a mirror image of the top.

One last important feature of the interconnect structure is how it handles the array borders. Since there are no RLBs beyond the right and left edges for the channel wires to route to, the channels on the edges tie the two directions of RLBs together. This way of handling the border cases leads to a different way of looking at the array, namely as a cylinder of RLBs. If the diagonals leading to the opposite direction of RLBs were cut except for those at the edges, the chip would appear to be a folded cylinder of RLBs. In fact, it is often helpful to think of the array as containing many smaller cylinders. For example, a six by six square of RLBs can be broken off from the rest of the array and considered to be a cylinder three RLBs high and twelve RLBs in circumference. This is not quite true, since the vertical channel for the left and right edges of the original six by six square will be unusable on the cylinder, but it can still be a useful abstraction for mapping. In fact, the current Triptych chip is an array of 64x8 RLBs, designed to yield a 32x16 cylinder.

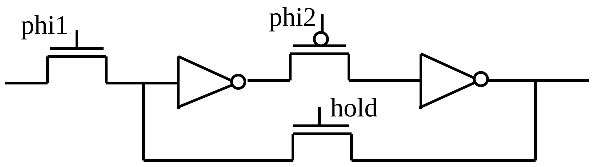

Figure 7. Programming bit cell schematic. The clock is only active during programming. A separate clock is used for the latches after the FPGA is configured.

2.4 Programming bit implementation and the scan path

Triptych is a RAM-based reprogrammable gate array with 26 memory bits per RLB, including those bits used for all three types of routing. The memory cells are implemented pseudo-statically with a “hold” signal asserted during normal operation and unasserted during programming. We found that this gave a much smaller layout than a fully static design (including the space needed for this extra hold line), especially when it was realized that the hold signal was necessary for disabling RLB output drivers during programming. However, since the pass gates controlled by phi2 and hold are non-complimentary, and thus the normally off transistors in the inverters will be slightly on, there will be extra static power dissipation during chip operation. These cells could easily be replaced by 5-transistor static RAM cells, and the overall RLB size would not change appreciably.

The memory cells are connected by a scan path running throughout the chip, allowing it to be programmed by cycling data through the bits. This scan path is also attached to the RLBs’ master/slave D-latches. This not only allows the chip to start in any combination of latch states, but also allows the contents of the latches to be shifted out after the chip has run an arbitrary number of cycles, thus aiding in debugging and chip testing.

2.5 Vital statistics

The speed of a path in a Triptych mapping can be calculated from the numbers given below in Table 1. For example, a path using 4 RLBs, 2 for routing and 2 for function calculation, and 1 channel wire would take 13.3 to 14.5 nanoseconds ($4\times1.6 + 2\times2.2 + 2.5$ to $3.7 = 13.3$ to 14.5). Note that being able to use such a simple speed calculation method is due both to the simplicity of the interconnect and also to the design philosophy of "independence of paths". Simply put, "independence of paths" means that gate logic is used in place of switch logic in much of the FPGA, making signal propagation insensitive to the amount of fanout contained in the current mapping.

Resource Used	Delay
RLB	1.6ns
Function Block	additional 2.2ns
Channel Wire	2.5-3.7ns

Table 1. Speed of important features, estimated using HSPICE with parameters for the 1.2µm CMOS n-well process available from MOSIS.

Routing resources, their programming bits, buffers and tri-state drivers account for only 60% of the total area, as compared to at least 90% for the Xilinx FPGA. Approximately one-fourth (26%) of a Triptych RLB is devoted to the 26 programming bit memory cells.

3 Using Triptych

In this section we present two examples of circuits mapped to Triptych. The purpose of these examples is to demonstrate the constraints on routing and how multilevel logic circuits do indeed map to the physical structure provided by Triptych. In these examples, each RLB is shown as a cell with three input entries and three output entries. Each entry indicates an incoming or outgoing signal. Note that each block may create a new signal by computing a logic function over the inputs. Diagonals and reverse diagonals that are used in the implementation are highlighted, as are connections to the channel wires. For clarity, only those vertical wires carrying signals are shown.

3.1 8-bit rotate function

The power of using columns of RLBs for routing only is shown in this example, which rotates a set of 8 bits 4 positions. Each level can be used to send one signal from each RLB to a neighbor of the final position. Since each RLB carries two outputs, one intermediate RLB column and two vertical channels are required to route the signals to their final destination (see Figure 7). This generalizes to the case where three signals are routed per RLB, which requires two intermediate RLB columns and three channels.

Generalizing this use of the vertical channels suggests a naive place and route algorithm that alternates columns of RLBs used for routing with columns used to compute logic functions. Subject to a sufficient number of routing tracks, this leads to a viable routing of arbitrary logic functions. However, as the next example shows, this scheme is much less area-efficient than is generally achievable.

Figure 7. Triptych mapping of a 4-bit rotate of 8 bits.

3.2 State machine example

Figure 8 shows the factored logic equations and Figure 9 the corresponding Triptych implementation for the ubiquitous traffic light example. This example shows that circuit mappings can be very compact if the individual logic blocks are correctly placed. The inputs and outputs of this circuit are all connected at the left and right of the array, except for three signals that use the pin input track of the vertical channels (shown dangling off the bottom). In this example 16 RLBs are used to compute logic functions, 2 RLBs are used only for routing, and 6 RLBs are left unused (these 6 RLBs must be counted in order to achieve a rectangular mapping; they might be used in neighboring circuits). Also, this circuit is assumed to be placed along the left edge of the chip, so the vertical tracks at that edge are used to connect RLBs in the same column. This is about as compact a Triptych layout as can be achieved for a random logic function.

```
INORDER = s1 s2 d1 st SB0 SB1;
OUTORDER = NSB0 NSB1 r1 y1 g1 r2 y2 g2 sd;
NSB0 = !st * !r2;    NSB1 = !st * !g1 * !g2;
r1 = NSB0;           r2 = !st * (SB0 * !9 + !SB0 * 9);
y1 = r2 * 51;        y2 = 53 + 45;
g1 = r2 * !51;       g2 = !st * !r2 * !y2;
sd = 12 + 45;        9  = !SB1 + !d1;
12 = !SB1 * 18;      18 = !SB0 * s2 * !st;
46 = !st * SB0;      45 = s1 * !SB1 * 46;
52 = !d1 * SB1;      51 = s2 * !SB1 + !SB0 * SB1;
53 = 52 * 46;
```

Figure 8. Factored logic equations for the traffic light controller finite state machine.

Figure 9. Triptych realization of the traffic light controller.

4 Area comparisons with other FPGAs

In this section, we present a comparison of the area efficiency of four different FPGA architectures, based on several representative examples. The FPGAs compared are CLI’s CFA, Algotronix’s CAL1024, the Xilinx 3000 series, and Triptych. The example circuits were chosen across a wide range of domains, rather than a specific one for which an architecture might have been optimized (e.g., bit-serial computations). The circuits were mapped by hand and only optimized for area. Automatic tools were not used as this would make the comparison as much about the CAD algorithms as the architectures themselves.

Even after eliminating these two factors, domain and CAD support, performing an architectural comparison between FPGAs is difficult because of complicating factors due to particular implementations. The most significant of these include the manufacturing process, the design rules employed, and the quantity and quality of designer effort. These factors make the straightforward approach of comparing actual chip area inadequate for purposes of architectural comparison. Simply using bits per cell also fails because it assumes that all FPGAs have equal per bit logic complexity, which is not true in current FPGAs (see discussion of table 2 below).

Our approach to architectural comparison is based on the premise that we can normalize cell area (the basic tiling structure in the FPGA) based on the size of the programming bits. There are two assumptions underlying this claim. First, a programming bit should occupy the same area if two different architectures are implemented on the same fabrication line, using the same design rules, and with the same amount of designer expertise and effort. It is not obvious that this is true, since the programming bits may be designed quite differently in different FPGAs. In the case of Triptych, the bits are 7-transistor pseudo-static shift-register cells interconnected in a scan-path, while Xilinx and CAL use 5-transistor static RAM cells (CFA's bit cell implementation is unspecified). However, even with these differences, we claim that the programming bits would take up the same area, due to the fact that the Triptych design can avoid ratioed transistors and use minimum-size devices (there is no feedback inverter to overpower through a pass-gate), and that it does not have the overhead of addressing and decode logic.

The second assumption is that the rest of the cell's logic and routing will scale proportionally to the programming bit cell size. This second assumption can be stated as the following relationship:

$$\frac{\text{total cell area in process } \alpha}{\text{total RAM bit area in process } \alpha} = \frac{\text{total cell area in process } \beta}{\text{total RAM bit area in process } \beta} \quad (1)$$

There are at least two ways in which this assumption may not hold true. First, speed impacts area, and one could design a very small cell using minimum-size devices that would be unreasonably slow. Since speed of programming is not a major issue, programming bit cells are almost always made as small as possible, so decreasing the logic size couldn't decrease the programming bit size, and the ratio given above would change. This means that any area comparison must also include a speed comparison of the circuit mappings in the sample. We present such a comparison below. Second, programming bit cells are the easiest and most advantageous cell components to optimize, due to the lack of speed requirements and their high degree of replication, and thus receive a much higher share of optimization efforts in less optimized designs. Also, as the programming bits shrink, the number of bit-equivalents the cell occupies increases. Therefore, less optimized implementations will compare less favorably to more highly optimized implementations (this is in fact the case for Triptych, thus handicapping it somewhat in the comparison).

Given these assumptions, we compare cell areas using the following normalization to a hypothetical standard process and design effort, Π:

$$\text{total cell area in process } \Pi \times \frac{1}{\text{single RAM bit area in process } \Pi} \qquad (2)$$

Note that this is simply the area of the cell scaled by the area of a single RAM bit, which our first assumption says will be the same size for any architecture. Equation 2 can be rewritten to yield:

$$= \text{ \# of RAM bits per cell} \times \frac{\text{total cell area in process } \Pi}{\text{total RAM bit area in process } \Pi} \qquad (3)$$

And finally, by applying equation 1, we obtain the following expression, which consists of three parameters easily obtained from specific FPGA implementations (where α is whatever process, design rules, and designer effort were used to implement that particular FPGA):

$$= \text{ \# of RAM bits per cell} \times \frac{\text{total cell area in process } \alpha}{\text{total RAM bit area in process } \alpha} \qquad (4)$$

Equations 2 to 4 allow us to scale all FPGA cell areas to our hypothetical process Π. Table 2 shows the results of applying equation 4 to the four FPGAs, with Triptych as the baseline for comparison.

	CFA	CAL	Xilinx	Triptych
RAM bits per cell	18	18	201	26
Percentage of cell area for bits	40%	40%	22%	26%
Normalization factor	0.45	0.45	9.1	1.0

Table 2. Area normalization figures calculated by applying equation 4, and the data used to obtain them. Triptych is used as the baseline. The numbers for CFA, CAL, and Xilinx were obtained from [7, 10, 14] respectively.

Normalization is also necessary for speed comparisons. In this case, we use the delay values provided by the manufacturers (Xilinx values are for the 3020-50 [12]), and scale them based on the feature sizes of the processes used. As shown in [15], FPGA logic speed scales linearly with process size, while routing delay scales quadratically. However, since the division of FPGA timings into routing-based and logic-based delays isn't always clear, we have scaled CFA linearly from 1.0µm to 1.2µm, and Xilinx quadratically from 1.25µm to 1.2µm, both of which are generous to the given FPGAs. For CAL, which has no long-distance routing from cell outputs, all delays were scaled linearly. Table 3 shows the normalization factors calculated for the FPGAs in question.

	CFA	CAL	Xilinx	Triptych
Technology feature size	1.0	1.5	1.25	1.2
Scaling method	Linear	Linear	Quadratic	—
Normalization factor	1.2	0.8	0.92	1.0

Table 3. Speed normalization figures and the data used to obtain them. A 1.2µm process is used as a baseline.

4.1 Summary of mapping results

Table 4 shows the results of area comparisons using this methodology for a wide range of circuits. They are collected into the following general categories: systolic, arithmetic, linear-growth bit-parallel, exponential-growth bit-parallel, and finite state machines (random logic). The systolic circuits are two versions of a string comparison circuit that computes the “edit distance” of two strings [11], the two versions differing on whether both strings move or if one is fixed in place, as well as a FIFO [8]. The arithmetic circuits chosen are a counter with a 4-stage carry look-ahead and an adder with a 2-stage carry look-ahead. The linear-growth bit-parallel circuits (circuits whose interconnection tends to scale linearly with problem size) include a simple equal/not-equal comparison and a shift register, while the exponential-growth bit-parallel circuits are a decoder and a multiplexor. Finally, for the random logic/FSMs, we chose the two smallest ISCAS benchmarks (s27 and s208), as well as the traffic light controller example.

Table 5 shows the results of delay comparisons for the circuits of Table 4. As discussed above, an FPGA architecture yielding much faster or slower circuits should have its area decreased or increased respectively to achieve a more accurate architectural comparison. Note also that these numbers do not form a true comparison of each FPGA’s inherent delay requirements for the given circuits, since the circuits were optimized purely for area. Instead, these speed numbers are a part of the area comparison, making sure that no FPGA completely ignores speed in the quest for even smaller cells.

Even under the model given above, which for several reasons is biased towards the other FPGAs, Triptych does quite well in both area and speed. For all but the counter, whose CFA mapping is only 10% smaller, Triptych mapping are consistently smaller than CFA mappings, quite often by a factor of 2 or more, and Xilinx mappings are usually three to four times larger. As to CAL mappings, the best are only about 10% smaller than Triptych mappings, with the worst being more than twice as large. At the same time, delays in these circuits are almost always worse, up to a factor of more than 4 in the larger multiplexors. All of these factors indicate that Triptych is a viable architecture. This is especially true since the only FPGA surveyed that comes close to matching Triptych’s area efficiency is CAL, which has no global routing facilities. This absence implies that larger, more complex circuits than those surveyed would be more difficult to map to CAL than to Triptych. They will be larger and slower, due to more cells being utilized for routing.

	size	Triptych area	CFA factor	CAL factor	Xilinx factor
Systolic					
Single-flow string compare (# of bits per character)	2	7	3.14	1.14	3.86
	4	10	2.90	1.10	3.60
	8	16	2.69	1.00	3.69
Double-flow string compare (# of bits per character)	2	20	2.30	1.00	2.75
	4	24	2.38	1.04	3.42
	8	36	2.17	1.00	3.81
FIFO (width in bits)	8	58	—	1.05	2.36
	16	90	—	1.00	2.32
	32	154	—	.96	2.31
	64	282	—	.93	2.29
Arithmetic					
Counter (width in bits)	8	24	.92	.92	2.67
	16	48	.90	.90	2.65
	32	96	.90	.90	2.66
Adder (width in bits)	8	40	1.63	.90	3.65
	16	80	1.63	.90	3.64
	32	160	1.62	.90	3.64
Linear-growth bit-parallel					
Comparator (width in bits)	8	8	2.25	.88	4.50
	16	16	2.25	.88	4.56
	32	32	2.38	.91	4.56
	64	64	2.36	.91	4.56
Shift Register (number of bits)	8	8	2.75	2.25	4.50
	16	16	2.69	2.25	4.56
	32	32	2.69	2.25	4.56
	64	64	2.70	2.25	4.55
Exponential-growth bit-parallel					
Decoder (inputs:outputs)	2:4	4	—	1.00	4.50
	3:8	10	—	1.00	3.60
	4:16	22	—	1.14	3.32
	5:32	46	—	1.35	3.57
Multiplexor (inputs:outputs)	4:1	3	2.67	1.67	4.67
	8:1	8	2.13	1.63	4.00
	16:1	18	2.06	1.67	3.78
	32:1	41	2.00	1.78	3.44
Random Logic/FSM					
s27		11	—	—	4.18
s208		59	—	—	2.93
traffic light controller		24	—	—	3.04
Geometric Means			2.06	1,17	3.53

Table 4. Area costs of circuits hand-mapped to four FPGAs using normalized area values. For each circuit, the Triptych area and its relative size in the three other FPGAs is listed (e.g., the Triptych s27 mapping is 11 normalized area units, while the Xilinx mapping is 4.18 times as big, or 46 area units).

	size	Triptych speed	CFA factor	CAL factor	Xilinx factor
Systolic					
Single-flow string compare (# of bits per character)	2	14.6	2.66	1.77	.91
	4	20.9	2.15	1.69	.64
	8	33.5	1.34	1.63	.59
Double-flow string compare (# of bits per character)	2	26.3	1.92	1.09	.76
	4	31.7	1.60	1.02	.84
	8	64.2	.88	.70	.62
FIFO (controller speed only)	—	34.0	—	3.01	1.52
Arithmetic					
Counter (width in bits)	8	21.2	1.62	1.52	.96
	16	32.0	1.63	1.56	1.27
	32	53.6	1.63	1.59	1.51
Adder (width in bits)	8	42.3	1.63	1.92	.88
	16	73.9	1.43	1.77	.92
	32	137.1	1.31	1.67	.94
Linear-growth bit-parallel					
Comparator (width in bits)	8	30.4	1.56	1.35	.87
	16	60.8	1.22	1.31	.87
	32	121.6	1.26	1.29	.87
	64	243.2	1.18	1.27	.87
Exponential-growth bit-parallel					
Decoder (inputs:outputs)	2:4	7.9	—	1.29	.96
	3:8	7.9	—	2.25	1.34
	4:16	19.0	—	1.47	.73
	5:32	24.0	—	1.88	.76
Multiplexor (inputs:outputs)	4:1	7.6	3.47	3.24	1.72
	8:1	13.9	2.68	3.05	1.50
	16:1	17.7	3.02	3.94	1.72
	32:1	26.5	2.88	4.32	1.52
Geometric Means			1.74	1.73	.99

Table 5. Delay of circuits hand-mapped to 4 FPGAs, in nanoseconds, using normalized delay values. The Triptych column contains actual speed numbers, while the other columns are the factor by which the given FPGA's mapping is slower (or faster) than Triptych's.

Three more points need to be made about these comparisons. First, the Xilinx mappings are the only ones for which we did not perform the assignment of routing resources. This was not done due to the complexity of hand-mapping Xilinx routing, and it was simply assumed that the circuit could be routed without leaving unreachable CLBs. This is not always the case in the 3000 series, where large designs can often only achieve 50% cell utilization due to the scarcity of resources. Thus, the area numbers for Xilinx should be considered as a lower bound, and the mappings may actually require significantly more space. Since we did not do routing, we have no direct method for measuring the speed of Xilinx circuits. However, by using a routing delay model for Xilinx [12], we were able to compute estimated routing delay, and these numbers were presented in Table 5. Second, since all the circuits were mapped by hand, we were only able to use small examples. No comparisons of large, complex random logic circuits is presented. Third, since FPGAs normally come in families, with the amount of cells in a chip variable, the issue of I/O blocks has been ignored and the circuits mapped to an unbroken plane of cells with no chip edges (except for a few circuits who actually map easier along a chip boundary, in which case an arbitrarily long boundary was assumed).

Clearly, more comparisons need to be made on larger circuits. We are currently developing automatic mapping tools to make this task tractable, although it will be more difficult to factor out CAD technology effects across the different architectures. We also need to compare the performance of circuits specifically optimized for speed instead of area. However, the results of Table 4 and Table 5 indicate the Triptych architecture is promising as it consistently yields smaller and faster circuits than the others.

5 Conclusions

The new FPGA architecture presented in this paper was motivated by three needs: including data-path and control elements in the same array; minimizing the space devoted to routing resources; and permitting the realization of delay-critical circuits. We believe that Triptych achieves these goals given the experience gained so far with many example circuits, a few of which have been presented above. The examples have proven to be more densely packed and to have delay characteristics at least comparable to the other FPGAs.

The most interesting and challenging future direction for research is automatic mapping. Triptych requires that the functional and interconnect elements not be treated separately. Combining the considerations for covering, placement, and routing should allow us to develop mapping tools that more precisely predict the performance of circuits and more accurately trade off density for speed. In fact, such an approach could benefit any FPGA.

In summary, we have a viable new FPGA architecture for circuits that demonstrates a higher area-efficiency, with comparable delay, for a wide range of circuits that include both data-path elements and control logic. There is much work to be done, especially in the area of automatic mapping, and promising directions are just beginning to be pursued.

Acknowledgements

Thanks to Christopher Hébert and the other students of CSE568, Winter quarter 1991 for contributing to the early design phases of the Triptych layout and mapping techniques. This research was funded in part by the Defense Advanced Research Projects Agency under Contract N00014-J-91-4041. Gaetano Borriello and Carl Ebeling were supported in part by NSF Presidential Young Investigator Awards, with matching funds provided by IBM Corporation and Sun Microsystems.

References

[1] Actel Corporation, “ACT Family Field Programmable Gate Array Data Book”, 1991.

[2] Algotronix Limited, “CAL1024 Preliminary Datasheet”, 1991.

[3] J. F. Beetem, “Simultaneous Placement and Routing of the Labyrinth Reconfigurable Logic Array”, International Workshop on Field-Programmable Logic and Applications, Oxford, 1991.

[4] W. Carter et al., “A User Programmable Reconfigurable Gate Array”, Proceedings of the IEEE Custom Integrated Circuits Conference, May 1986.

[5] Concurrent Logic, Inc., “CFA6006 Field Programmable Gate Array”, March 1991.

[6] K. A. El-Ayat, A. El-Gamal, R. Guo, J. Chang, R. K. H. Mak, F. Chiu, and E. Z. Hamdy, “A CMOS Electrically Configurable Gate Array”, IEEE Journal of Solid-State Circuits, Vol. 24, No. 3, June 1989, pp. 752-761.

[7] F. Furtek, Personal communication, October 1991.

[8] L. J. Guibas and F. M. Liang, “Systolic Stacks, Queues, and Counters”, Second MIT Conference on Advanced Research in VLSI, 1982.

[9] T. Kean, “Configurable Logic: A Dynamically Programmable Cellular Logic Architecture and its VLSI Implementation”, PhD Thesis, Dept. of Computer Science, University of Edinburgh, 1989.

[10] T. Kean, Personal communication, October 1991.

[11] R. J. Lipton and D. Lopresti, “A Systolic Array for Rapid String Comparison”, Chapel Hill Conference on VLSI, 1985.

[12] M. Schlag, P. K. Chan, and J. Kong, “Empirical Evaluation of Multilevel Logic Minimization Tools for a Field Programmable Gate Array Technology”, International Workshop on Field-Programmable Logic and Applications, Oxford, 1991.

[13] S. Singh, J. Rose, D. Lewis, K. Chung, and P. Chow, “Optimization of Field-Programmable Gate Array Logic Block Architecture for Speed”, Proceedings of the IEEE Custom Integrated Circuits Conference, May 1990.

[14] S. Trimberger, Personal communication, October 1991.

[15] J. Vuillamy, Z. G. Vranesic, and J. Rose, “Performance Evaluation and Enhancement of FPGAs”, International Workshop on Field-Programmable Logic and Applications, Oxford, September 1991.