Memory-to-Memory Connection Structures in FPGAs with Embedded Memory Arrays

Steven J.E. Wilton [*]
Dept. of Electrical Engineering
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
stevew@ee.ubc.ca

Jonathan Rose and Zvonko G. Vranesic
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 3G4
{jayar,zvonko}@eecg.toronto.edu

Abstract

This paper shows that the speed of FPGAs with large embedded memory arrays can be improved by adding direct programmable connections between the memories. Nets that connect to multiple memory arrays are often difficult to route, and are often part of the critical path of circuit implementations. The memory-to-memory connection structure proposed in this paper allows for the efficient implementation of these nets, resulting in a reduction in memory access time of up to 25% and a slight improvement in routability.

1 Introduction

As FPGAs become larger, they will be used to implement entire systems, rather than small logic subcircuits. One of the key differences between these large systems and the smaller logic subcircuits is that the systems often contain memory. Architectural support for the efficient implementation of memory in next-generation FPGAs, therefore, is crucial.

Several vendors offer FPGAs with architectural support for memory [1, 2, 3, 4, 5, 6, 7]. The memory resources in these devices can be classified into two categories: fine-grained and coarse-grained. Two examples of the fine-grained approach are the Xilinx XC4000 and Lucent Technologies ORCA FPGAs, in which each 4-input lookup table can be used as a 16-bit memory, and these small RAMs can be combined to implement larger memories. The coarse-grained approach is used in the Altera 10K CPLDs and the Actel 3200DX and ES devices. These FPGAs, which contain large embedded arrays, can implement large user memories much more efficiently since the per-bit overhead is significantly smaller. In this paper, we consider only coarse-grained memory architectures.

In our previous work, we examined a simple interconnect structure between the memory and logic resources in an FPGA, and concluded that only a small amount of flexibility is required, especially in FPGAs where there are few memory arrays [8]. For devices with 8 or more embedded arrays, it was suggested that the memory/logic interconnect flexibility requirements increase, due, in part, to connections between memory arrays. In this paper, we propose supporting these difficult nets by adding programmable connections between memory arrays. We show that this enhancement decreases the memory access time significantly, provides a slight improvement in routability, and requires very little additional chip area.

The enhancement will be presented in the context of a specific experimental architecture. Sections 2 and 3 of this paper describe the baseline architecture and the proposed enhancement. Sections 4 and 5 present experiments that evaluate the enhanced architecture.

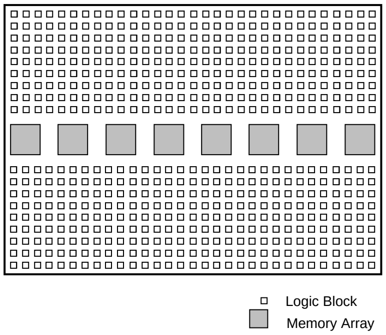


Figure 1: FPGA with embedded memory.

[*] This work was performed at the University of Toronto. Financial support was provided by the Natural Sciences and Engineering Research Council of Canada, MICRONET, and the Walter C. Sumner Foundation.

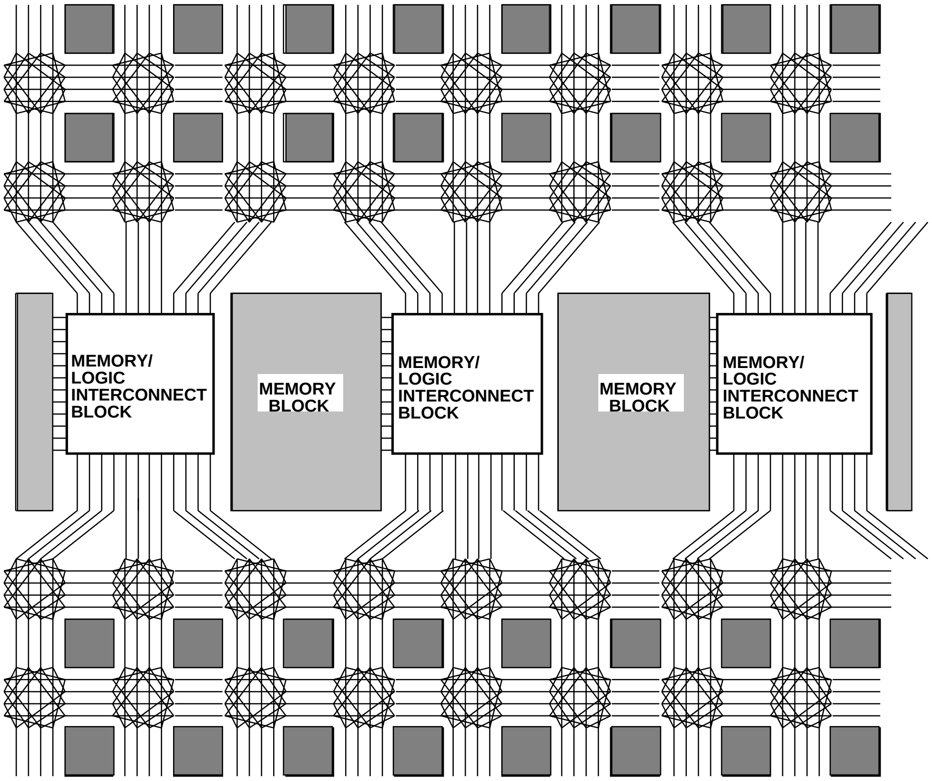


Figure 2: Memory/logic interconnect architecture.

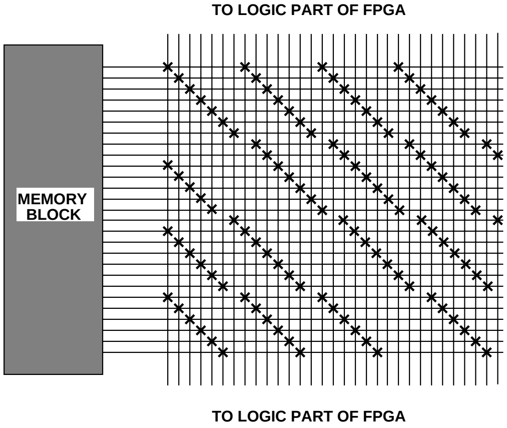


Figure 3: Memory/logic interconnect block.

2 Baseline Architecture

As in [8], we assume an FPGA with embedded arrays positioned in a row across the middle of the chip, as shown in Figure 1. Each array contains 2Kbits, and can be configured as 2Kx1, 1Kx2, 512x4, or 256x8 (similar to the Altera 10K series CPLDs [1]).

The logic resources consist of a grid of five-input lookup tables, connected using vertical and horizontal channels similar to the Xilinx and Lucent ORCA FPGAs [4, 5]. 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 [9]). 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.

Figure 2 shows the interconnect structure between the logic and memory parts of the FPGA. The vertical tracks in the top half of the FPGA are connected to those in the bottom half. The pins of each memory block are connected to one or more vertical tracks through a memory/logic interconnect block. An example memory/logic interconnect block is shown in Figure 3. The flexibility of this block, F_m, is defined as the number of vertical tracks to which each pin can connect; in Figure 3, $F_m = 4$. The switch pattern in each memory/logic interconnect block depends on F_m, the number of memory pins, and the number of incident vertical tracks. In all experiments described in this paper, the basic pattern of Figure 3 is retained [10].

3 Memory-to-Memory Connections

As suggested in [8], the routability and delay of the FPGA described in the last section is limited by nets that connect to more than one memory block. Mem-

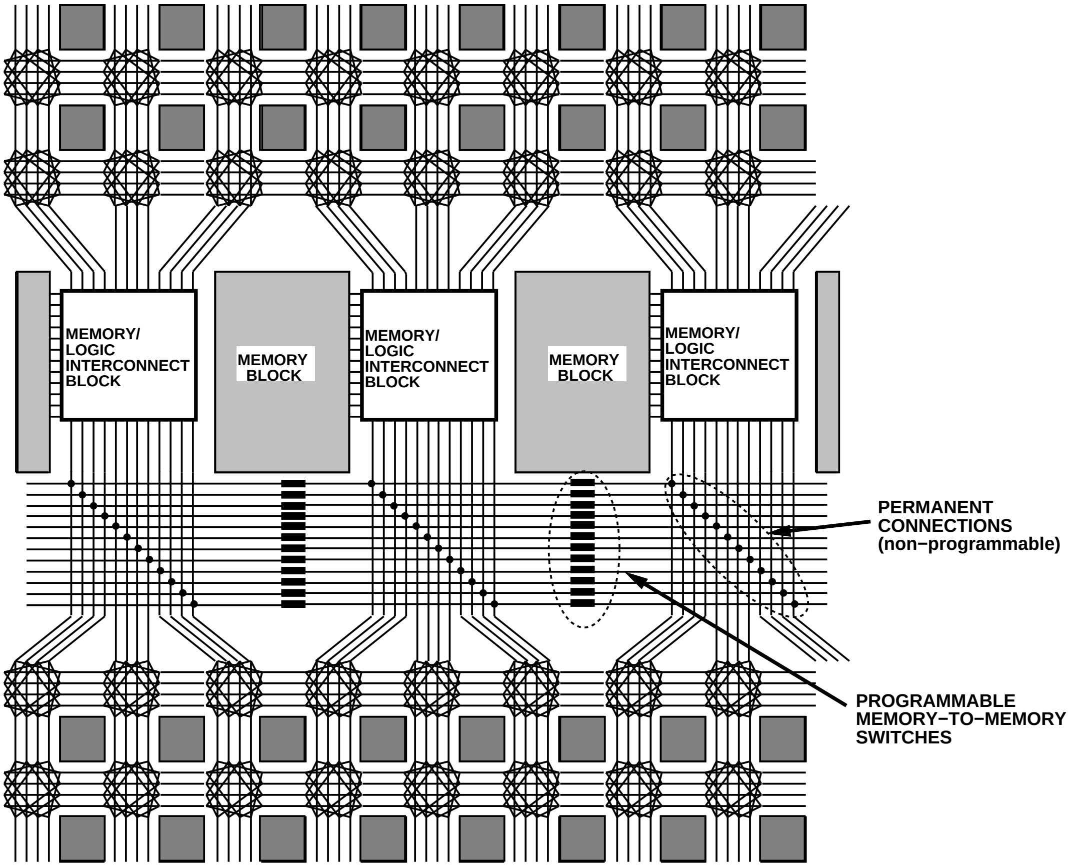


Figure 4: Enhanced memory/logic interconnect architecture.

ory blocks are connected together when they are combined to form larger memories, or when independent user memories share a common data bus. These nets are especially difficult to implement when the memory/logic flexibility is low. In addition, when memory arrays are to be connected to each other, usually there are *many* nets that connect the arrays in parallel, since typically *all* the address pins or all data pins of one array need to be connected to the corresponding pins of the other array. The high concentration of these difficult connections causes congestion near the memories. We propose an enhancement to the base architecture that supports these memory-to-memory nets.

Figure 4 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 Figure 4. 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.

Figure 5 shows an example of a net connecting three arrays implemented on the new architecture. Two memory-to-memory switches are used to connect the three memory arrays. The same net implemented on an FPGA without memory-to-memory switches is shown in Figure 6. In this case, the net must be 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.

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.

4 Experimental Methodology

In order to quantify the gains obtained by the memory-to-memory switches, we employ an experimental approach in which benchmark circuits are placed and routed on the baseline and enhanced architectures. In each case, we estimate the area and speed of the resulting circuit implementations.

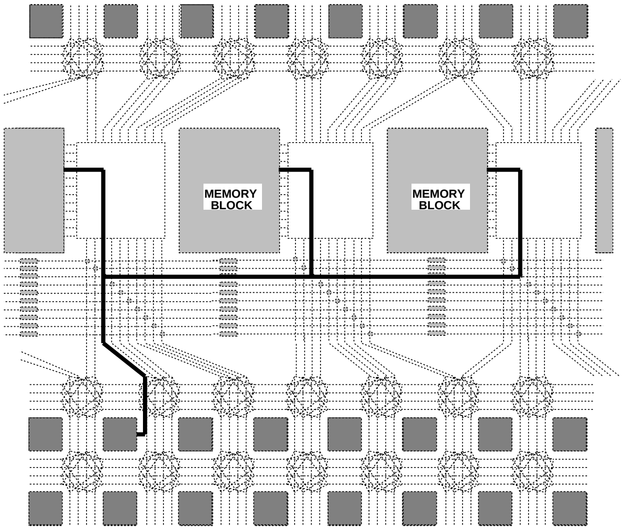


Figure 5: Enhanced architecture example.

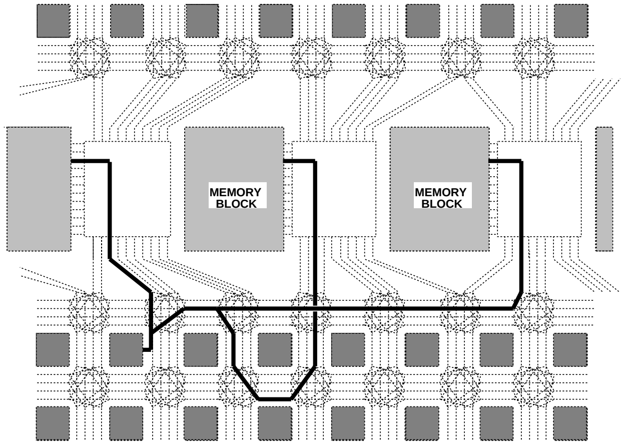


Figure 6: Baseline architecture example.

4.1 Benchmark Circuit Generation

The traditional method of placing and routing 10 to 20 benchmark circuits [9, 11] 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.

A memory configuration is the set of all memories required in an application circuit, and consists of the number of memory *clusters* (groups of memories which share data input or data output subcircuits [10]), the number of memories within each cluster, and the width and depth of each memory. The generator chooses these parameters using probability distributions based on statistics gathered from a set of 171 custom ASIC and programmable logic designs.

The generator then randomly chooses logic subcircuits from a collection of 38 MCNC circuits [12] and connects them to the memories. These subcircuits supply data to and consume data from the memories, as well as drive the memory address lines. These logic subcircuits have been optimized using SIS [13] and technology-mapped to 5-input lookup tables using FlowMap [14]. The interconnect patterns between the memories and logic subcircuits are chosen from a set of commonly occurring patterns in the set of ASIC designs. The actual construction of the circuits is non-trivial and is described in detail in [10].

This stochastic generation approach was also used in [8] and [15]. Unlike this previous work, we constrain the generator to create circuits that use between 75% and 100% of the total bits in the target architecture in order to fully stress the memory architectures. In the experiments described in Section 5, we consider architectures with both eight and sixteen 2-Kbit memory arrays. For each of these two memory sizes, we generate 100 circuits. The first nine columns of Table 1 show statistics for the generated circuits. In the circuits generated for the 8-array FPGA, 5.9% of the nets connect to memory, 24% of which connect to more than one memory array. In the circuits generated for the 16-array FPGA, 10.6% of the nets connect to memory, 23% of which connect to more than one memory array.

4.2 Circuit Implementation Procedure and Delay Modeling

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 *logical-to-physical mapping* and use an algorithm described in [10].

Next, the mapped circuits are then placed on an appropriately-sized FPGA using a simulated annealing-based placement program, and the detailed routing is performed using a multi-pass maze router (both tools are described in [10]). 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 equivalent, as well as all address pins of a memory array. 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).

The size of the FPGA used in the place and route

Arrays	5-LUTs Used		Arrays Used		I/O Blocks		Nets		R		Aspect Ratio	
in FPGA	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev
8	913	467	7.72	0.57	146	85.5	1069	497	3.93	0.86	1.08	0.25
16	849	500	15.2	1.33	154	102	1030	500	3.01	0.01	3.16	1.72

Table 1: Circuit and implementation statistics.

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 1 shows the average values of R and the average aspect ratio (ratio of horizontal logic blocks to vertical logic blocks) for circuits implemented on the two architectures.

Finally, a timing analyzer is used to measure the read access time of all memories in the circuit, including the routing to and from the memory arrays (but not including the logic block delays at either end). The timing analyzer finds each net delay by constructing an RC-tree and finding the Elmore delay [16]. 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 are similar to those that would be obtained had a more intelligent buffer placement policy been employed. The memory access time is measured from a modified version of CACTI, a detailed cache access time model [17]. A 0.5μm CMOS process was assumed.

5 Results

In this section, we 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.

We first show results for the case when the router is free to use the memory-to-memory connections for all nets. Figure 7 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.

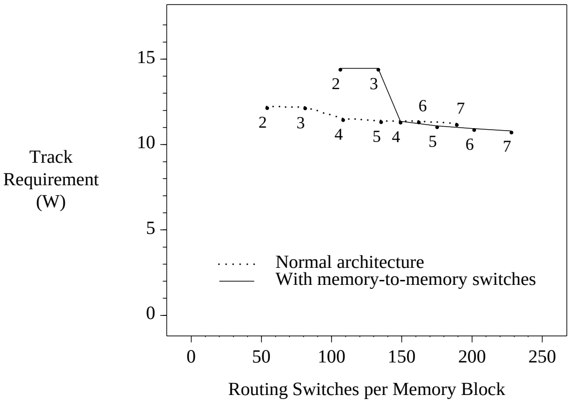


Figure 7: Routing results using **standard maze router**

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 Figure 8, 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 tran-

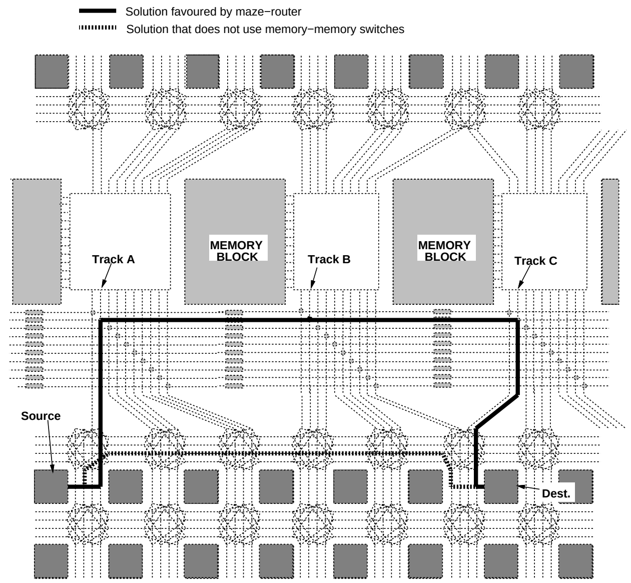

Solution favoured by maze−router
Solution that does not use memory−memory switches
Track A
MEMORY BLOCK
Track B
MEMORY BLOCK
Track C
Source
Dest.

Figure 8: Routing a logic net with and without memory-to-memory switches

sistors (one under each memory block) must be traversed. Since the latter alternative is cheaper, it will be favoured by a standard maze-type router.

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.

Figures 9(a) and 10(a) give the results obtained using this algorithm for the 8 and 16 array FPGAs respectively. 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.

Figures 9(b) and 10(b) give delay results. In each graph, the vertical axis is the time to perform a read access, including the routing to the address pins and from the data-out pins. Since each address and data line likely has a different delay, the combination that gives the maximum delay is chosen. 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 longer read time. The memory-to-memory switches result in more direct routes for these nets, leading to lower memory read times. In the architecture considered here, the difference 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.

6 Conclusions

In this paper, we have shown that the routability and speed of FPGAs containing large embedded memory arrays can be improved by adding programmable switches between neighbouring memory blocks. In our experimental 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. These results were obtained using an algorithm that uses the new switches only to implement memory-to-memory connections. 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 Figures 9 and 10.

References

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

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

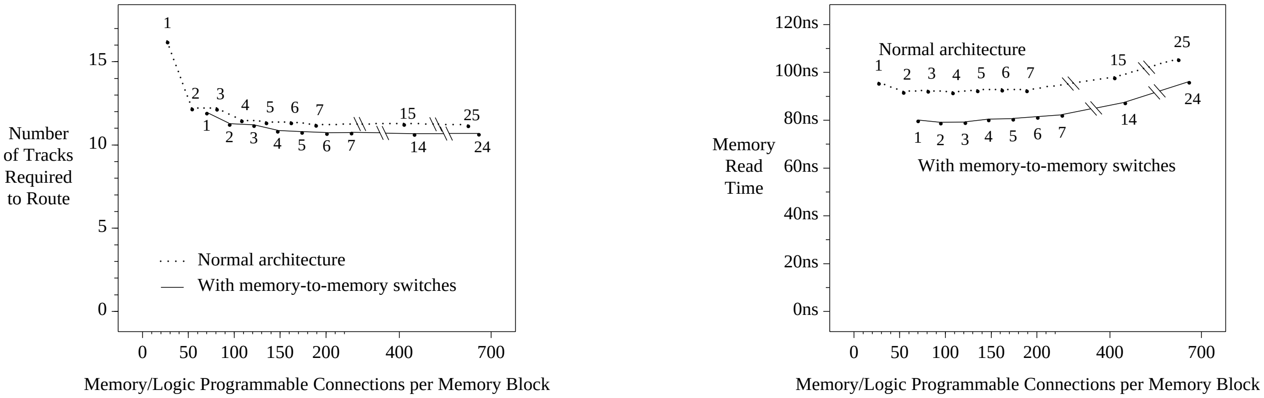
[3] Actel Corporation, *Datasheet: 3200DX Field-Programmable Gate Arrays*, 1995.

[4] Xilinx, Inc., *The Programmable Logic Data Book*, 1994.

[5] AT&T Microelectronics, *Product Brief: AT&T Optimized Reconfigurable Cell Array (ORCA) Series Field-Programmable Gate Arrays (FPGAs)*, April 1993.

[6] Lattice Semiconductor Corporation, *Datasheet: ispLSI and pLSI 6192 High Density Programmable Logic with Dedicated Memory and Register/Counter Modules*, July 1996.

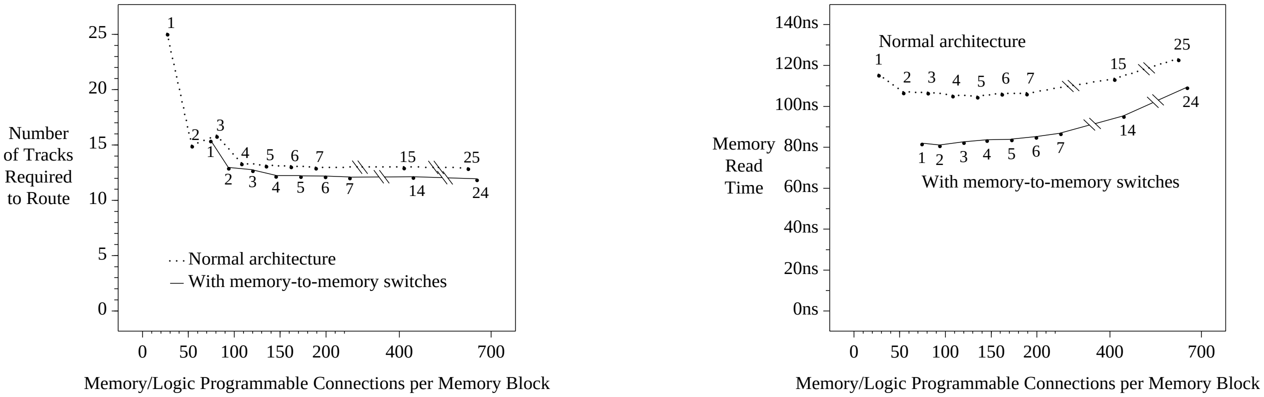
[7] Crosspoint Solutions, Inc., *CP20K Field Programmable Gate Arrays*, November 1992.



a) Routability

b) Delay

Figure 9: Results for an 8-array FPGA with memory-to-memory switches



a) Routability

b) Delay

Figure 10: Results for a 16-array FPGA with memory-to-memory switches

[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] 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.

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

[11] 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.

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

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

[14] J. Cong and Y. Ding, "An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 48–53, November 1992.

[15] 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.

[16] 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.

[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.