

Figure 3: Mapping a Circuit to a 8-Input, 1-Output Memory Block

6 Future Directions

This paper has outlined recent research into FPGA memory architectures. We are currently extending this work in several ways:

- We are investigating alternative memory architectures, in particular, dual-port arrays. Our initial results show that dual-port arrays are very suitable for inclusion in FPGAs.
- We are investigating other means of packing logic into unused memory arrays. In particular, Altera has announced a new memory block in which each array can be used in a *product-term mode* [16]. In this mode, each array can be used to implement any 16 functions with up to 32 inputs and containing 32 product-terms. It remains to be seen if such an architecture can be used effectively.
- We are investigating how well our architectures and algorithms work when migrated to very large arrays and very large numbers of arrays.

References

- [1] Altera Corporation, *Datasheet: FLEX 10K Embedded Programmable Logic Family*, May 1998.
- [2] Altera Corporation, *Datasheet: FLEX 10KE Embedded Programmable Logic Family*, August 1998.
- [3] Xilinx, Inc., "Virtex: Our new million-gate 100-MHz FPGA technology." XCell: The Quarterly Journal for Xilinx Programmable Logic Users, First Quarter 1998.
- [4] Actel Corporation, *Datasheet: Integrator Series FPGAs: 40MX and 42MX Families*, April 1998.
- [5] Lattice Semiconductor Corporation, *Datasheet: ispLSI and pLSI 6192 High Density Programmable Logic with Dedicated Memory and Register/Counter Modules*, July 1996.
- [6] 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.
- [7] 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.
- [8] S. J. Wilton, J. Rose, and Z. Vranesic, "The memory/logic interface in FPGA's with large embedded memory arrays," *IEEE Transactions on Very-Large Scale Integration Systems*, vol. 7, March 1999.
- [9] S. J. E. Wilton, "SMAP: heterogeneous technology mapping for FPGAs with embedded memory arrays," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 171–178, February 1998.
- [10] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 179–187, February 1998.
- [11] R. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based fpgas," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 227–233, June 1991.
- [12] 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.
- [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] S. J. E. Wilton, *Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory*. PhD thesis, University of Toronto, 1997.
- [15] S. J. E. Wilton, "Implementing logic in FPGA embedded memory arrays: Architectural implications," in *Proceedings IEEE Custom Integrated Circuits Conference*, May 1998.
- [16] F. Heile and A. Leaver, "Hybrid product term and LUT based architectures using embedded memory blocks," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 13–16, 1999.

Note: Many of these papers can be obtained from <http://www.ece.ubc.ca/~steve>

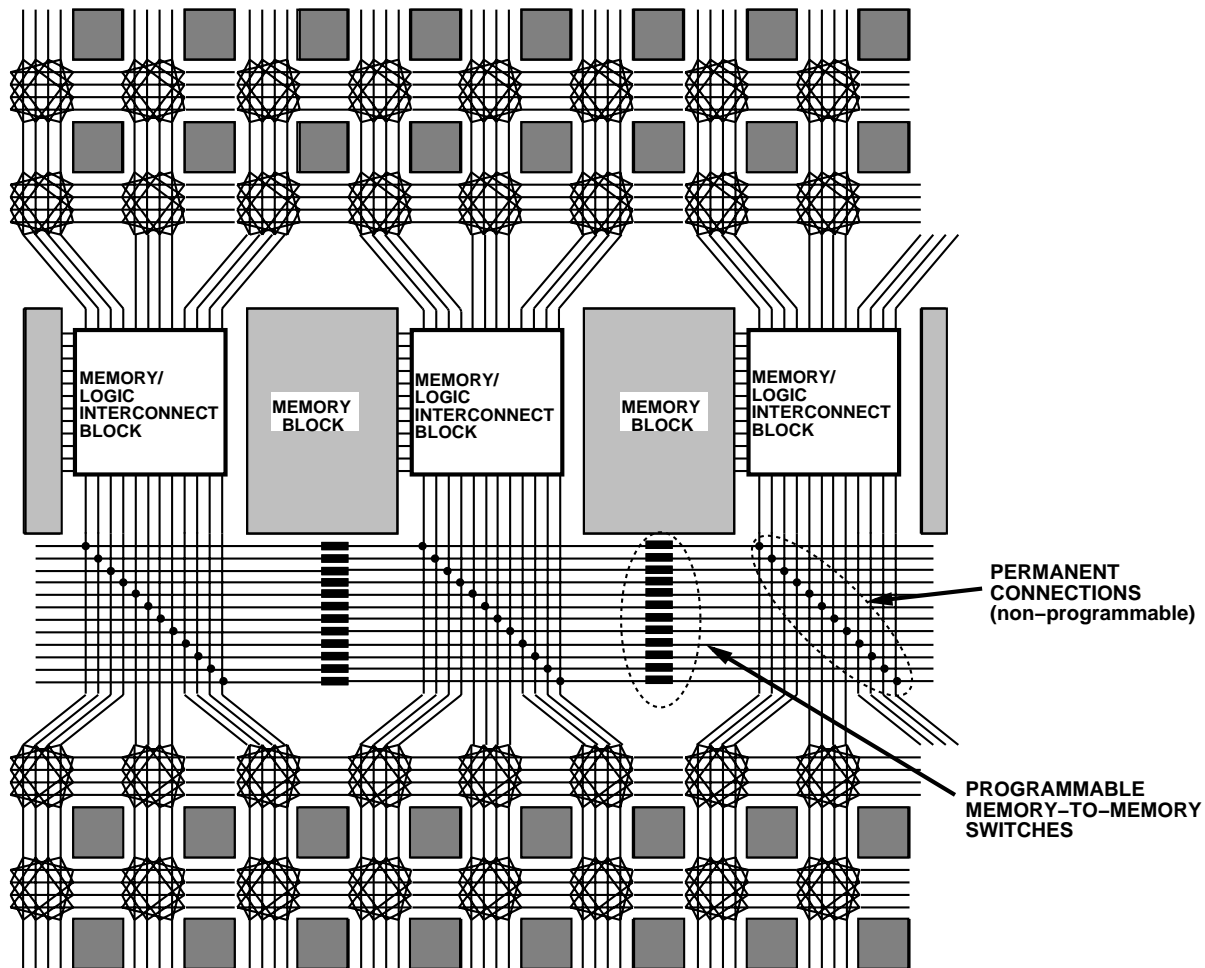


Figure 2: Memory/logic interconnect architecture.

There have been two published algorithms that perform this packing automatically [9, 10]. Both algorithms determine which parts of the logic circuit can be efficiently mapped into the memory arrays, and figure out the best way to do this packing. This task is much harder than previously developed FPGA synthesis algorithms [11, 12], since memory arrays are large, and typically have multiple outputs.

To evaluate the algorithm in [9], we used 29 large benchmark circuits. For each, we implemented the circuit two ways: (a) we used standard experimental CAD tools ([13, 12]) to implement the circuit assuming no memory arrays are available; (b) we used the new algorithm to pack as much of the circuit as possible into the available arrays, and then implemented the rest using the same standard experimental CAD tools. In each case, we counted the number of logic blocks required to implement the circuit. On average, assuming only one 2 K-bit memory array is available, circuits implemented using method (b) required 60 fewer logic blocks than those implemented using method (a). In other words, on average, 60 logic blocks worth of information was packed into the available array.

To appreciate the significance of these results, consider the

following. Using a detailed area model, we have estimated that the chip area required by a single 2 K-bit memory array is the same as the area required to implement 16 logic blocks (including routing) [14]. This area would be wasted if an unused array is not used to implement logic. Using our packing algorithm, however, not only is this area not wasted, but it is used *more efficiently than it would have been used if the array was replaced by logic blocks*. Had the array not been present, the user would be able to implement 16 logic blocks of his/her circuit in that chip area, while, using our algorithm, the user can implement 60 logic blocks of circuitry in the same chip area. Thus, the presence of embedded memory blocks leads to density improvements even if the user's circuit requires no storage at all.

We also investigated how the array size and flexibility affects its ability to implement logic [15]. The larger or more flexible the array, the more logic that can be packed. On the other hand, a larger and more flexible array consumes more chip area. Overall, we found that a 2 K-bit array that can be configured as a 2Kx1, a 1Kx2, a 512x4, or 1024x8 provides the best tradeoff.

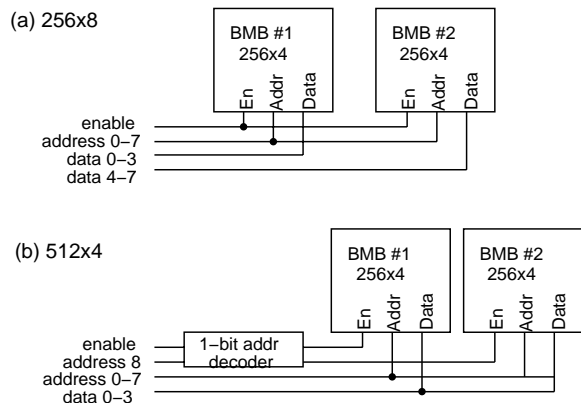


Figure 1: Two ways to group two 256x4 BMBs

the flexibility of the interconnect matrix, a total of 133 unique memory configurations can be implemented.

The chip was implemented and fabricated through the Canadian Microelectronics Corporation. Full details on the architecture and implementation can be found in [6].

3.2 Architecture Study

Using size and speed information obtained during the implementation of the prototype configurable memory, we then performed an architecture study to determine (1) the optimum number of BMBs (for a given total memory size), and (2) whether or not the flexibility of each BMB is warranted. We considered architectures with between 8Kbits and 64Kbits of memory. Full details on these studies are available in [7].

First consider the optimum number of BMBs. For a given total memory size, the more BMBs, the smaller they are. A smaller BMB is faster (since the wordlines and bitlines are shorter), however the interconnect will be slower (since there are more BMBs to connect). In terms of area, the more BMBs there are, the larger the device, since there is more overhead (interconnect matrix, decoders, multiplexors, etc.). In terms of flexibility, if there are more BMBs, the device will be more flexible (in terms of the total number of configurations that can be implemented). Considering all three criteria, we found that 8 BMBs was optimum for all memory sizes we considered.

We then investigated whether the flexibility in each BMB (the ability to take on different aspect ratios) is worthwhile. Again considering speed, area, and overall device flexibility it was determined that the BMB flexibility *is* worthwhile. The ability of the device to implement different memory configurations is severely hampered if we reduce the flexibility of each BMB.

4 Embedded Memory Architectures

We then investigated the embedding of flexible memory arrays onto an FPGA. 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.

Figure 2 shows the memory/logic interface architecture we investigated. We assume that the arrays are positioned in a single row across the width of the chip, as is the case in Altera 10K parts [1]. 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.

Each memory block is connected to the logic routing through a *memory/logic interconnect block*. Each such block programmably connects each memory pin to one of the vertical tracks connected to the upper and lower halves of the logic array. The block only allows a small subset of possible connections; in [8] we show that the optimum density and speed occurs if each memory pin can be connected to between 4 and 7 logic routing tracks.

FPGA memory arrays are very often combined to implement larger user memories, as shown in Figure 1. It is shown in [8] that these memory-to-memory connections are very difficult to implement in an FPGA, and often result in an unroutable circuit or an implementation that is slower than necessary. To improve the ability of an FPGA to implement these connections, we include a number of *memory-to-memory switches*, as shown in Figure 2. In [8], it is shown that, by incorporating these switches into our FPGA architecture, the memory access time (including routing into and out of the memories) is reduced by up to 25%.

5 Mapping Logic in to Memory Arrays

Unfortunately, embedding large memory arrays onto an FPGA requires the FPGA vendor to partition the chip into memory and logic regions when the FPGA is designed. Since circuits have widely-varying memory requirements, this “average-case” partitioning may result in poor device utilizations for logic-intensive or memory-intensive circuits. In particular, if a circuit does not use all the available memory arrays to implement storage (or in the worst-case, uses none at all), the chip area devoted to the unused arrays is wasted.

This chip area need not be wasted, however, if the unused memory arrays are used to implement logic. Configuring the arrays as ROMs results in large multi-output lookup-tables that might very efficiently implement some logic circuits. Figure 3 shows this graphically. The left-hand side of the figure shows a circuit; each square represents a gate with between one and four inputs and one output. Each such gate would normally be implemented using a single logic block in an FPGA. Thus, this implementation would require 13 logic blocks. If, however, a 8-input, 1-output memory array is available, the circuit can be implemented as in the right-hand side of Figure 3. In this case, only 4 logic blocks and one memory array is required.

System	Memory Requirements ¹
Graphics Chip	eight 128x22, two 16x27
Neural Networks Chip	16x80, 16x16
Translation Lookaside Buffer	two 256x69 (buffers), 16x18 (register file)
Proof-of-concept Viterbi Decoder	three 28x16, one 28x3
Fast Divider	2048x56, 4096x12 (ROM)
Communications Chip 1	two 1620x3, two 168x12, two 366x11
Communications Chip 2	six 88x8, one 64x24
Communications Chip 3	one 192x12

Table 1: Example systems.

these devices vary greatly from manufacturer to manufacturer; since embedded FPGA memory is new, the best way to include it on an FPGA is not known. The goal of research projects in this area is to investigate novel FPGA memory architectures, and examine some of the issues that arise as FPGA vendors embed memory onto their devices. The early work of this project dates back to 1993, before any vendor offered memory on their chips; some of the early results significantly affected the architecture of these commercial devices. The experiments being performed today will have a direct impact on the architecture of next-generation FPGAs.

In this paper, we will describe recent research into FPGA memory architectures, mostly performed at the University of British Columbia and the University of Toronto. Section 2 outlines the requirements of a FPGA memory architecture. Section 3 describes research into stand-alone configurable memory architectures, in which the goal is to provide a flexible memory architecture that can be used for many different applications, each with very different memory requirements. Section 4 then describes work in embedding these memory architectures on an FPGA; the challenge here is to create an interconnect architecture that is flexible, yet small and fast. The projects outlined in Sections 3 and 4 assume the memory arrays are used to implement the storage parts of circuits; Section 5 shows that the arrays can also be used to efficiently implement the logic parts of circuits. Finally, Section 6 outlines some of the future directions of this research area.

2 Requirements of a Good Memory Architecture

Since FPGA memory will be used in many different contexts, it must be flexible. Table 1 shows a number of circuits described in recent circuits conferences and journals as well as three circuits obtained from a Canadian telecommunications company; each of these circuits requires a different number of memories and different memory sizes.

¹In this paper, a memory written $m \times n$ consists of m words of n bits each.

Another element of flexibility not reflected in Table 1 is that the logical memories within a circuit will be used in many different contexts, and will therefore communicate with each other and with the logic in many different ways. Some circuits require a single large bank of memory that is connected directly to logic subcircuits. Other circuits might contain smaller memories connected to a common bus; this bus then might drive (or be driven by) one or more logic subcircuits. A third type of circuit might require many small memories distributed throughout the circuit, each connected to its own logic subcircuit. Again, a good FPGA will be able to implement circuits that interact with memory in many different manners.

In general, the more flexible an FPGA architecture, the more programmable switches and programming bits are required. Programmable switches add delay to the paths within a circuit implementation; if these paths are part of the circuit's critical path, the achievable clock speed will suffer. Similarly, the extra switches and programming bits will use area that could otherwise be used for additional circuit elements. Thus, the design of a good FPGA architecture involves finding a balance between area, speed, and flexibility.

3 Stand-Alone Memory Architectures

The first research project in this area investigated efficient stand-alone field-configurable memory architectures. Such an architecture is not embedded on an FPGA, but rather is implemented on a separate chip. The primary reason for considering such an architecture is that it allowed us to focus on the memory itself, without having to worry about the interconnect of the configurable memory to the logic part of an FPGA.

This research project had two components: we first built a prototype proof-of-concept configurable memory [6], and then performed an architecture study to determine the best parameters for such an architecture [7]. The architecture we proposed is very similar to what was subsequently included in the Altera FLEX 10K device [1].

3.1 Prototype Configurable Memory

The prototype configurable memory consists of four *basic memory blocks* (BMBs), each containing 1Kbit of memory. Flexibility is provided in two ways: the aspect ratio of each BMB is configurable, and the BMBs can be combined in a flexible manner. First, consider the aspect ratio of each BMB. Although the physical layout of the array is fixed, a programmable multiplexor connected to the data lines of each BMB allow each BMB to appear to the user as a 1024x1, a 512x2, a 256x4, or a 128x8 memory. The aspect ratio of each BMB can be configured independently. Second, a flexible interconnect matrix is provided to allow the user to programmably combine the BMBs to implement larger user memories. Figure 1 shows two ways in which two 256x4 BMBs can be combined. The interconnect matrix is flexible enough that all possible BMB combinations can be implemented. Considering both the BMB's flexibility and

Embedded Memory in FPGAs: Recent Research Results

Steven J.E. Wilton

Department of Electrical and Computer Engineering

University of British Columbia

Vancouver, BC, Canada, V6T 1Z4

steve@ece.ubc.ca *

Abstract

Recent dramatic improvements in integrated circuit fabrication technology have led to Field-Programmable Gate Arrays (FPGAs) capable of implementing entire digital systems. Unlike the smaller circuits that have traditionally been targeted to FPGAs, these large systems often contain memory. Architectural support for the efficient implementation of memory in next-generation FPGAs is therefore crucial.

In this paper, we will describe recent research into FPGA memory architectures. We seek to uncover not only the best architecture for the memory arrays themselves, but the best architecture for their interconnection, and the interconnection of the memory architecture to the rest of the FPGA. We also show how memory arrays that are not used to implement storage can be used to implement the logic parts of circuits very efficiently. Many of the early research results have already been used in commercial FPGAs; others are likely to be used in the future.

1 Introduction

Since their introduction in 1985, Field-Programmable Gate Arrays (FPGAs) have rapidly become the implementation medium of choice for many digital circuit designers. The reconfigurable nature of FPGAs provides risk-free large-scale integration that has been applied in areas as diverse as telecommunications, high-speed graphics, and digital signal processing. Unlike Mask-Programmed Gate Arrays (MPGAs), which must be personalized by the MPGA vendor, FPGAs can be programmed by the user in minutes. For small and medium volumes, this reduces the cost, and shortens the time-to-market.

Unfortunately, there is an area and speed penalty associated with user-configurability. Unlike MPGAs, in which circuit elements are connected using metal wires, in an FPGA, programmable switches are used to connect circuit elements. These programmable switches add resistance and capacitance to all connections within a circuit, lowering the achievable clock frequency. The switches also require significant chip area, reducing the amount of logic on each device. In some FPGAs, Static RAM (SRAM) bits are required to control

the programming switches, reducing the number of circuit elements even further. For the most part, this has limited FPGAs to the implementation of relatively small logic subcircuits, often the “glue-logic” portions of larger systems.

Recent years, however, have seen dramatic improvements in processing technology. Today, $0.35\mu\text{m}$ and $0.25\mu\text{m}$ processes are common, and even smaller feature sizes are on the horizon. These smaller feature sizes have led to impressive improvements in the density of integrated circuits (ICs), which, in turn, have had a profound impact on the possible applications and design of ICs.

The impact of improving process technology is very evident in the evolution of FPGAs. Recently, FPGA vendors have introduced devices capable of implementing relatively large circuits and systems. These large systems are quite different than the smaller logic subcircuits that have traditionally been the target of FPGAs. One of the key differences is that these systems often contain memory. Therefore, next-generation FPGAs must be able to efficiently implement memory as well as logic. If they can not, the implementation of such a system would require both FPGAs (for the logic portion of the system) and separate memory chips. On-chip memory has several advantages:

- Implementing memory on-chip will likely decrease the number of chips required to fully implement a system, reducing the system cost.
- Implementing memory and logic on separate chips will often limit the achievable clock rate, since external pins (and board-level traces) must be driven with each memory access. If the memory access time is part of the critical path delay of the circuit, on-chip memory will allow a shorter clock period.
- For most FPGA packaging technologies, as the devices get larger, the number of logic elements grows quadratically with the edge-length of the chip. The number of I/O pins, however, grows only linearly. Thus, the availability of I/O pins is increasingly becoming a problem. This problem is aggravated if an FPGA is connected to an off-chip memory, since many I/O pins on the FPGA must be devoted to address and data connections. If the memory is implemented on-chip, these pins can be used for other purposes.

Several vendors now offer devices with significant amounts of on-chip memory [1, 2, 3, 4, 5]. The architectures of

*This work is being supported by B.C. Advanced Systems Institute, NSERC, the UBC Centre for Integrated Computer Systems Research, Cadence Design Systems and Cypress Semiconductor