

# **TRIPTYCH: A New FPGA Architecture**

**Carl Ebeling, Gaetano Borriello, Scott A. Hauck,  
David Song, Elizabeth A. Walkup**

Department of Computer Science and Engineering  
University of Washington  
Seattle, WA 98195

## ***Abstract***

*Existing FPGA architectures can be classified along two dimensions: reprogrammable vs. one-time programmable and general-purpose vs. domain specific. The most challenging class of FPGA architectures to design is the reprogrammable, general-purpose FPGA, of which Xilinx is the most well-known example. In this paper we describe Triptych, a new FPGA architecture that addresses two problems of current reprogrammable FPGAs: the large delays incurred in composing large functions and the strict division between routing and logic resources. Our studies indicate that Triptych is more area-efficient than current architectures and has comparable delay characteristics for a large range of circuits that include both data-path elements and control logic.*

## **INTRODUCTION**

The most common approach to field-programmable gate array architectures is to dedicate a portion of the total chip area to logic functions and the remainder to interconnection resources. The logic functions may be fixed or programmable, while the routing is usually highly programmable to ensure that a large percentage of designs are routable. The flexibility of the interconnection network is limited by two factors: the number of configuration points (bits or fuses) that can be accommodated on chip and the speed requirements of the signals routed through the network (more switches or fuses on a signal path imply slower wires) (Rose 1991).

FPGAs can be programmed using a reprogrammable memory-based scheme or a one-time programmable fuse technology. Xilinx is the most well-known example of a reprogrammable FPGA (Carter 1986). It has logic blocks that can perform arbitrary functions of five inputs. The routing resources are arranged in an orthogonal grid around the function blocks and occupy approximately 90% of the chip area. Approximately 300 function blocks can be placed in a single device, the number being limited by the extra routing resources additional function

cells would require. In a chip with 320 cells, 64,160 programming bits are required, or approximately 200 bits per cell (Xilinx 1991).

Among one-time programmable FPGAs, Actel is the most common (El-Ayat 1989). Actel arranges a basic cell in rows similar to an arrangement of standard cells in a semi-custom integrated circuit. The cell functionality is fixed, with the logic function determined by where inputs are connected to the cell (typical usage is as a 3-input function). The interconnection resources are also similar to the standard cell style with wires running in segmented channels between the rows of cells and orthogonally across the cells to provide routing in all four directions. The logic cells account for 10-15% of the chip area and 750,000 bits are required to program a typical chip of 1200 cells (Actel 1991). The number of routing tracks limits the total number of cells that can be placed with reasonable routability on a single chip.

A more recent entry in the FPGA arena is the Apple Labyrinth architecture (Furtek 1990). Rather than dedicating chip area to either computation or interconnect, the Labyrinth FPGA tiles the chip with identical small cells that can perform either 2-input functions or routing, depending on the user-specified programming in the 4 bits per cell. Each cell is connected only to its four nearest neighbors. The design is intended for pipelined bit-serial applications, because the delays incurred in routing through many cells severely limit the cycle time.

In this paper, we present an alternative structure for reprogrammable FPGAs that blends logic and routing resources more closely than most other FPGAs. That is, each routing and logic block (RLB) in the Triptych array can be used both to compute a logic function and 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. Circuits use varying numbers of RLBs for routing depending on how much their structure diverges from the Triptych structure.

We decided to undertake the detailed design and implementation of Triptych in the graduate VLSI implementation course (CSE568) in the winter quarter of 1991. The problem was an ideal class project because there was only a small collection of basic cells to design, and students could work on implementation and mapping issues in parallel. This paper describes the basic Triptych architecture and the experience we gained implementing it and mapping circuits to it. The two sections following this introduction describe the architecture in detail and the issues and design choices encountered during implementation. The next section provides a first look at how the architecture can be used and how it compares to others, as well as some ideas for automatic mapping. Finally, we conclude with remarks about both the architecture and the educational experience.

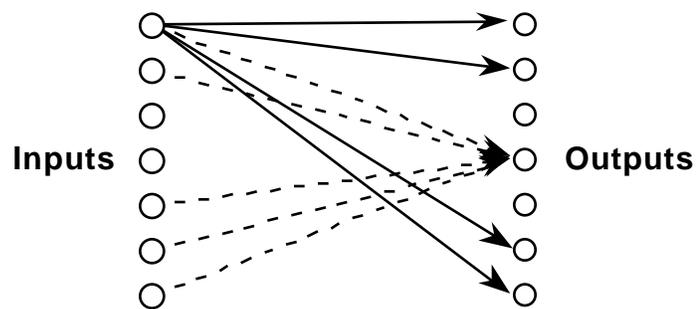
## **TRIPTYCH**

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.

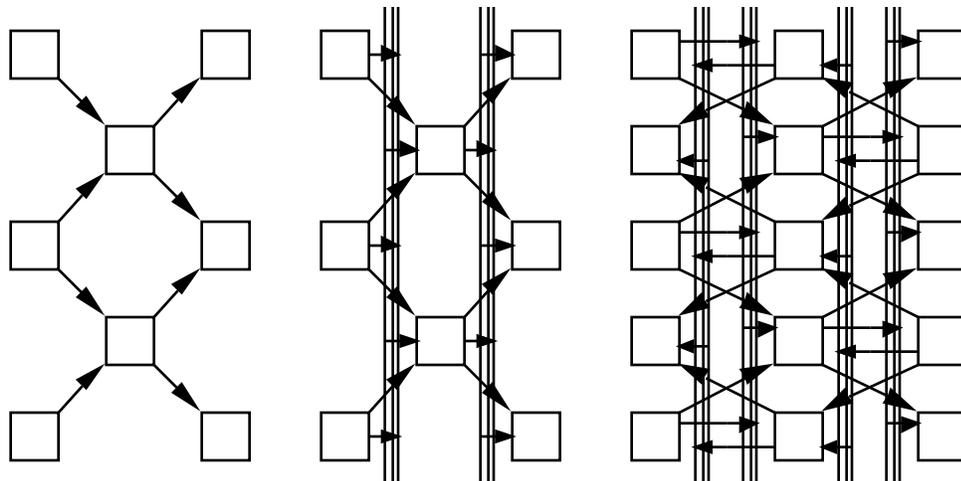
Figure 1 shows a high-level view of a typical multi-level combinational logic circuit. The flow is shown as unidirectional, from inputs to outputs. From the point of view of each input, the data flow forms a fanout tree (shown with solid arrows) to those outputs that the input affects. From the point 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 into columns, with each RLB having a short, hard-wired connection to its nearest neighbors in adjacent columns (see Figure 2).

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 Xilinx and Actel FPGAs. However, we claim that this array matches the form of a large class of circuits and that mapping will produce routable implementations.



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

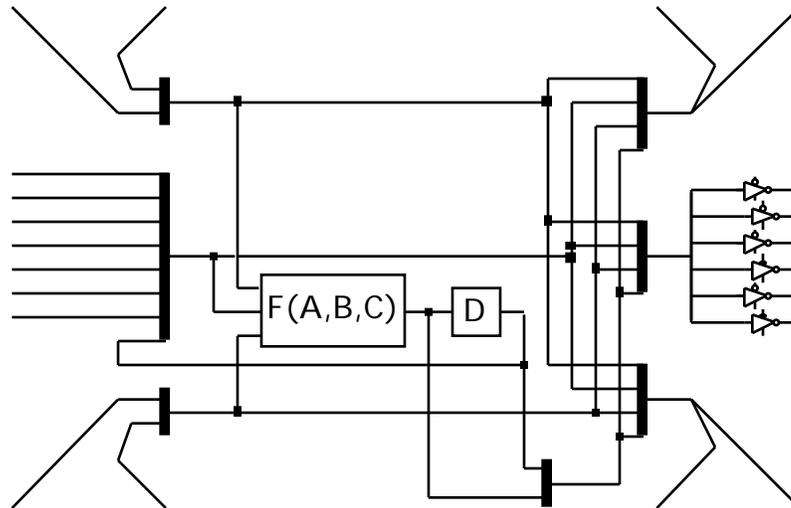


**Figure 2** The overall structure of the Triptych FPGA shown in a progression of steps highlighting more and more features. The basic fanin/fanout structure on the left is augmented with segmented routing channels that make a third input and a third output available to the RLBs. The structure on the 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.

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 (Rose 1990). 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.

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 at one side of this structure 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 Rose suggests in (Singh *et al* 1990), delay can be minimized by using mostly direct, hard-wired connections for the critical path. Triptych implementations do not strive for 100% logic utilization. Many RLBs will be used to provide routing, either to fanout a signal or to pass it forward to the next level. Sometimes a mapping will leave some cells unused to achieve a routable placement of nodes. Examples are provided below.



**Figure 3.** Triptych 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.

### RLB structure

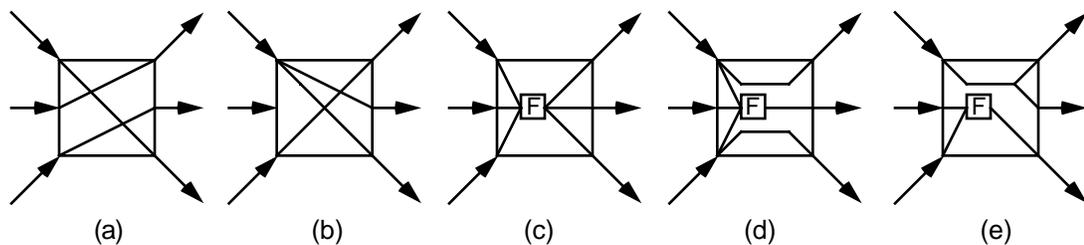
A logical schematic of the basic Triptych RLB is show in Figure 3. As can be seen, the cell is designed to handle both function calculation and signal routing simultaneously (hence the name

*routing and logic block*, RLB). It takes input from three sources and feeds them into a function block capable of computing any function of the 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 last feature 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.

### 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 just 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 generalized interconnect for moving signals horizontally. Instead, there is a heavy reliance on unused RLBs and unused portions of RLBs to perform these routing tasks.

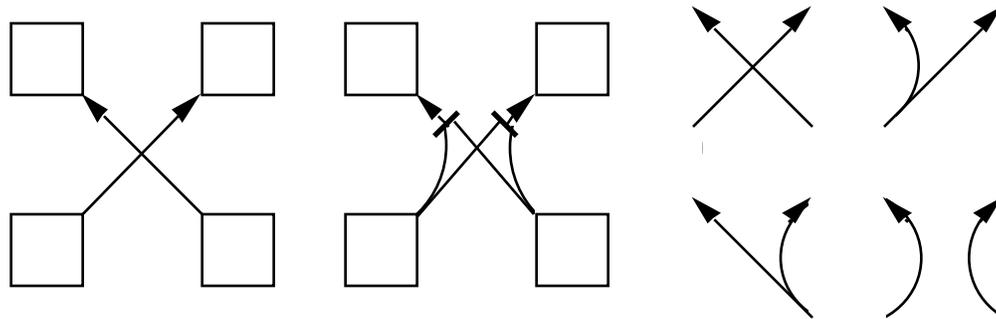


**Figure 4** Five typical uses of Triptych RLBs.

### Interconnection

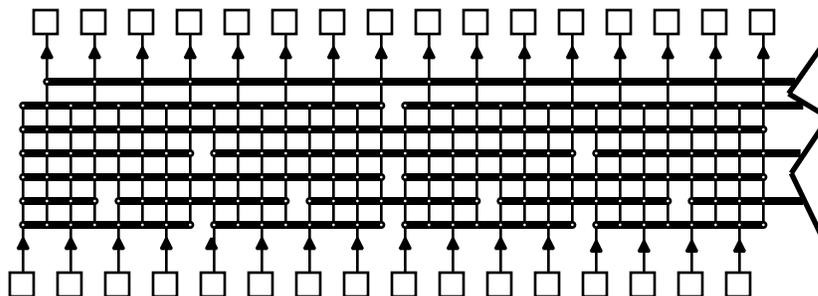
The Triptych RLBs are connected by three separate interconnection schemes. The first is for horizontal interconnect and is accomplished through the RLBs as described above. The second

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 RLBs to send outputs to the RLBs immediately above and below them, which flow in the opposite direction, and to the two RLBs in the same position in the next column, which flow in the same 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.



**Figure 5** Schematic view of a pair of diagonals and the routing combinations they allow (implemented by a multiplexer at each diagonal input). The diagonals connect an RLB’s outputs to the RLB’s four nearest neighbors: two directly above and below in the same column and the two in the same positions in the next column.

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 a slower path, 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 because the vertical channel for the left and right edges of the original six by six square will be unusable on the cylinder, however it can be a useful abstraction for hand mapping. In fact, the Triptych chip is an array of 64x8 RLBs, yielding a cylinder of 32x16.

### 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 selectively disabling RLB output drivers during programming. The memory cells are connected by a scan path running throughout the chip, allowing it to be programmed by cycling data through the bits.

The scan path used for programming is also attached to the RLB’s master/slave D-latches. This not only allows the chip to start in any arbitrary combination of latch states, but it also allows the contents of the latches to be shifted out after the chip has run an arbitrary number of cycles to facilitate debugging. Also, if the scan path input is connected to the output, a programmed circuit can be stopped at any point, the contents of the D-latches analyzed, and the circuit resumed at the previous starting point.

### Vital statistics

The speed of a path in a Triptych RLB 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  $13.9 \pm 0.6$  nanoseconds ( $4 \times 1.6 + 2 \times 2.2 + 3.1 \pm 0.6 = 13.9 \pm 0.6$ ). 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” described below.

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

---

Resource Used	Delay
---------------	-------

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

**Table 2** Estimated space and memory utilization per RLB of various features. (Note: percentage of RLB area includes area for memory cells.)

	Percentage of RLB area	Number of bits
Vertical Segmented Channels	54%	9
Diagonals	6%	2
Internal Routing & Multiplexers	23%	7
Function Block & D-Latch	17%	8
Total		26

Table 2 describes the relative sizes of the main components of the Triptych RLBs. The features measured are “Vertical Segmented Channels”, including the line drivers and line readers; “Diagonals” which includes similar features as the “Vertical Segmented Channels”; “Internal Routing & Multiplexers” which includes the three 4:1 multiplexers for selecting the signal to send to each output as well as the 2:1 multiplexer that chooses between the latched and unlatched function block output; and “Function Block & D-Latch”. Note that each category not only includes the area needed for the given functionality, but also the area necessary to store the configuration bits (which contributes 1% of RLB area per bit, since 26 programming bits take up 26% of the RLB area).

Probably the most important observation to be made from the table is that 83% of RLB area is devoted to routing of one form or another, with the actual function calculation only occupying 17%. Note that this number is fairly small compared to other reprogrammable FPGAs since a full 30% of the space for “Vertical Segmented Channels” is actually the inverters and tri-state buffers used to drive the channel wires, with another 6% in associated memory cells. These features would be included in the function blocks of other FPGAs.

## DESIGN ISSUES

The design and implementation of the Triptych FPGA brought up several issues that we feel are of general interest. These are discussed in the following subsections.

### Regularity

A goal in the design of the Triptych cell and interconnect was to achieve as regular a structure as possible. This was done because technology mapping is difficult, and any irregularities only complicate the issue. For example, the Triptych function block can compute any function of three inputs, as opposed to designs such as the Actel FPGA where only a subset of the possible functions can actually be realized in a cell. Also, an arbitrary function block removes

the worry of what to do for inversions, since an inverter can easily be factored into any or all of the inputs and the output of the function block.

The interconnect scheme follows the same philosophy; the only deviation is caused by the edges of the array. A more creative structure with the interconnect optimized differently (e.g., as a butterfly) could have been implemented, but we feel that the complications added to the technology mapping stage would negate any potential gains.

### **Independence of paths and logical effort**

The Triptych RLB is mostly composed of multiplexers and bus drivers. Early on, the decision had to be made whether to implement most of the multiplexers with switch or gate logic. Our original choice was to do most of the RLB in switch logic and only insert inverters where necessary to drive loads. We have since decided this was a mistake and have redesigned the circuit almost completely in gate logic. The main reason for this is something we call “independence of paths”. The idea is that the routing of two different paths should affect the timing of each other as little as possible. This point is much the same as the one above, except that where the above rule dealt with the logical specification of the RLB and the interconnect, this deals with how the RLBs are actually implemented. Take for example the case where a single RLB output fans out to several inputs. If the RLBs were implemented in switch logic, with pass-gates taking inputs off the vertical channels, a signal would propagate more slowly if several RLBs were reading the same interconnect line than if each had its own. Thus, a technology mapper designed to optimize for speed would have to make sure that the critical path always used its own interconnect line. There are several other places where this effect can manifest itself, such as routing an input to an output (which slows down the function calculation) and splitting a signal to two or more outputs (which slows down both signals). This rule exists not just to make technology mapping easier; by making paths independent, it is also much easier to optimize the RLB channel wire drivers.

The Triptych chip was originally laid out by a handful of graduate students with little or no previous integrated circuit design experience. The project was carried on by one of these graduate students (Scott Hauck), who did a completely new layout aided significantly by the model of logical effort (Sutherland and Sproull 1991) which assists in the proper sizing of transistors and insertion of buffers to optimize speed. Although we have no firm numbers determining how much better the second design is than the first, we feel that logical effort can help novice designers develop faster circuits.

### **Routing flexibility**

There are several unsettled issues in the design of the Triptych routing network. First and foremost is the sharing of tracks in the vertical segmented channels. By sharing tracks between RLBs flowing in opposite directions, we could implement a more flexible feedback capability than is possible using only the diagonals. Currently, the array has seven tracks for each direction, for a total of 14 in each segmented channel. One alternative is to have 5 tracks for each direction with another 2 shared for a total of 12. It is difficult to tell just how much sharing is needed. The shared tracks would have more drivers and receivers than they would if they were not shared and thus be slower. More experience with manual and automatic mapping will be needed before this issue is resolved.

Another issue relates to the D-latch loopback capability, which replaces the channel wire input in RLBs that use the loopback. Most likely, this input will be needed for an input and conflict with the use of the loopback. The loopback exists because it was extremely cheap to include. The alternative is to route the output of the D-latch around through the RLB above or below on diagonals. Whether this is sufficient or a form of internal loopback is required (possibly coming in on diagonal inputs) will also be determined by experimentation.

Finally, we still must resolve how the Triptych array will be connected to the chip's I/O pins. Inputs can reach the array by entering RLBs on either vertical edge or by entering the vertical channels from the top and bottom. We expect to provide general input/output pads on all four sides with routing channels along the top and bottom of the array. Connections with either vertical edge will be more direct to provide a fast path into and out of the array for data-path applications.

### **Programming hazards**

In FPGA design it is very tempting to ignore the programming phase, except to demand the most compact implementation of the programming bits as possible. However, this can lead to some serious problems. In an FPGA, there are certain assumptions made about the programming that are not enforced by the hardware. For example, it is assumed that at most one RLB drives any specific channel wire. This is automatically enforced in intra-cell routing and diagonals by virtue of multiplexer logic. In the case of channel wires, special hardware is required to enforce this constraint, with a very high overhead. In the Triptych FPGA, we simply assume that the software performing the mapping deals with this problem and that no configuration loaded will violate this constraint. During programming, however, bits stream through the programming memory, violating this programming assumption. This leads to short-circuits in the chip and possible damage. One solution is to adopt a bit-addressable scheme for the programming memory rather than a scan-path, but this is quite expensive due to the extra routing and decode logic required. Instead, we use the same signal that enables the scan-path to disable all channel wire drivers. Thus, while the chip is being programmed no drivers are active, thereby eliminating the problem. This costs approximately an extra 3% in chip area for the transistors and wiring required.

## **USING TRIPTYCH**

In this section, we present several circuits that we have mapped by hand 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.

### **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 has 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.

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

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.

### **State machine example**

Figure 8 shows the factored logic equations and 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. Note that this example would have been easier to map if the vertical wires could be used to route within a column anywhere in the chip, not just at the borders, and in fact such an extension is under consideration. This is about as compact a Triptych layout as can be expected for a random logic function.

```

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

```

**Figure 8** Factored equations and Triptych realization of the traffic light controller.

### Lyon bit-serial multiplier

Although our experience shows that Triptych can be used to implement a wide range of circuits, its locally connected structure makes it especially good for repetitive arrays like bit-serial arithmetic circuits. The Triptych structure has some of the same features (e.g., nearest neighbor connections) as the Labyrinth FPGA which was targeted to bit-serial and pipelined/systolic circuits. We have chosen the Lyon bit-serial multiplier cell (Lyon 1981) shown in Figure 9 as a representative circuit from this class. A full n-bit multiplier comprises n copies of this cell, and signal processing circuits typically make use of several of these multipliers, containing many individual cells.

**Figure 9** Design of a single Lyon bit-serial multiplier cell.

**Figure 10** Layout for the Lyon bit-serial multiplier cell.

This multiplier cell presents the same classic layout problem as that faced by VLSI cell designers. The cells need to tile horizontally so that inputs match outputs and vertically so that little space is wasted between adjacent multiplier cells (see Figure 10). In this case, however, there is an extra dimension since a string of multiplier cells will wrap around the chip on the opposite direction of RLBs. Since there is one RLB that is used from the opposite direction (position E-2), the layout must provide a “hole” into which this RLB can fit (position B-4). Note that these two logical RLBs can share a single physical RLB since they use independent paths through the RLB. The cost of this multiplier cell design is 12.5 RLBs which is not much more than the smallest conceivable design, which costs 11 RLBs. The 0.5 RLB results from the sharing of one RLB (positions A-3 and F-4) between two vertically adjacent multiplier cells.

### **Measurement and comparison**

Although our experience with mapping circuits to Triptych is thus far very limited since automated placement and routing are still being developed, we have some preliminary measurements of the cost of Triptych implementations relative to Labyrinth and Xilinx. Since

the area cost is measured for each FPGA type in terms of the number of logic blocks used for that technology, we must first normalize the cost of the different FPGA logic blocks to be able to compare the different FPGAs. Although such relative figures are difficult to come by, we have combined a relative size estimate based on die size and number of logic blocks, along with the relative number of program bits to arrive at the following relative cost figures. Using the cost of the Labyrinth logic block as the normalized unit cost, we estimate that the cost of a Triptych RLB is about 4-5 (4.5) units and that of a Xilinx CLB (configurable logic block) is about 20-25 (22) units. This places the Triptych logic cell squarely in the middle between the very cheap Labyrinth cell and the relatively expensive Xilinx cell.

Table 3 gives the approximate cost of implementing a number of circuits using all three FPGAs, both in terms of each technology’s logic blocks and in normalized cost as defined above. We believe these figures indicate that Triptych is a promising architecture for a range of different circuits. These results are of course very preliminary and many more experiments must be done with other circuits and using automatic place and route tools.

Circuit	Labyrinth # blocks	normalized cost	Xilinx # blocks	normalized cost	Triptych # blocks	normalized cost
Multiplier	54	54	5	110	12.5	56
Traffic	280	280	6	132	24	108
s208	N/A	N/A	26	572	61	275

**Table 3.** Results of mapping three examples: the Lyon bit-serial multiplier; a traffic light controller; and ISCAS benchmark s208 the Labyrinth, Xilinx and Triptych.

### Issues in mapping to Triptych

We have successfully mapped a number of regular structures and small control circuits to the Triptych architecture, and we are currently working on CAD tools that will automatically perform the mapping for arbitrary circuits. As with other FPGAs, the process of mapping a circuit onto Triptych can be considered to consist of three steps:

- covering: forming a circuit graph containing function nodes with at most three inputs,
- placement: assigning these function nodes to cell locations on Triptych, and
- routing: making the connections in the graph through the available routing on Triptych.

If the circuit to be mapped has a regular structure, as is encountered in domain-specific applications such as digital signal processing, an initial pattern for the repeating portion may be derived by hand. Circuits without regular structure, or “random logic”, must rely on heuristic-based automatic placement and routing methods similar to those used by other FPGAs. However, because Triptych’s routing resources are highly constrained, placement and routing must be more closely integrated than they are in other FPGAs.

For the covering portion of mapping to Triptych, we assume that a tool such as *chortle* or *mis-pga* is available to express the original circuit as a graph of elementary gates and then cover the graph’s fanout-free trees with collections of three-input RLBs (Francis 1991, Murgai 1990). It should be noted, however, that a covering which minimizes the total number of RLBs may not be optimal when placement and routing are taken into consideration. For example, if after placement two of the inputs to a three-input RLB naturally both occur at a

single location distant from that RLB, it is usually advantageous to split the RLB into two two-input functions. If this is possible, we can route one less signal across the large distance. Clearly, such situations are not unique to Triptych. However, we particularly wish to avoid routing extra signals horizontally whenever it can be avoided. Otherwise, RLBs become congested with signals they do not use. Such optimizations are difficult to predict at cover time and thus need to be attempted during routing.

Because Triptych's routing resources are limited and fairly tightly constrained, we believe it is necessary to keep placement and routing well integrated. Evaluating possible placements with simple measures of routing length can lead to placements whose congestion make routing nearly impossible. Currently, we are exploring iterative improvement methods for placement which will assign an RLB only into locations which are adjacent to enough free tracks to route the RLB's inputs and outputs. Thus, we avoid congestion at a local level whenever we place an RLB.

A complicating factor is that Triptych's distance metric is non-symmetric. All pairs of RLBs that face in the same direction, except those in the same column, have a distance from the first's output to the second's input different than that of the second's output to the first's input. Also, vertically adjacent blocks have the same routing distance as diagonally adjacent blocks. For these reasons, routing distance is not well represented by the x-y coordinates given to the RLBs. Work is ongoing to develop an integrated force-directed placement procedure, a Triptych-specific distance measure, and the congestion avoiding method mentioned above.

## CONCLUSIONS

The new FPGA architecture presented in this paper was motivated by three needs: permitting the realization of delay-critical circuits; including data-path and control elements in the same array; and minimizing the space devoted to routing resources. 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 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.

Pedagogically, the design of a field-programmable gate array made an excellent class project. It exposed our students to a large vertical slice of the design problem stretching from electrical details to technology mapping issues. They were able to experience many of the issues and tradeoffs that must be resolved in both integrated circuit and logic design. Furthermore, the design and layout were ideal for a class project because the work was easily partitioned and only a small number of cells needed to be considered. In this respect, the Triptych effort was a resounding success and has motivated several of the non-VLSI, non-CAD oriented students to continue to look into VLSI issues.

In summary, we learned much from the design experience and believe we have a viable new FPGA architecture for circuits where either minimization of delay is of critical importance and/or data-path elements must be included with 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 David Hubbell, Daniel Kerns, Alexander Klaiber, Dylan McNamee, J. Scott Penberthy, Radhika Thekkath, and especially Christopher Hébert for participating in CSE568 and contributing to the early design phases of the Triptych layout and mapping techniques. Thanks also to Rick Hood for the Labyrinth measurements in Table 3 and help developing the relative cost measures. 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. Elizabeth Walkup holds an NSF Graduate Fellowship.

## REFERENCES

- Actel Corporation, "ACT Family Field Programmable Gate Array Data Book", 1991.
- W. Carter et al., "A User Programmable Reconfigurable Gate Array", Proceedings of the IEEE Custom Integrated Circuits Conference, May 1986.
- 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.
- R. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs", Proceedings of the 28th Design Automation Conference, June 1991.
- F. Furtek, G. Stone and I. W. Jones, "Labyrinth: A Homogeneous Computational Medium", Proceedings of the IEEE Custom Integrated Circuits Conference, May 1990.
- R. F. Lyon, "A Bit-Serial VLSI Architectural Methodology for Signal Processing", Proceedings of VLSI'81, August 1981.
- R. Murgai, Y. Nishizaki, N. Shenoy, R. Brayton, A. Sanggiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays", Proceedings of the 27th Design Automation Conference, June 1990.
- J. Rose and S. Brown, "Flexibility of Interconnection Structures for Field-Programmable Gate Arrays", IEEE Journal of Solid-State Circuits, Vol. 26, No. 3, March 1991, pp. 277-282.
- J. Rose, R. J. Francis, D. Lewis, and P. 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, October 1990, pp. 1217-1225.
- 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.
- I. Sutherland and R. Sproull, "Logical Effort: Designing for Speed on the Back of an Envelope", keynote address at the University of California at Santa Cruz Advanced Research Conference in VLSI, March 1991.
- Xilinx, Inc., "The Programmable Gate Array Data Book", 1991.