The Memory/Logic Interface in FPGAs with Large Embedded Memory Arrays

Steven J.E. Wilton, Jonathan Rose, Zvonko G. Vranesic

Abstract— **As the capacities of field-programmable gate arrays (FPGAs) grow, they will be used to implement much larger circuits than ever before. These larger circuits often require significant amounts of storage. In order to address these storage requirements, FPGAs with large embedded memory arrays are now being developed by several vendors. One of the crucial components of an FPGA with on-chip memory is the routing structure between the memory arrays and the logic resources. If this memory/logic interface is not flexible enough, many circuits will be unroutable, while if it is too flexible, it will be slower and consume more chip area than is necessary. In this paper, we show that an interconnect in which each memory pin can connect to between 4 and 7 logic routing tracks is best in terms of both area and speed. We also show that by adding switches to support nets that connect multiple memory arrays, we can reduce the memory access time by up to 25% and improve the routability slightly.**

Keywords— **FPGAs, Reconfigurable Systems, Embedded Memory Arrays**

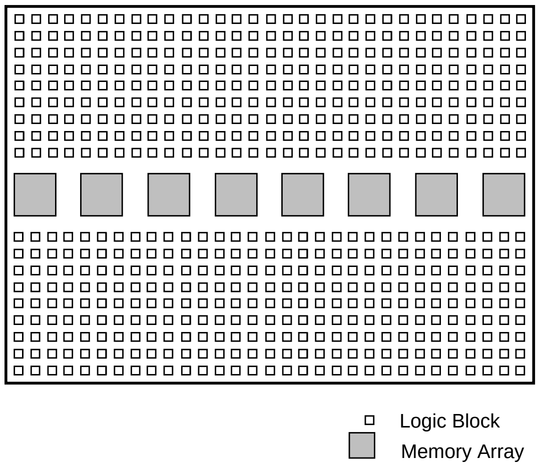


Fig. 1. FPGA with Embedded Memory

I. Introduction

RECENT years have seen dramatic improvements in FPGA capacities and speeds. Such improvements are changing the way FPGAs are used. In the past, these devices have been primarily used to implement small logic subcircuits (often the "glue-logic" portions of larger systems), but as FPGAs get larger, they are being used to implement much larger circuits and even entire systems. One of the most important differences between these large systems and the smaller logic subcircuits is that the systems often require significant amounts of storage. Architectural support for the efficient implementation of memory in next-generation FPGAs, therefore, is crucial.

In [1], [2], a configurable memory architecture was described which is flexible enough to implement a wide variety of storage requirements. This configurable architecture can be used in a stand-alone device or can be embedded onto an FPGA to support on-chip memory. Implementing memory on-chip has a number of advantages over off-chip memory: it reduces the system cost by decreasing the number of chips required to fully implement a system, it often allows for faster clock rates since external pins (and board-level traces) need not be driven with each memory access, and it frees I/O pins that would otherwise be devoted to address and data connections. These advantages have led several FPGA vendors to produce FPGAs with significant amounts of on-chip memory [3], [4], [5], [6], [7].

One of the challenges when embedding memory arrays onto an FPGA is to provide enough interconnect between the memory arrays and the logic resources. The design of a good memory/logic interface is critical. Since memory access time is often the performance bottleneck in many systems, it is crucial that the memory/logic interface provides a flexible high-speed link between logic and memory. If the interface is not flexible enough, many circuits will be unroutable, while if it is too flexible, it will be slower and consume more chip area than is necessary. This paper focuses on the design of a good memory/logic interface. We concentrate on two issues: how flexible the interconnect must be, and how the interconnect can be enhanced by providing efficient connections between the memory arrays themselves.

The paper is organized as follows. In the next section, we describe the baseline FPGA architecture for our experiments, and quantify the flexibility of the memory/logic interconnect structure. Section III then determines how flexible the interconnect must be, taking into account the overall FPGA routability, area, and delay. Finally, Section IV shows that by adding efficient paths between memory arrays, the FPGA routability and delay can be improved significantly.

Early versions of some of these results appear in [8] and [9]. More details regarding much of the material is available in [10].

II. Baseline Architecture

The FPGA considered in this paper consists of distinct logic and memory resources, as illustrated in Fig. 1. The memory resources consist of a set of identical arrays that can be combined to implement user memory configurations, similar to [1]. The number of bits in each array is fixed,

S. Wilton is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada. E-mail: stevew@ee.ubc.ca. J. Rose and Z. Vranesic are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada. E-mail: {jayar,zvonko}@eecg.toronto.edu .

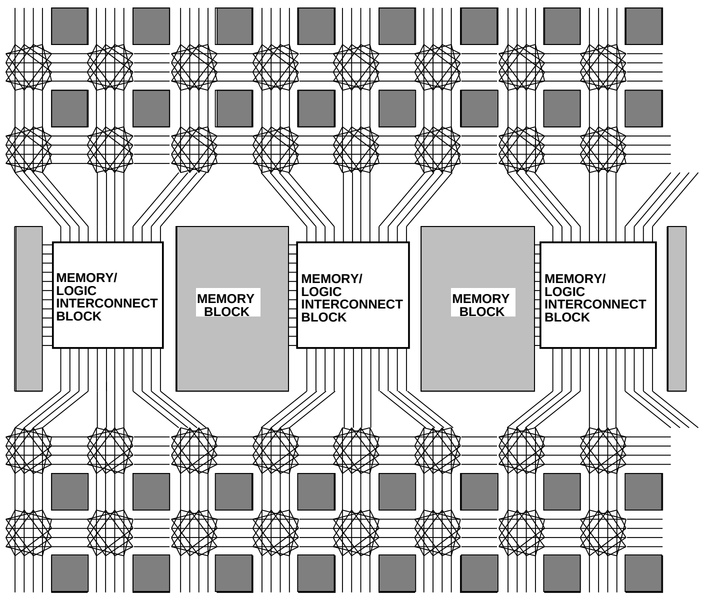

MEMORY/ LOGIC INTERCONNECT BLOCK
MEMORY BLOCK
MEMORY/ LOGIC INTERCONNECT BLOCK
MEMORY BLOCK
MEMORY/ LOGIC INTERCONNECT BLOCK

Fig. 2. Memory/logic interconnect architecture.

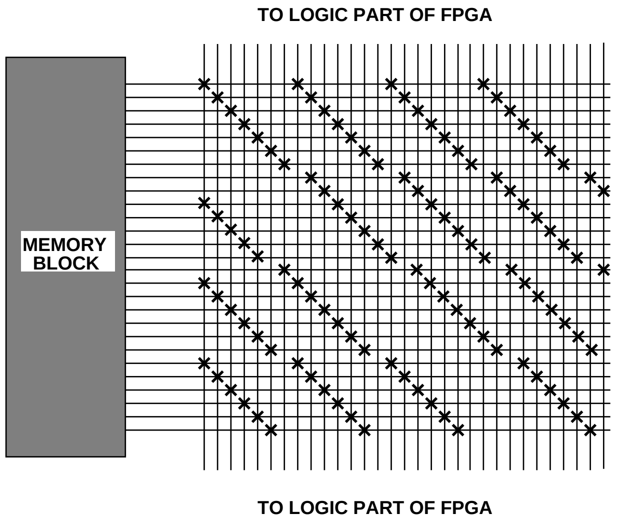


Fig. 3. Memory/logic interconnect block.

but the aspect ratio can be configured by the user. In the results presented in this paper, we assume that each array contains 2-Kbits of storage, and has a configurable data width of 1, 2, 4, or 8 (similar to the Altera 10K series CPLDs [3] and [2]). We further assume that the arrays are positioned in a single row across the width of the chip, as is the case in Altera 10K and Actel SPGA parts [4], [5]. Positioning the arrays in this way allows for easy connections to logic, as well as easy connections between memory arrays that are combined to implement large user memories. Unlike [1], here we assume that each memory array has separate input and output data ports (as well as an address port).

The logic resources of the FPGA are assumed to consist of five-input lookup tables, interconnected using symmetric horizontal and vertical channels similar to the Xilinx and Lucent ORCA FPGAs [6], [7]. At the intersection of every horizontal and vertical channel is a switch block; the switch block offers each incoming wire three possible connections (i.e. $F_s = 3$ in the terminology of [11]). Each logic block pin can be connected to W tracks, where W is the number of tracks in each channel. We have assumed that all segments are of length 1; that is, segments only connect neighbouring switch blocks. Each pin of each lookup table can be connected to two channels; within each channel, each pin can be connected to all W tracks.

Fig. 2 shows the interconnect structure between the logic and memory. Each memory block is connected to the logic routing through a *memory/logic interconnect block*. An example memory/logic interconnect block is shown in Fig. 3. The vertical tracks in Fig. 3 are connected to the upper and lower halves of the logic array and can be programmably connected to the memory pins. We define the flexibility of this block, F_m, as the number of vertical tracks to which each memory pin can be connected. In Fig. 3, each cross represents a programmable connection; thus, in this example, $F_m = 4$. The minimum value of F_m is 1, while the maximum is V where V is the number of vertical wires incident to the memory/logic interconnect block.

Another parameter affecting the FPGA routability is the number of logic blocks per memory block in the horizontal dimension. Section III-C discusses this further.

III. Memory/Logic Interconnect Flexibility

As described above, the flexibility of the interconnect structure is quantified by the parameter F_m which indicates the number of tracks to which each memory pin can be connected. In this section we vary F_m and examine the effects on the overall FPGA routability, area, and delay.

Note that we are concerned with the area and speed of the entire FPGA, not just the memory/logic interconnect region. Clearly, the lower F_m, the smaller the memory/logic interconnect will be because fewer programming bits and pass transistors are needed. Decreasing F_m, however, places additional demands on the rest of the FPGA since it makes it more difficult to route nets between the logic and memory blocks. One way the designer of an FPGA can compensate for this reduction in routability is to add extra tracks to each channel across the entire FPGA. The area cost of these additional tracks must be considered when determining a good value of F_m.

A similar tradeoff exists between circuit speed and F_m. The lower F_m, the fewer switches there are on any path into and out of the memory blocks. Since switches add parasitic capacitance to the memory nets, the reduction of F_m shortens the memory access times. However, the lower-flexibility memory/logic interconnect blocks may result in circuitous routes (that are slower than direct routes) between logic and memory. The extra delay due to these circuitous routes must also be considered when determining a good value for F_m.

A. Experimental Methodology

We employ an experimental methodology in which benchmark circuits are "implemented" on candidate FPGA architectures using custom-written CAD tools. For each circuit and each architecture, we measure the minimum number of tracks required in each channel to completely route the circuit, and use this in an area model to estimate the area-efficiency of the architecture (a fairly standard method used when exploring the area effects of architectural choices [11], [12]). A detailed delay model is used to estimate the speed-efficiency of each architecture. The following subsections describe the source of the benchmark circuits and the CAD tools employed.

B. Benchmark Circuit Generation

The traditional method of placing and routing 10 to 20 benchmark circuits [11], [12] is not suitable for the analysis of configurable memory architectures. Since circuits typically have only a few memories each, hundreds of such examples may be required to properly exercise the architecture. Because it isn't feasible to gather that many benchmark circuits, our approach is to study the types of memory configurations found in systems, and then to develop a stochastic memory configuration generator based on that study.

It is crucial that the generated circuits are realistic. We ensure this by basing the generator on the results of a detailed circuit analysis. The next subsection briefly outlines the analysis, while the following subsection describes how we use the analysis results to ensure our stochastically generated circuits are realistic. A full explanation of both the analysis and generation is given in [10].

B.1 Structural Analysis of Circuits with Memory

This analysis is based on 171 circuits containing a total of 268 user memories. Data regarding these circuits was obtained from several sources: recent conference proceedings, recent journal articles, local designers at the University of Toronto, a major telecommunications company, and a customer study conducted by Altera [13]. Although we were unable to obtain netlists for the circuits, we could gather several key memory parameters.

As an example of the data gathered during the analysis, Fig. 4 shows the distribution of the number of memories in our sample circuits. As the graph shows, most circuits require only a small number of memories. It is important to note that the horizontal axis is the number of *user* memories; many of these user memories will require more than one physical array. We have also examined the widths and depths of the memories used in our sample circuits. Widths between 8 and 32 were common, while the range of depths varied considerably. Of particular interest was that approximately 70% of the dimensions were powers-of-two. The uses of memories (RAMs vs. ROMs and single- vs. dual-port) were also examined. Further data is presented in [10].

We have also examined how the user memories are connected to the logic parts of circuits. We have observed that memories tend to form "tightly connected" *clusters*, where is a cluster is a group of memories that are connected to common data input or data output subcircuits. Figure 5 shows the number of clusters in our sample circuits (we could only gather this information from 31 of our circuits)

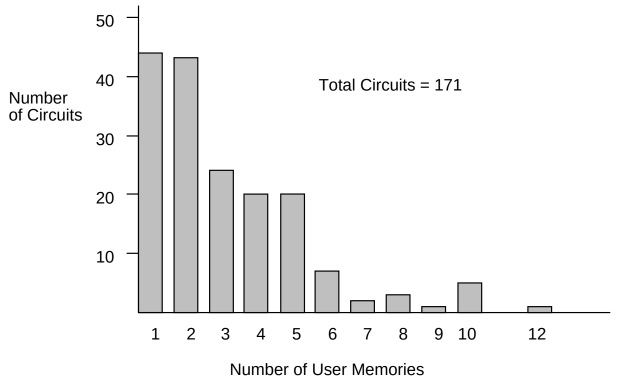


Fig. 4. Distributions of number of user memories in our sample circuits.

and the sizes of these clusters. We also examined common interconnect patterns between the memory and logic within clusters. Figure 6 summarizes the common patterns we observed. In Figure 6(a), each memory is driven by a seperate logic subcircuit. In Figure 6(b), each logic subcircuit is connected to seperate bits in each memory. Finally, in Figure 6(c), a common bus is used to connect the memory and logic. The details of the analysis results are presented in [10].

B.2 Circuit Generation

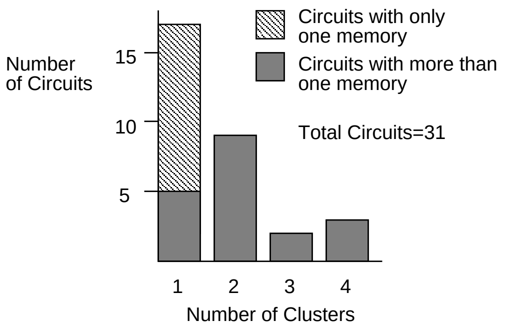
In order to ensure that the circuits from the circuit generator are realistic, we closely based the generation on the results of the circuit analysis. The distributions gathered during the analysis were used to chose memory configurations, to partition the memories into clusters, and to connect logic subcircuits to the memories. The logic subcircuits themselves where chosen randomly from a collection of 38 MCNC circuits [14], and were optimized using SIS [15] and technology-mapped to 5-input lookup tables using FlowMap [16]. The actual construction of the circuits is non-trivial and is described in detail in [10].

In the experiments described in this paper, we consider architectures with two, four, eight and sixteen 2-Kbit memory arrays. For each of these four FPGA sizes, we generate a separate set of 100 benchmark circuits; the user memories in each circuit use between 75% and 100% of the available bits in the target architecture. Table I shows statistics for the four sets of benchmark circuits.

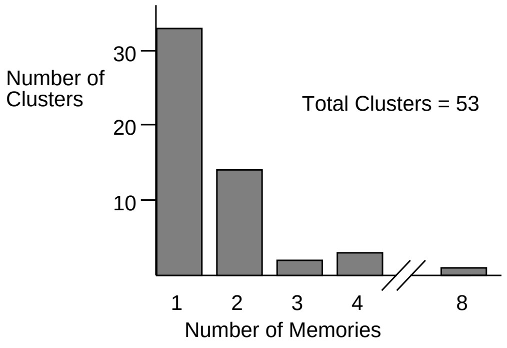
C. Implementation Tools

Each benchmark circuit is "implemented" in each FPGA using custom-built CAD tools [10]. First, each memory in the circuit must be implemented using one or more of the physical memory arrays. For example, a 4Kx2 user memory can be implemented using four 2-Kbit arrays each configured as a 2Kx1 memory with appropriate decoding. We call this process the *logical-to-physical mapping* and use an algorithm described in [10].

Next, the mapped circuits are then placed and routed on an appropriately-sized FPGA. The placer uses simulated-annealing to determine good locations for both the memory



a) number of clusters distribution



b) number of memories per cluster

Fig. 5. Cluster statistics.

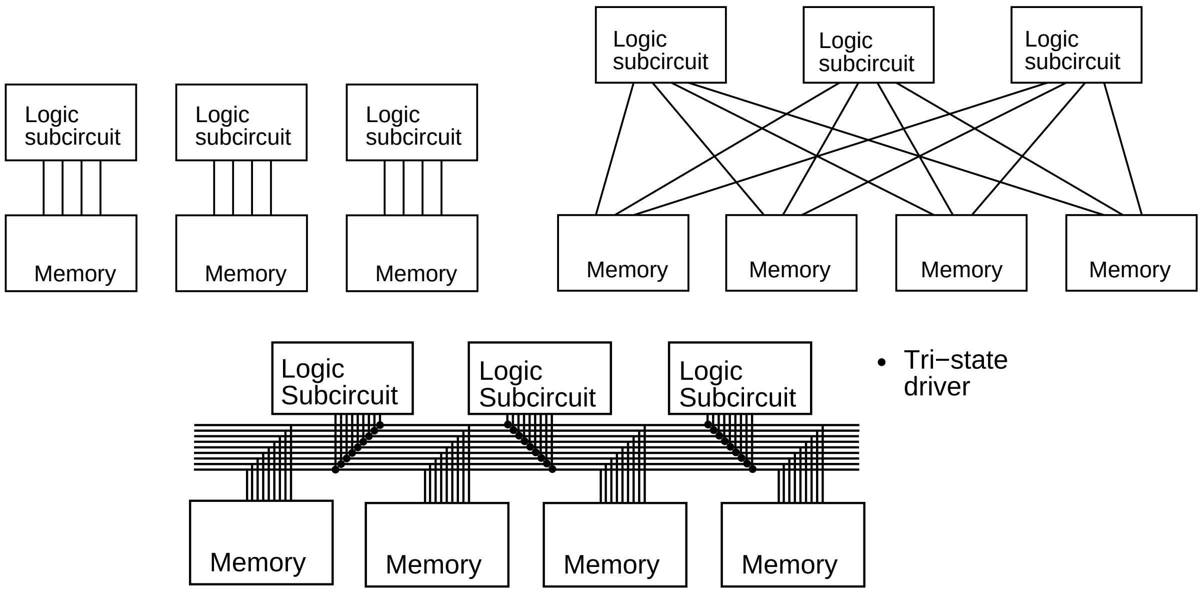


Fig. 6. Common interconnect patterns between memory and logic in our sample circuits.

Architecture to which circuits are targeted	Number	Logic Blocks		Memory Blocks		I/O Blocks		Nets	
		Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev
2 arrays	100	535	279	2	0	123	75	651	307
4 arrays	100	657	373	3.63	0.81	134	85.9	788	415
8 arrays	100	913	467	7.72	0.57	146	85.5	1069	497
16 arrays	100	849	500	15.2	1.33	154	102	1030	500

TABLE I
Circuit statistics.

and logic blocks simultaneously. The cost function minimized during the placement process is the sum of the dimensions of the bounding box surrounding each net; nets connecting to memory are treated the same as nets that only connect to logic.

The router uses a multi-pass maze routing algorithm. Initially, nets that connect to memory arrays are given higher priority (routed first). Between each iteration of the router, the nets are re-ordered such that the nets that could not be routed during one iteration are routed first during the next iteration. This is repeated 10 times; if a successful routing has not been found after 10 iterations, the circuit is deemed unroutable. The routing is repeated for different values of W (the number of tracks per channel) to determine the minimum W that gives a 100% routable solution. The router considers all input pins of a lookup table to be logically equivalent. Similarly, all address pins of a memory array are logically equivalent. Data pins are also considered equivalent with the constraint that a pin assignment for a specific bit in the data-out port fixes the corresponding assignment in the data-in port (and vice versa). More details on both the placer and router can be found in [10].

The size of the FPGA used in the place and route step depends on both the number of lookup tables and memory arrays required by the circuit. For a given number of arrays, we place the required number of logic blocks above and below the memory row, creating a roughly square chip. This will result in different values of R (the number of logic blocks per memory block in the horizontal dimension) for different circuits. If R is less than 3, we force R to be 3, resulting in a non-square aspect ratio. For some circuits, the number of inputs/outputs will determine the chip area; for these circuits, a square array with the required number of input/output blocks will be used. Table II shows the average values of R and the average aspect ratio (ratio of horizontal logic blocks to vertical logic blocks) for the benchmark circuits.

D. Memory/Logic Flexibility Results

The first set of results is for an FPGA with sixteen 2-Kbit memory arrays. The solid line in Fig. 7(a) shows the average number of tracks per channel required to route each benchmark circuit (averaged over all circuits) as a function of F_m. The right-most point on the horizontal axis represents the case when F_m equals its maximum value. The dotted lines show the track requirement plus and minus one standard deviation. As expected, as F_m increases, the average track requirement drops. Beyond $F_m = 4$, however, very little further drop is seen. The anomaly at $F_m = 3$ is a result of the particular switch pattern chosen for each memory/logic interconnect block [10].

The track requirements for all four FPGA sizes are shown in Fig. 7(b). Notice that the results for the smaller FPGAs are less sensitive to F_m that those for the larger architectures. In fact, for the 2 and 4 array FPGAs, an F_m of 1 or 2 provides sufficient flexibility. To understand why the smaller architectures can tolerate such low values of F_m, it is necessary to examine the circuits that are used to evaluate each architecture.

Table III breaks down all nets in the benchmark circuits into three categories: nets that connect only logic blocks, nets that connect exactly one memory block to one or more logic blocks, and nets that connect *more* than one memory block to one or more logic blocks.

Consider the third category of nets: nets that connect to more than one memory array. In the generated circuits, these nets serve two purposes. First, when arrays are combined to implement a larger user memory, the address pins (and possibly data-in pins) of the arrays are connected together. Second, often the data-in pins of several user memories are driven by a common data bus. These memory-to-memory nets are particularly hard to route with small values of F_m. Fig. 8 illustrates a net that connects to three memory pins. Assuming a low value of F_m, there are three regions of low flexibility; the logic routing resources must be used to connect to specific tracks incident to each of these three low-flexibility regions. Given the relatively few options available within each switch block, circuitous routes

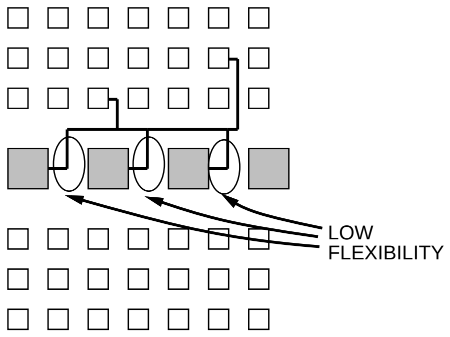


Fig. 8. A net connected to three memory blocks: three regions of low flexibility

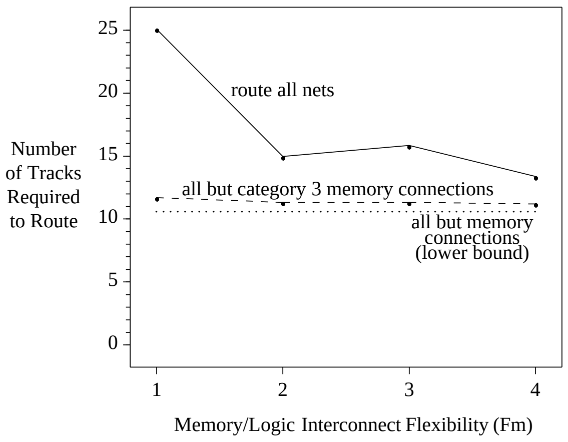


Fig. 9. Effect of removing memory-to-memory connections for 16-array FPGA.

are often required to make connections between these low flexibility regions, causing routability problems.

Intuitively, these nets will appear more often in circuits aimed at larger architectures, since there are likely more memory blocks to connect. Table III shows that this intuition is correct. Thus, the FPGAs used to implement the larger circuits need a higher value of F_m.

To investigate this further, we removed all memory-to-memory connections from the circuits, and repeated the experiment. Fig. 9 shows the results for the 16-array case. The solid line shows the original results from Fig. 7(a). The dashed line shows the results obtained from the circuits with the memory-to-memory connections removed. The dotted line shows the results obtained from the circuits with *all* memory connections removed (clearly, this is independent of F_m). As the graph shows, removing just the memory-to-memory connections gives a routability only slightly worse than that obtained by removing all memory connections (only slightly more tracks are required). This motivates us to study memory-to-memory connections more closely; in Section IV we will present architectural enhancements aimed at efficiently implementing memory-to-memory connections.

Class of architectures	$R = \frac{\text{num. logic blocks in horiz. dimension}}{\text{num. memory blocks}}$		aspect ratio	
	Average	St.Dev	Avg	St.Dev
2 arrays	12.1	3.35	1	0
4 arrays	6.60	1.99	1	0
8 arrays	3.93	0.86	1.08	0.25
16 arrays	3.01	0.01	3.16	1.72

TABLE II
Architecture statistics.

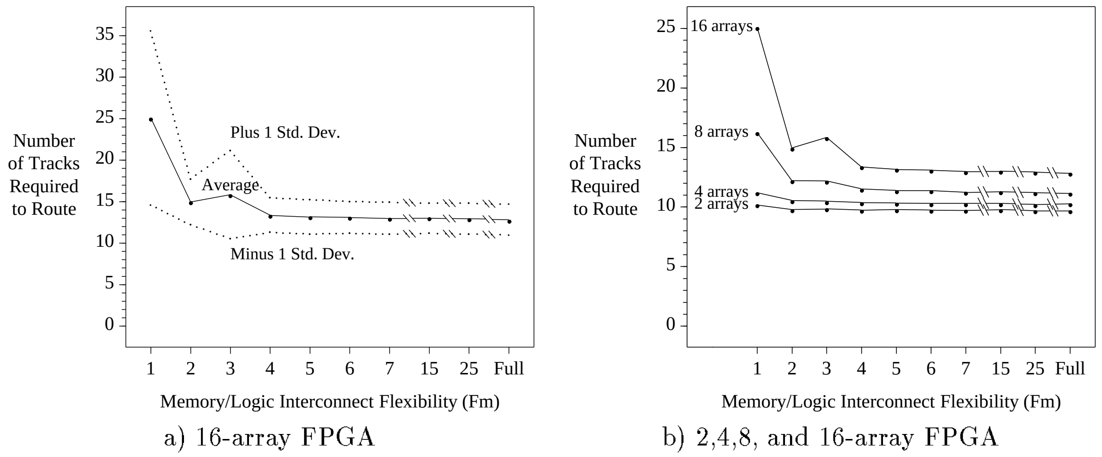


Fig. 7. Average track requirement as a function of F_m.

Architecture to which circuits are targeted	Nets not connected to memory	Nets connected to exactly 1 memory	Nets connected to more than 1 memory
2 arrays	94.2%	4.32%	1.45%
4 arrays	93.0%	5.24%	1.78%
8 arrays	92.2%	5.93%	1.91%
16 arrays	89.4%	8.17%	2.43%

TABLE III
Net statistics.

E. Area Results

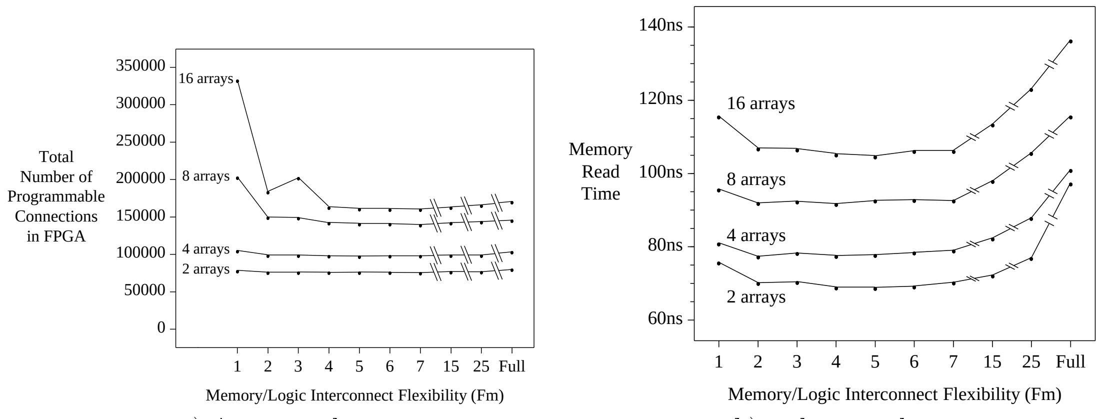
The area required by an FPGA architecture is the sum of the area required by the logic blocks, memory blocks, and routing resources. Since the value of F_m does not affect the number of memory blocks or logic blocks required to implement a circuit, we focus on the routing area. The routing resources are made up of three components: programmable switches, programming bits, and metal routing segments. Since estimates of the area required by the metal routing segments are difficult to obtain without performing a detailed layout, we concentrate on the area due to the programming bits and switches.

We are interested in the area of the entire FPGA, and thus must consider not only those switches in the memory/logic interconnect blocks, but also those in the logic routing architecture. The number of switches in the memory/logic interconnect block is proportional to F_m, while the number of switches in the logic routing depends on W (number of tracks in each channel). The sum of these two components gives an estimate of the total number of programming bits required in the FPGA routing.

Fig. 10(a) shows the number of programmable connections in each of the four architectures as a function of F_m. The results follow the routability measurements (Fig. 7) closely. Since the logic routing resources contain many more switches that the memory/logic interconnect does, the increase in the memory/logic interconnect area as F_m increases is swamped by the decrease in the area of the logic routing resources due to the decrease in the number of tracks per channel. In all architectures, the best area efficiency is obtained for $F_m \geq 4$. For very large values of F_m, the area increases slightly due to the larger memory/logic interconnect blocks and the inability of the extra switches to reduce the track requirement.

F. Delay Results

A detailed delay model is used to measure the memory read time of all memories in the circuit. The memory read time is the sum of the access time of the array itself, the delay of the network driving the address pins, and the delay of the network driven by the data-out pins. The array access



a) Area Results b) Delay Results

Fig. 10. FPGA Area and Delay Results as a function of F_m.

time is estimated by a modified version of CACTI, a detailed cache access time model [17]. For the address-in and data-out networks, the Elmore delay is used [18]. Commercial FPGAs often contain repowering buffers to reduce the delay of long nets; rather than assuming a specific repowering buffer strategy, we make the pessimistic assumption that each signal is repowered in *every* switch and memory/logic interconnect block. Although this architecture is not likely to be used in practice, the delay estimates obtained by assuming such an architecture will behave in a manner similar to those that would be obtained had a more intelligent buffer placement policy been employed.

Fig. 10(b) shows the average memory read time of the memories in the 100 benchmark circuits. The extremes of $F_m = 1$ and F_m equal to its maximum value are both bad choices. If $F_m = 1$, circuitous routes are required to connect nets to the low flexibility memory pins. These circuitous routes pass through more switch blocks than would otherwise be necessary, leading to longer net delays. When F_m is its maximum value, the large number of switches in the memory/logic interconnect block adds a significant amount of extra capacitance to the routing wires, again leading to longer routing delays. Between $F_m = 2$ and $F_m = 7$, the delay is roughly constant.

Combining these results, the most efficient FPGA architecture occurs $4 \leq F_m \leq 7$. As F_m approaches its maximum value, the speed-efficiency is reduced considerably, and the area-efficiency somewhat.

IV. Enhancement to Support Memory-to-Memory Connections

In Section III-D it was suggested that nets that connect to more than one memory array are difficult to route in low-F_m architectures, and that these nets are common in circuits that use many memory arrays. In this section we propose adding programmable switches between neighbouring memory arrays to support these nets. These extra switches take up negligible area. Below we will show that, if they are employed correctly, these switches provide a significant improvement in both the routability and speed of the device.

A. Enhanced Architecture

Fig. 11 illustrates the enhanced architecture. Each vertical wire incident to a memory/logic interconnect block can be programmably connected to the corresponding wire incident to the two neighbouring memory/logic interconnect blocks. The connection is made through pass transistors denoted by rectangles in Fig. 11. We refer to each of these pass transistors as a *memory-to-memory switch*. Note that the connections shown as solid dots are non-programmable permanent connections.

Fig. 12 shows an example of a net connecting three arrays implemented on both the baseline and the enhanced architectures. In the baseline architecture, the net is implemented using the logic routing resources that could otherwise be used to route signals between logic blocks. Because of the limited connectivity within each switch block, the route through the logic routing resources is somewhat circuitous; in the presence of routing contention, the route may be even worse. In the enhanced architecture, however, two memory-to-memory switches are used to connect the three memory arrays.

The area cost of the new memory-to-memory switches is small. If there are N arrays and V vertical tracks per memory block, then NV extra switches and programming bits are required.

This enhancement is related to the broader channel segmentation issue in FPGAs [6], [19]. There are, however, several major differences from this previous work:

1. In a segmented routing architecture, all channels across the chip usually contain an identical distribution of segment lengths. In our architecture, there are exactly V additional horizontal tracks, regardless of how many logic routing channels exist on the chip. Thus, the routing architecture is heterogeneous to better match the heterogeneous logic/memory block architecture.
2. Each memory-to-memory connection consists of a programmable switch connecting two tracks. This is topologically different than a standard routing track (of any length), in which two programmable switch blocks are connected using a fixed track.

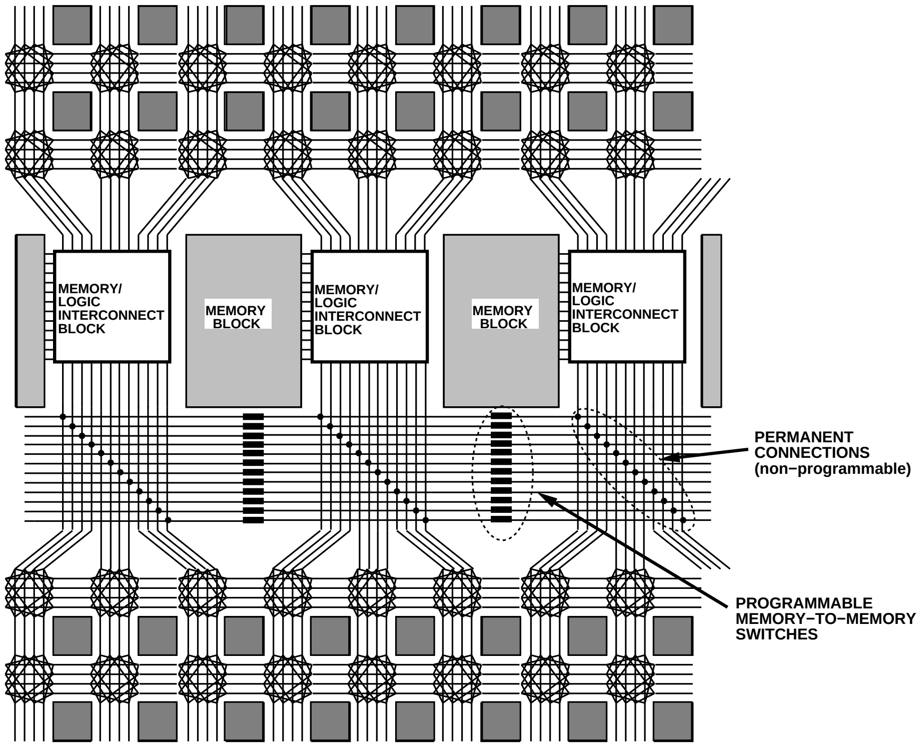


Fig. 11. Enhanced memory/logic interconnect architecture.

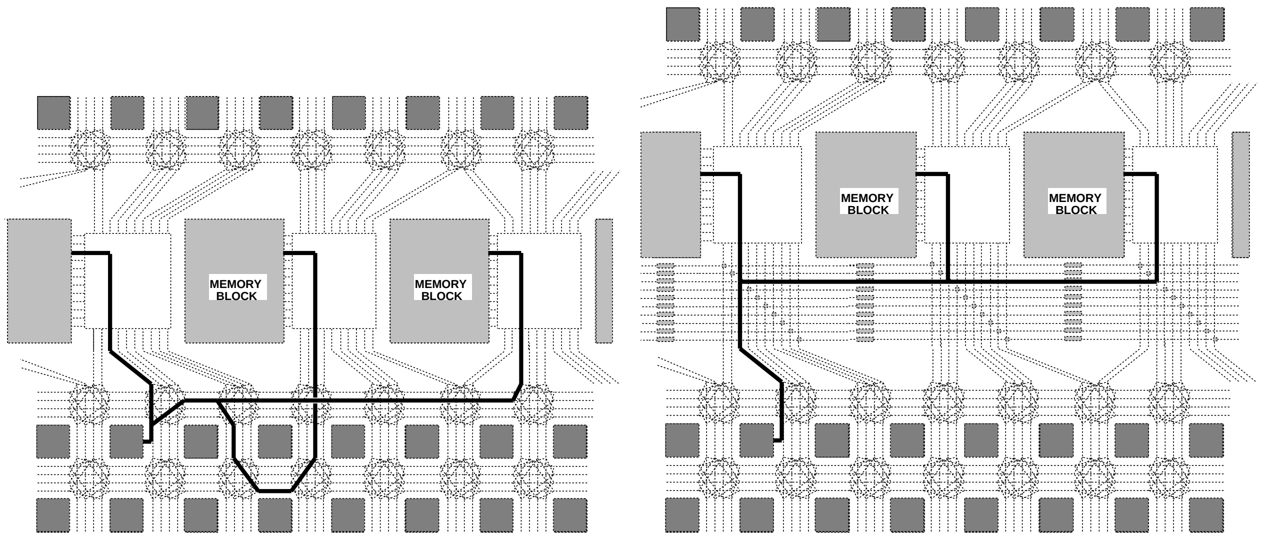


Fig. 12. Routing a net on the baseline and enhanced architectures.

3. The purpose of these tracks is different than the long lines used previously. In our architecture, the tracks are used to efficiently connect memory arrays that are next to each other; long lines, on the other hand, are used to connect distant logic blocks.

Our enhancment is also related to the direct connections between adjacent logic blocks in some commercial FPGAs [7], [5]. These connections are provided to provide low delay paths between logic blocks, and are often very effective at speeding up certain logic structures (carry chains, for example). Our enhancment can be thought of as an application of this technique to memory. By providing fast connections between neighbouring memory arrays, we can more efficiently implement large user memories that share address or data connections. Typically, if memories are connected together, they will share *all* their address or *all* their data connections. This regularity is reflected in the proposed architectural enhancement. By exploiting the regularity and using the direct memory-to-memory switches to connect adjacent blocks, we would expect significant performance improvements.

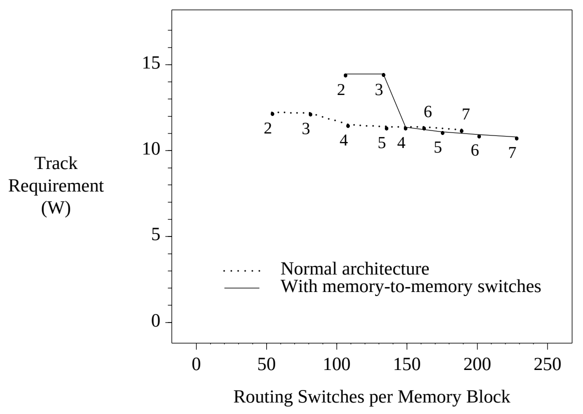


Fig. 13. Routing results using **standard maze router**

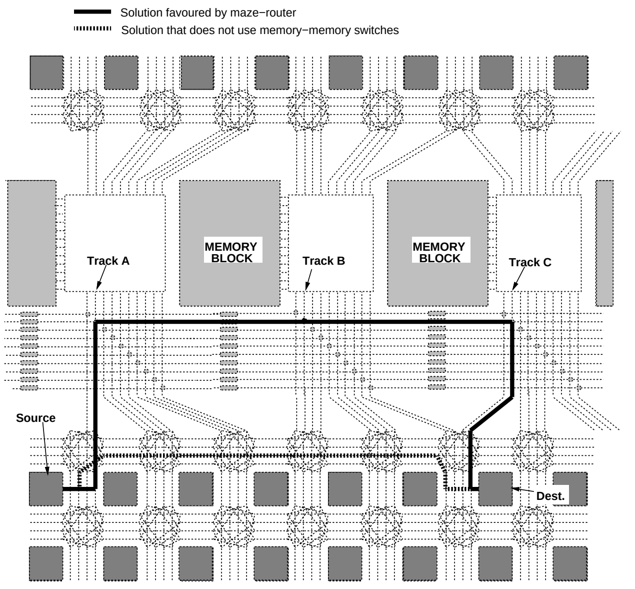
B. *Evaluation of Enhanced Architecture*

In this section, we will show that the proposed enhancement improves the speed and routability of circuit implementations. To obtain these improvements, however, the maze-routing algorithm must be restricted such that it uses the memory-to-memory switches *only* to implement memory-to-memory connections. If a standard maze-router that is free to use the memory-to-memory connections for *all* nets is employed, the extra switches *actually reduce the routability* of the device.

In order to quantify the gains obtained by the memory-to-memory switches, we employ the same experimental approach as that used in Section III. We first show results for the case when the router is free to use the memory-to-memory connections for all nets. Fig. 13 shows routability results for an FPGA with eight memory arrays with and without the memory-to-memory switches for several values of F_m. The horizontal axis is the number of programmable switches per memory array in the memory/logic interconnect. This includes the switches in the memory/logic interconnect blocks (proportional to F_m), and, in the enhanced architecture, the memory-to-memory switches. The vertical axis is the number of tracks required in each channel in order to completely route the circuit, averaged over all 100 benchmark circuits. Each point is labeled with the corresponding value of F_m.

As the graph shows, the required track count is significantly increased for low values of F_m and relatively unchanged for higher values. At $F_m = 1$, in the enhanced architecture, more that half of the circuits could not be routed using less than 45 tracks; we do not present results for this case.

The primary reason for these disappointing results is that nets that *do not* connect to memory will often use the memory-to-memory switches as a low-cost route to travel from one side of the chip to the other. Consider Fig. 14, which shows the connection between two distant logic blocks. If the net is implemented using only the logic routing resources, at least six switch blocks would lie on the path between the two logic blocks. Using the memory-to-memory switches, only two switch blocks and two pass transistors (one under each memory block) must be traversed. Since the latter alternative is cheaper, it will be favoured by a standard maze-type router.



Solution favoured by maze−router
Solution that does not use memory−memory switches
MEMORY BLOCK
MEMORY BLOCK
Track A
Track B
Track C
Source
Dest.

Fig. 14. Routing a logic net with and without memory-to-memory switches

Although this provides an efficient implementation of this net, the vertical tracks labeled A and C in the diagram become unavailable for future nets (the router processes nets one at a time). If future nets require connections to the memory, the loss of vertical tracks A and C may severely hamper the routing of these nets, especially in low-F_m architectures. Also, since the connections between the vertical tracks incident to each memory/logic interconnect block and the horizontal tracks connecting the memory-to-memory switches are permanent, the track labeled B will also be unavailable for future nets.

To alleviate this problem, we modified the router so that the memory-to-memory switches are used *only* to implement memory-to-memory connections. Although this means that these tracks are wasted if a circuit contains no (or few) memory-to-memory connections, it alleviates the problems described above.

Figs. 15 gives the track requirement results obtained using this algorithm for the 8 and 16 array FPGAs. Again the horizontal axis is the number of switches per memory array (including the memory-to-memory switches in the enhanced architecture) and the label above or below each point is F_m. As the graph shows, the memory-to-memory switches help somewhat, reducing the average track requirement by between 0.5 and 1 track.

Fig. 16 shows the area results for the 8 and 16 array FPGAs. As before, the area results closely match the track requirement measurements.

Fig. 17 gives delay results. As before, the vertical axis in each graph is the time to perform a read access, including the routing to the address pins and from the data-out pins. If an address net connects to more than one memory array, it might have a circuitous route in the baseline architecture, resulting in a longer net delay, and hence, a

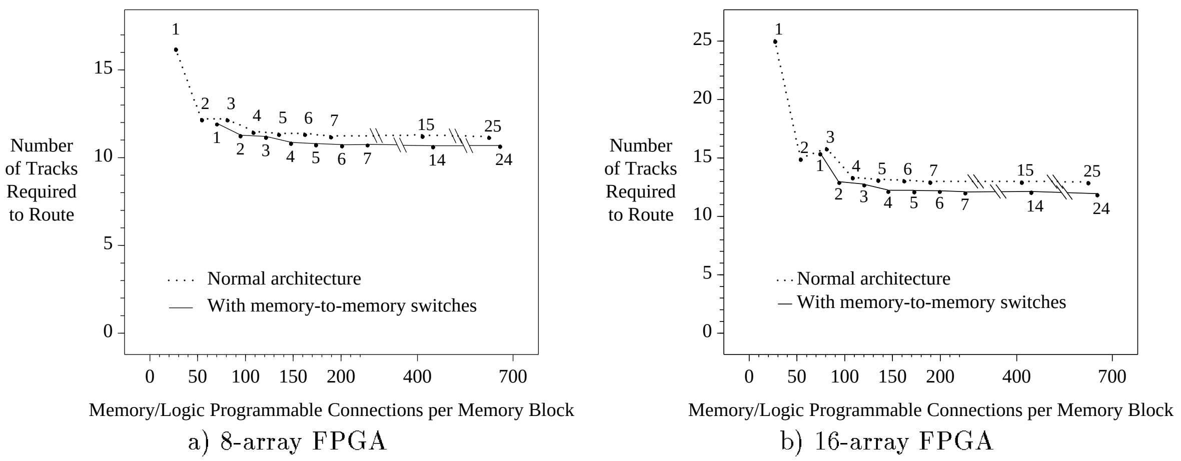


Fig. 15. Routability results for FPGAs with memory-to-memory switches.

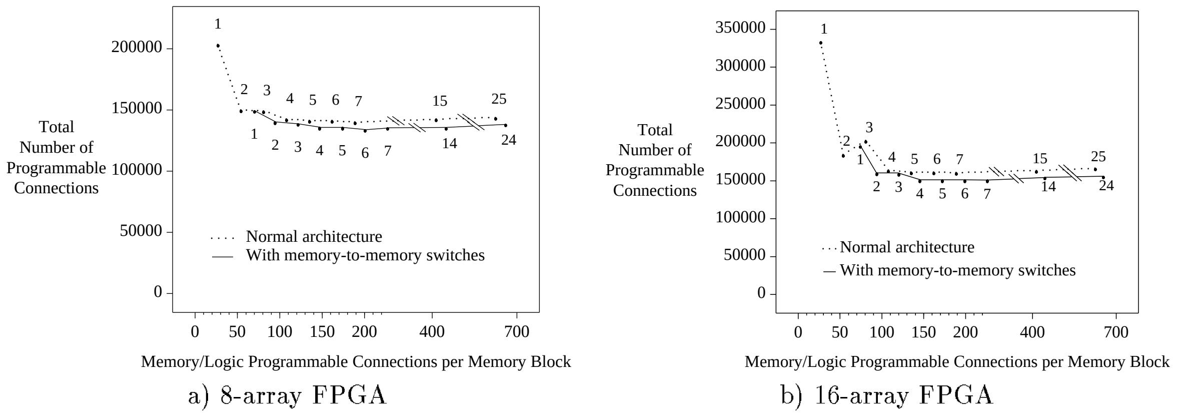


Fig. 16. Area results for FPGAs with memory-to-memory switches.

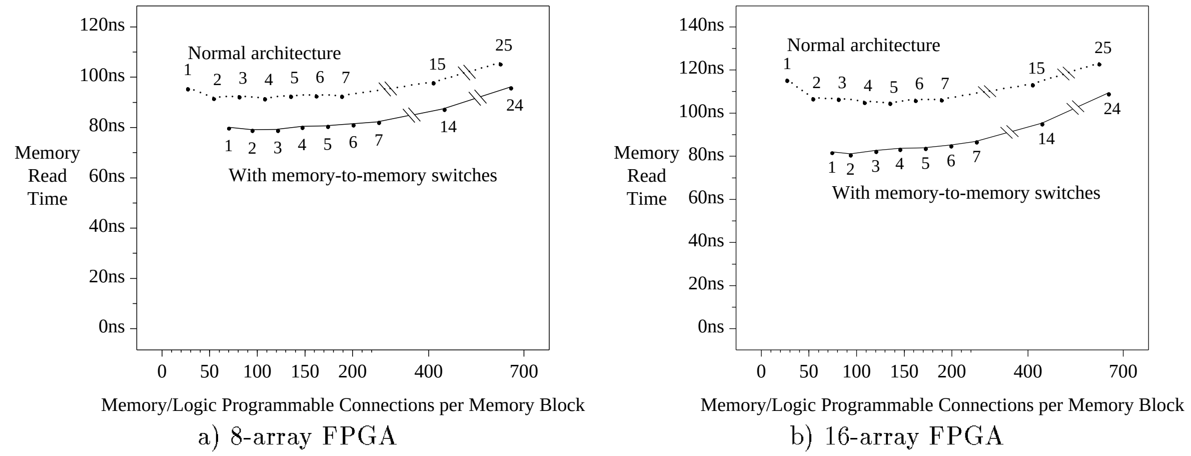


Fig. 17. Delay results for FPGAs with memory-to-memory switches.

longer read time. The memory-to-memory switches result in more direct routes for these nets, leading to lower memory read times. In the architectures considered here, the improvement is as much as 25%. Since the critical path of a circuit implementation often includes the memory read time, this speed-up will significantly impact the achievable clock frequency of circuits implemented on the FPGA.

These results show that even with this relatively unaggressive use of the memory-to-memory switches, area is improved somewhat, and speed is improved significantly. The development of algorithms that use these tracks more aggressively is left as future work; it is likely that such algorithms would give improvements beyond those presented in Figs. 15 through 17.

V. Conclusions

In this paper, we have examined the architecture of an FPGA with on-chip memory, focusing on the interconnection structure between the memory arrays and the logic resources. We have found that, in our architecture, the most area-efficient and speed-efficient architecture occurs if each memory pin can be programmably connected to between 4 and 7 tracks. This is in contrast to [11] which shows that circuit routability for logic circuits is severely hampered by low values of the connection block flexibility.

For FPGAs with 8 or more arrays, we showed that the routability and speed of the FPGAs can be improved by adding programmable switches between neighbouring memory blocks. In our architecture, the enhancements reduced the channel width by between 0.5 and 1 track (averaged over all benchmark circuits) and improved the speed of circuit implementations by as much as 25%. The area cost of these additional switches is small.

Acknowledgments

This work was performed at the University of Toronto. Financial support was provided by Xilinx, MICRONET, the Natural Sciences and Engineering Research Council of Canada, the Walter C. Sumner Memorial Foundation, and UBC's Centre for Integrated Computer Systems Research (CICSR).

References

[1] S. J. E. Wilton, J. Rose, and Z. G. Vranesic, "Architecture of centralized field-configurable memory," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 97–103, 1995.

[2] T. Ngai, J. Rose, and S. J. E. Wilton, "An SRAM-Programmable field-configurable memory," in *Proceedings of the IEEE 1995 Custom Integrated Circuits Conference*, pp. 499–502, May 1995.

[3] Altera Corporation, *Datasheet: FLEX 10K Embedded Programmable Logic Family*, July 1995.

[4] Actel Corporation, *Datasheet: 3200DX Field-Programmable Gate Arrays*, 1995.

[5] Actel Corporation, *Actel's Reprogrammable SPGAs*, 1996.

[6] Xilinx, Inc., *XC4000 Series (E/L/EX/XL) Field Programmable Gate Arrays v1.04*, Setpember 1996.

[7] AT&T Microelectronics, *Data Sheet: Optimized Reconfigurable Cell Array (ORCA) Series Field-Programmable Gate Arrays*, March 1994.

[8] S. J. E. Wilton, J. Rose, and Z. G. Vranesic, "Memory/logic interconnect flexibility in FPGAs with large embedded memory arrays," in *Proceedings of the IEEE 1996 Custom Integrated Circuits Conference*, pp. 144–147, May 1996.

[9] S. J. E. Wilton, J. Rose, and Z. G. Vranesic, "Memory-to-memory connection structures in FPGAs with embedded memory arrays," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 10–16, February 1997.

[10] S. J. E. Wilton, *Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory.* PhD thesis, University of Toronto, 1997.

[11] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 277–282, March 1991.

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

[13] K. Veenstra. private communications, 1995.

[14] S. Yang, "Logic synthesis and optimization benchmarks," tech. rep., Microelectronics Center of North Carolina, 1991.

[15] E. Sentovich, "SIS: A system for sequential circuit analysis," Tech. Rep. UCB/ERL M92/41, Electronics Research Laboratory, University of California, Berkeley, May 1992.

[16] J. Cong and Y. Ding, "FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, pp. 1–12, January 1994.

[17] S. J. E. Wilton and N. P. Jouppi, "CACTI: an enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 677–688, May 1996.

[18] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, pp. 55–63, Jan. 1948.

[19] S. Brown, G. Lemieux, and M. Khellah, "Segmented routing for speed-performance and routability in field-programmable gate arrays," *Journal of VLSI Design*, vol. 4, no. 4, pp. 275–291, 1996.

Steven J.E. Wilton received the B.Eng. degree in Computer Engineering from the University of Victoria, Victoria, B.C., Canada in 1990 and the M.A.Sc. and Ph.D. degree from the University of Toronto, Toronto, Ontario, Canada in 1992 and 1997 respectively.
Since 1997, he has been an Assistant Professor in the Department of Electrical and Computer Engineering at the University of British Columbia in Vancouver, B.C., Canada. His research interests include FPGA architecture, CAD algorithms for FPGAs, and VLSI design. In 1998, he won the Douglas R. Colton Medal for Research Excellence for his work in FPGA architectures and their associated CAD tools.

Jonathan Rose is a Professor of Electrical and Computer Engineering at the University of Toronto, and an NSERC University Research Fellow.
He received the Ph.D. degree in Electrical Engineering in 1986 from the University of Toronto. From 1986 to 1989, he was a Research Associate in the Computer Systems Laboratory at Stanford University. In 1989, he joined the faculty of the University of Toronto. He spent the 1995-1996 year as a Senior Research Scientist at Xilinx, Inc, in San Jose, CA, working on a next-generation FPGA architecture.
He is the co-founder of the ACM FPGA Symposium, and remains part of that Symposium on its steering and program committees. He has worked for Bell-Northern Research and a number of FPGA companies on a consulting basis.
His research covers all aspects of FPGAs including architecture, CAD, Field-Programmable Systems, and graphics and vision applications of rapid prototyping systems.

Zvonko G. Vranesic received the B.A.Sc., M.A.Sc., and Ph.D. degrees in Electrical Engineering from the University of Toronto in 1963, 1966, and 1968, respectively. From 1963 to 1965 he worked as a design engineer for Northern Electric Co. Ltd., Bramalea, Ontario, Canada. In 1968, he joined the faculty of the Departments of Electrical Engineering and Computer Science at the University of Toronto, where he is now a Professor. During the academic years 1977/78 and 1984/85 he was a Senior Visitor in the Computer Laboratory at the University of Cambridge, England, and in the Institut de Programmation at the University of Paris 6, France.
His research interests include computer architecture, VLSI systems, local area networks and many-valued switching systems. He has coauthored three books and published over 100 scientific papers.
He was the Chairman of the 3rd International Symposium on Multiple-Valued Logic in 1973 and of the 18th International Symposium on Computer Architecture in 1991.