# Architecture of Centralized Field-Configurable Memory

Steven J.E. Wilton, Jonathan Rose, and Zvonko G. Vranesic
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada, M5S 1A4
wilton@eecg.toronto.edu *

## Abstract

*As the capacities of FPGAs grow, it becomes feasible to implement the memory portions of systems directly on an FPGA together with logic. We believe that such an FPGA must contain specialized architectural support in order to implement memories efficiently. The key feature of such architectural support is that it must be flexible enough to accommodate many different memory shapes (widths and depths) as well as allowing different numbers of independently-addressed memory blocks. This paper describes a family of centralized Field-Configurable Memory architectures which consist of a number of memory arrays and dedicated mapping blocks to combine these arrays. We also present a method for comparing these architectures, and use this method to examine the tradeoffs involved in choosing the array size and mapping block capabilities.*

## 1 Introduction

Most digital systems are composed of both logic and memory. Field-Programmable Gate Arrays (FPGAs) have traditionally been used to implement the logic portion of a system, leaving the memory to be implemented using standard off-the-shelf memory chips. As the capacities of FPGAs grow, however, it becomes feasible to implement memory directly on the FPGA itself. This paper describes and compares a set of architectures for implementing on-chip Field-Configurable Memory (FCM).

*This work was supported by the Natural Sciences and Engineering Research Council of Canada, MICRONET, and the Walter C. Sumner Foundation.

| System | Memory Requirements |
|---|---|
| Viterbi decoder [1] | three 28x16, one 28x3 |
| Graphics Chip [2] | eight 128x22, two 16x27 |
| Neural Networks Chip [3] | 16x80, 16x16 |
| Translation Lookaside Buffer [4] | two 256x69 (buffers) 16x18 (register file) |
| Fast Divider [5] | 2048x56, 4096x12 (ROM) |
| Communications Chip #1 | two 1620x3, two 168x12, two 366x11 |
| Communications Chip #2 | six 88x8, one 64x24 |

Table 1: Example systems

There are several advantages to including memory on an FPGA. First, on-chip memory reduces the demand on the FPGA's I/O resources, especially for wide memories. Secondly, on-chip memory will likely result in faster circuits, since the I/O pins need not be driven which each access. Finally, on-chip memory will likely reduce a system's chip count, resulting in less expensive implementations.

Table 1 gives several example systems and their memory requirements. We will refer to the memory requirements of a given application circuit as a *logical memory configuration.* Each independent memory within a logical memory configuration will be referred to as a *logical memory.* Many configurations contain more than one logical memory; for example, the Viterbi decoder in Table 1 requires four logical memories. The specification of these four memories, with their widths, speeds, and any other special requirements (such as dual-port) make up the circuit's logical memory configuration.

The primary difference between standard memory and memory in FPGAs is that FPGA memory will be used in many different contexts and must, therefore, be flexible. Each circuit in Table 1 requires a different number of memories and different memory sizes. A good FCM architecture will allow the efficient implementation (in terms of area and speed) of a wide variety of logical memory configurations.

Since logic and memory have very different characteristics, we begin with the assumption that an FPGA that can implement both efficiently will have separate resources for each. It is well-known that look-up tables with about 4 inputs are well-suited for implementing logic [6]. A large memory, however, is implemented more efficiently using larger arrays; not only is the extra overhead of using many small look-up tables avoided, but also dedicated decoding and mapping circuitry can be provided instead of using logic blocks for that purpose. One of the questions we set out to answer in Section 4 is how big each of these blocks should be. In the architecture presented in Section 2, the data width of each block is configurable; the appropriate amount of configurability in each block will also be examined in Section 4.

FPGA architectures containing both logic blocks and memory arrays can be classified into two categories: centralized and distributed. In a centralized architecture, the memory arrays are all grouped together on the FPGA, which allows dedicated circuitry for combining these blocks to be easily included. In a distributed architecture, memory arrays are distributed throughout the chip. A distributed architecture should work well for applications that don't need to combine arrays to form large memories, since it would likely be easier to place the memories closer to their address and data sources and sinks. In this paper, we restrict our discussion to centralized architectures.

Several FPGA vendors already offer limited memory capability [7, 8, 9, 10, 11, 12]. For the most part, these existing architectures are aimed at implementing circuits with relatively small memory requirements. For circuits with larger memories, new architectures are needed. In the next section, we present a family of centralized Field-Configurable Memory architectures that can be included in an FPGA. Since little work has been done in this area, it is unclear how such architectures should be compared and evaluated. Section 3 describes our approach, and Section 4 examines the effects of changing various parameters of the architecture.

## 2 Configurable Memory Architecture

This section describes a family of Field-Configurable Memory architectures similar to the FiRM FCM described in [13]. The FCM architecture, illustrated in Figure 1, consists of $b$ bits divided evenly among $n$ arrays that can be combined (using the address and data mapping blocks) to implement logical memory configurations. The parameters used to characterize each member of this architectural family are given in Table 2. Since each logical memory requires at least one array, one address bus, and one data bus,

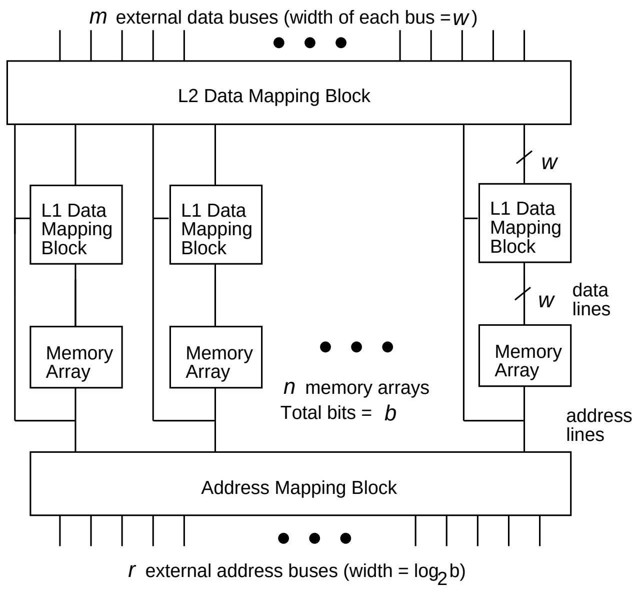


Figure 1: General architecture for a centralized FCM

| Parameter | Meaning |
|---|---|
| $b$ | Total bits |
| $n$ | Number of arrays |
| $m$ | Number of external data buses |
| $r$ | Number of external address buses |
| $w$ | Nominal data width of each array |
| $w_{\text{eff}}$ | Set of allowable effective data widths of each array |

Table 2: Architectural Parameters

the maximum number of logical memories that can be implemented on this architecture is the minimum of $m$, $r$, and $n$.

Flexibility is achieved by this architecture in two ways: by allowing the user to configure the effective output width of each array, and by allowing the user to combine arrays to implement larger memories. First consider the effective output width of each array. Each array has a nominal width of $w$ and depth of $\frac{b}{nw}$. This nominal aspect ratio can be altered by the level 1 (L1) data mapping block. Figure 2(a) shows an example L1 data mapping block in which $w = 8$. Each dot represents a pass-transistor switch. In this example, the set of allowable effective output widths, $w_{\text{eff}}$, is $\{1, 2, 4, 8\}$, meaning each array can be configured to be one of $\frac{b}{n}$x1, $\frac{b}{2n}$x2, $\frac{b}{4n}$x4, or $\frac{b}{8n}$x8. Figure 4(b) shows two sets of switches, $A$ and $B$, that are used to implement the $\frac{b}{4n}$x4 configuration. One of the memory address bits is used to determine which set of switches, $A$ or $B$, is turned on. Each set of switches connects a

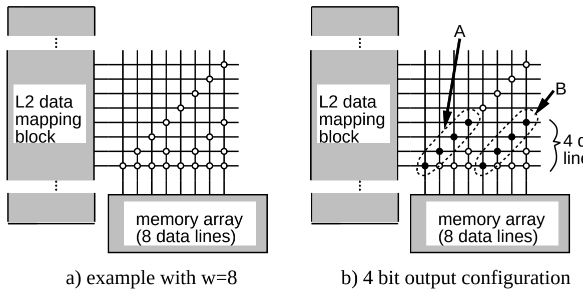


Figure 2: L1 data mapping block

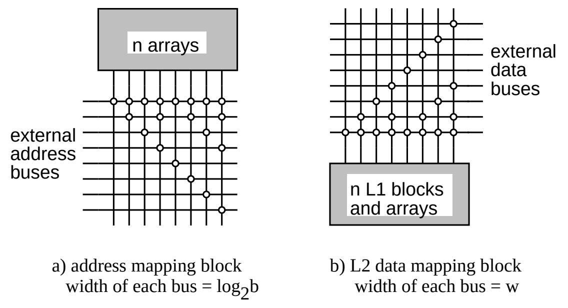
different portion of the memory array to the bottom four data lines.

Notice that the mapping block need not be capable of implementing all power-of-two widths between 1 and $w$. By removing every second switch along the bottom row of the block in Figure 2(a), a faster and smaller mapping block could be obtained. The resulting mapping block would only be able to provide an effective data width of 2, 4, or 8, however, meaning that the resulting architecture would be less flexible. Section 4 examines the impact of removing L1 data mapping block switches on area, speed, and flexibility.

Memory flexibility is also obtained by allowing the user to combine arrays to implement larger memories. Figure 3(a) shows how four 1024x8 arrays can be combined to implement a 1024x32 logical memory. In this case, a single external address bus is connected to each array, while the data bus from each array is connected to separate external data buses (giving a 32-bit data width). Each L1 data mapping block connects 8 array data lines directly to the 8 L1 outputs.

Figure 3(b) shows how this architecture can be used to implement a configuration containing two logical memories: one 24576x1 and one 2048x4. The three arrays implementing the 24576x1 memory are each configured as 8192x1 using the L1 data mapping block, and each output data line is connected to a single external data line using bidirectional pass transistors. Two address bits control the pass transistors; the value of these address bits determine which array drives (or is driven by) the external data line. The 2048x4 memory can be implemented using the remaining array, with the L1 block configured in the "by 4" mode.

The topology of the switches in the level 2 (L2) data mapping block and the address mapping block determine to what extent the arrays can be combined. If both of these mapping blocks are fully populated, meaning any external bus (both address and data) can be connected to any array, a very flexible, but slow, architecture would result. As a compromise between speed and flexibility, the switch topologies in Figure 4 will be used in this paper. In this figure, each dot



n arrays
external address buses
a) address mapping block width of each bus = log2b
external data buses
n L1 blocks and arrays
b) L2 data mapping block width of each bus = w

Figure 4: Level 2 data and address mapping block topology ($n = m = r = 4$)

represents a set of switches controlled by a single programming bit, one switch for each bit in the bus ($w$ in the L2 data mapping block and $log_2 b$ in the address mapping block) [13]. This topology can support almost all required mappings as long as the mapping algorithm is free to set the external bus and array assignments, but, because there are fewer switches than in a fully populated block, the delay is less than in the fully populated case. In Section 4, $n$ will be varied; it is simple to extend the same basic pattern to an array of any width.

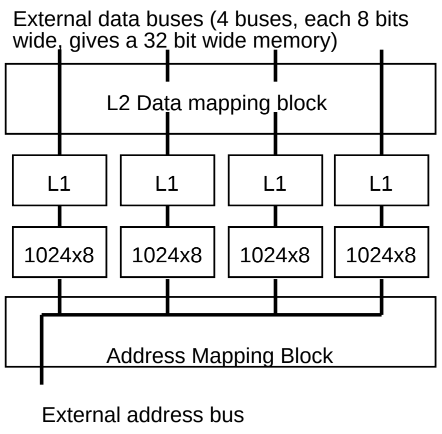
In addition to address and data lines, write enable signals are required for each array. The write enable lines can be switched in the L2 mapping block just as the data lines are. In order to correctly update arrays for effective widths less than the nominal width, we assume that the arrays are such that each column in the array can be selectively enabled. The address bits used to control the L1 mapping block can be used to select which array column(s) are updated.

## 3 Evaluation of Memory Architectures

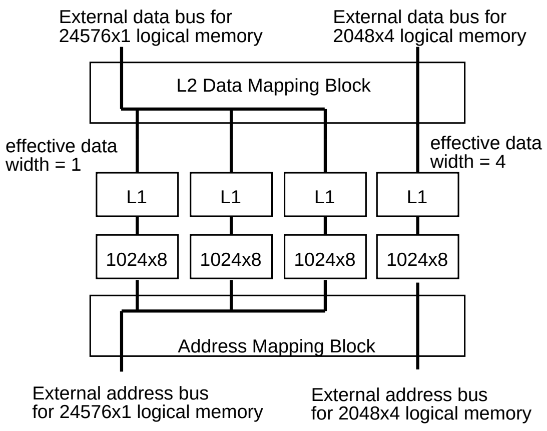
This section describes how the set of architectures obtained by varying the parameters discussed in Section 2 can be compared on the basis of speed, area, and flexibility.

One way to compare architectures would be to gather a set of benchmark circuits (each containing memory), and attempt to map the logical memory configuration of each circuit to each architecture. Each mapping attempt may or may not be successful. A flexibility measure could be obtained by counting the number of successful mappings to each architecture, and the architecture with the highest count would be deemed the most flexible. Detailed access time and area models can be used to estimate the access times and chip area of each memory implementation.

The problem with this approach stems from the fact that circuits typically have only a few logical memories. This is in contrast to previous studies on



a) 1024x32


External data bus for 24576x1 logical memory
External data bus for 2048x4 logical memory
L2 Data Mapping Block
effective data width = 1
effective data width = 4
L1
L1
L1
L1
1024x8
1024x8
1024x8
1024x8
Address Mapping Block
External address bus for 24576x1 logical memory
External address bus for 2048x4 logical memory

b) 24576x1 and 2048x4

Figure 3: Two example mappings

logic block architectures, where each circuit contains enough logic blocks that, even for a moderate number of benchmark circuits, hundreds (or thousands) of logic blocks will be used. Thus, all architectural features of the logic block are thoroughly exercised. This isn't the case with memory; to adequately exercise each configurable memory architecture, thousands of logical memory configurations would be required. Clearly, it isn't feasible to gather that many benchmark circuits.

As an alternative, we have developed a "logical memory configuration generator" that generates logical memory configurations randomly, constrained by the set of parameters shown in Table 3. This table also gives the parameter values we have used to gather all the results in this paper. Each configuration is generated as follows. First, the number of logical memories is randomly chosen (each number between 1 and $n_{gen}$ is equally likely). Then, for each logical memory, a width between $w_{min}$ and $w_{max}$ and depth between $d_{min}$ and $d_{max}$ are selected. The parameter $\alpha$ is used to indicate what proportion of the generated dimensions are a power of two. We have chosen $\alpha = 0.8$; this means 80% of the depths and 80% of the widths generated are a power of two (all powers-of-two between $w_{min}$ and $w_{max}$ or $d_{min}$ and $d_{max}$ are equally likely).

Once the dimensions of all $n_{gen}$ memories have been chosen, the total number of bits is compared to $b_{gen}$, and if it is larger, a completely new set of dimensions is chosen (for the same number of logical memories). This is repeated until the total number of bits is less than $b_{gen}$. To gather all results in the next section, 10,000 logical memory configurations were generated.

To map a logical memory configuration onto an architecture, an algorithm that assign arrays, address buses, and data buses to each logical memory is required. If the mapping blocks are fully populated, that is, any external bus can be connected to any array, the mapping problem is easy. However, for architectures with mapping blocks similar to Figure 4, the task is much less straightforward. Such an algorithm was developed, and is described in [14].

To compare implementations in terms of speed and area, detailed access time and area models are needed. The access time model used in this study was modified from a detailed cache access time model [15]. It contains terms for the delays due to the decoder, word lines, bit lines, column multiplexors, and sense amplifiers, as well as routing. The area model is based on a cache area model [16]. Area measurements are given in *memory bit equivalents* or *mbe's*; one mbe is equal to the size of one memory cell in an SRAM array (1 mbe = 0.6 rbe in [16] $\approx 250um^2$ in a 0.8um CMOS process).

As described earlier, flexibility is also an important metric. Each attempt to map a logical memory configuration to an architecture might or might not be successful. There are several reasons why an attempt might *not* be successful: the architecture might not contain enough bits, it might not have enough data lines, the mapping blocks might not be flexible enough to combine arrays in such a way that the configuration can be mapped, or the granularity of the arrays might be such that too many bits are wasted (the last of these reasons will be described in more detail in Section 4). A measure of flexibility can be obtained by counting the number of successful mappings to each architecture, and by using the following definition:

$$\text{flexibility} = \frac{\text{Number of configurations successfully mapped}}{\text{Number of configurations attempted}}$$

Using this definition, an architecture that is better able to adapt to the generated logical memory configurations has a higher flexibility.

| Param. | Meaning | Setting |
|---|---|---|
| $b_{gen}$ | Maximum number of bits per configuration | 65536 |
| $n_{gen}$ | Maximum number of logical memories | 4 |
| $w_{min}$ | Minimum width of each logical memory | 1 |
| $w_{max}$ | Maximum width of each logical memory | 64 |
| $d_{min}$ | Minimum depth of each logical memory | 16 |
| $d_{max}$ | Maximum depth of each logical memory | 65536 |
| $\alpha$ | Proportion of dimensions that are a power of two | 0.8 |

Table 3: Parameters for workload generator

## 4 Experimental Results

In this section, the effects of varying the number of memory arrays and the topology of each L1 data mapping block on access time, chip area, and flexibility are shown.

### 4.1 Number of Memory arrays

This subsection examines how the number of memory arrays ($n$) affects the delay, area, and flexibility of an FCM architecture. For all architectures in this section, both the number of external data buses ($m$) and the number of external address buses ($r$) are set to 4; thus, all architectures can implement at most 4 logical memories. In addition, the nominal data width of each array is fixed at 16, and the set of allowable effective data widths is $w_{\text{eff}} = \{1, 2, 4, 8, 16\}$. Four memory sizes (parameter $b$) from 8Kbits to 64Kbits were considered. For each memory size, the number of arrays was varied from 4 to 64 (a greater number of arrays implies that each array is smaller, since the total number of bits is kept constant). Figure 5(a) shows that as the number of arrays increases, the chip area required to implement the configurable memory increases. This is due to the need for more decoders, drivers, sense amplifiers, and mapping blocks.

Figure 5(b) shows the effect on the average logical memory access time (in the 0.8um CMOS process used in [15]). As the number of arrays increases, the delay due to the mapping blocks increases. However, smaller arrays are faster (due to the shorter wordlines, bitlines, and the smaller decoder). The two competing trends cause a minimum in the access time graph, which is especially clear in the 64Kbit case.

Figure 6 shows the effect of changing the number of arrays on the flexibility of the configurable memory (in order to focus on the dependency of flexibility on the number of arrays, only the 64Kbit results are shown). The vertical scale is the proportion of test configurations that could be successfully mapped. As mentioned in Section 3, one of the reasons a mapping might be unsuccessful is that the granularity of the arrays is too course. Since each array can be connected to at most one address bus, an array can not be shared between two logical memories. Thus, if a logical memory uses only half an array, the remainder is wasted. If the arrays are only half the size, however, those bits would be available to implement another logical memory (recall that each logical configuration contains several logical memories). As the graph shows, the increase in flexibility is not significant, especially past eight arrays. Thus, the access time and area should be of primary concern when choosing a value for $n$.

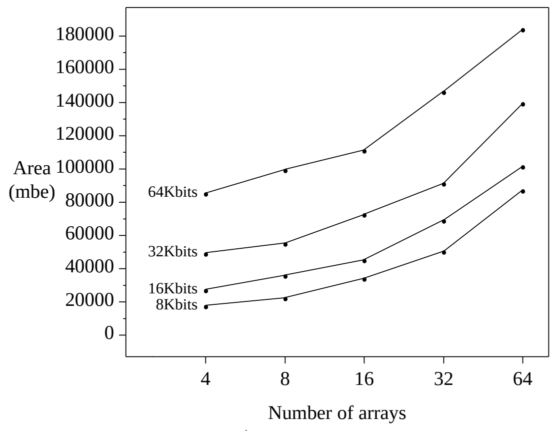
The results in Figures 5 and 6 apply only to the architecture parameters described above and the workload parameters in Table 3. For these parameters, the graphs suggest that the memory should be broken into eight blocks. Although these results are specific to the parameters shown, we believe that the trends shown in the graphs apply over a wide range of architecture and workload parameters.

It is important to point out that in order to concentrate on how the memory granularity affects flexibility, we have fixed the number of external buses ($r$ and $m$) in this experiment. If these parameters were allowed to increase with $n$, architectures with more arrays would be able to implement configurations with more logical memories, and would thus be more flexible.

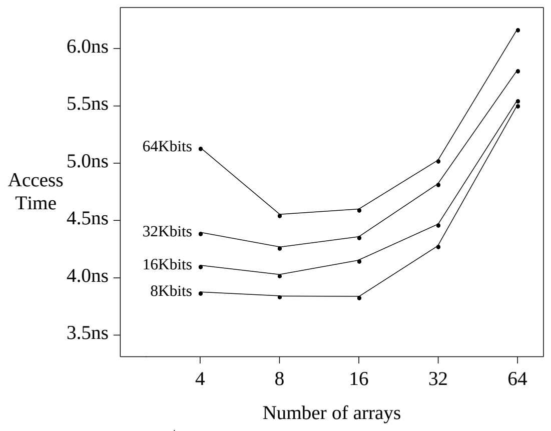
### 4.2 L1 Data Mapping Block Capability

As described in Section 2, the effective data width of each array can be set by configuring the L1 data mapping blocks. In the previous section, it was assumed that the set of effective output widths, $w_{\text{eff}}$, was {1,2,4,8,16}. Section 2 described how a faster, but less flexible architecture could be obtained by removing some of the capability of the L1 data mapping block. In this section, we investigate the effects of changing the minimum effective data width (smallest value of $w_{\text{eff}}$). Intuitively, the higher the minimum data width, the less flexible the L1 mapping block, and hence, the less flexible the architecture.

Figure 7 shows how the minimum L1 data width



a) area



b) access time

Figure 5: Effects of changing number of arrays ($n$) on delay and area

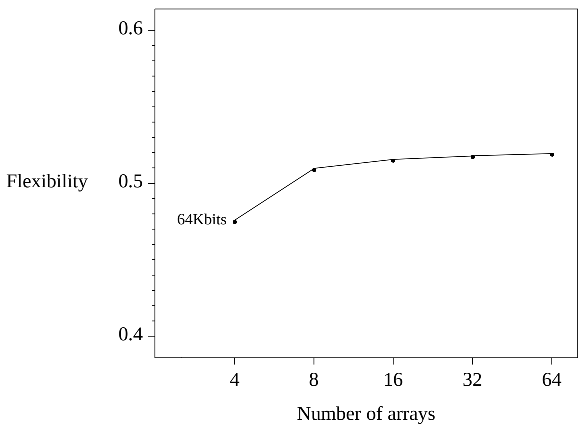


Figure 6: Effects of changing number of arrays ($n$) on flexibility

affects the average access time and area of each architecture for three values of $n$ ($b$ is fixed at 64Kbits). As the graphs show, removing switches from the L1 mapping block has almost no effect on area; this isn't surprising, since the switches are small and are controlled by a small number of programming bits. The average access time, however, decreases noticeably as switches are removed; the marked decrease at 16 is because if the minimum output width is 16, then no L1 mapping block is needed at all. Figure 8 shows that removing switches has a devastating effect on flexibility. Not only is the "by 1" configuration good for logical memories with width 1, but also for logical memories with odd output widths (a 3-bit wide memory can be implemented using 3 arrays in the "by 1" configuration). It is clear that the decrease in delay as the minimum L1 mapping block output width is increased does not make up for the loss in flexibility. Therefore, for this class of architectures, all possible power of two widths should be included in $w_{\text{eff}}$.

## 5 Conclusions

In this paper, we have presented a family of centralized field-configurable memory architectures. Each architecture is composed of a number of arrays, and a data and address mapping network. The architectures are flexible enough that they can implement logical configurations containing several logical memories of different depths and widths.

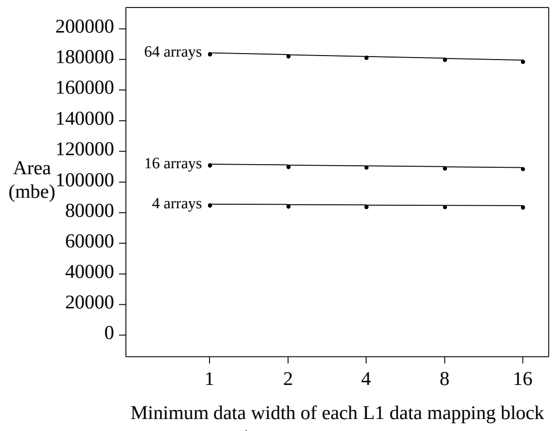
An important contribution of this work is the method used to analyze the architectures. We believe that by generating a large number of test configurations stochastically, we can obtain results that are much more useful than those that could be obtained using a handful of "real" circuits.

Although this paper has concentrated on centralized architectures, we believe that the evaluation method we have proposed is also useful when comparing distributed architectures. When investigating distributed architectures, however, it is necessary to take into account routing beyond the memory "boundaries" and also to consider logic block and memory array placement. Clearly, this is an exciting area of future research.

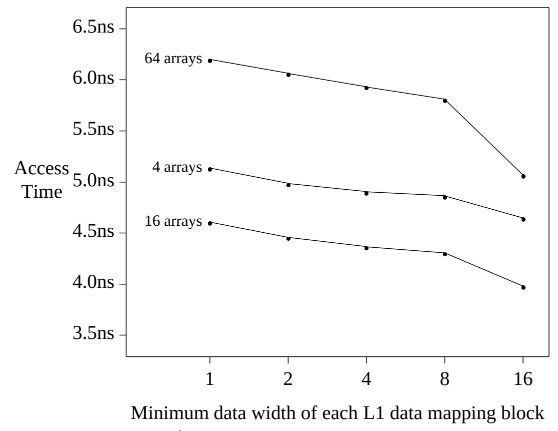
## References

[1] G. Feygin, P. Chow, P. G. Gulak, J. Chappel, G. Goodes, O. Hall, A. Sayes, S. Singh, M. B. Smith, and S. Wilton, "A VLSI Implementation of a Cascade Viterbi Decoder with Traceback," in *1993 IEEE International Symposium on Circuits and Systems*, pp. 1945–1948, May 1993.

[2] I. Agi, P. J. Hurst, and K. W. Current, "A 450 MOPS image backprojector and histogrammer," in



a) area



b) access time

Figure 7: Effects of changing minimum output width of L1 Data Mapping Block on delay and area

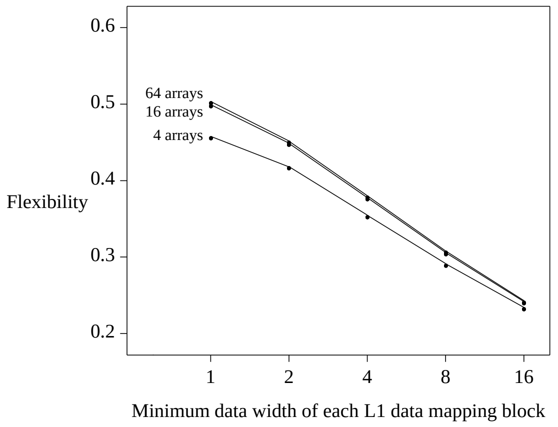


Figure 8: Effects of changing minimum output width of L1 Data Mapping Block on flexibility

*IEEE 1992 Custom Integrated Circuits Conference*, pp. 6.2.1–6.2.4, May 1992.

[3] R. Mason and K. Runtz, "VLSI neural network system based on reduced interchip connectivity," in *Canadian Conference on Very Large Scale Integration*, pp. 7.69–7.74, Nov. 1993.

[4] H. Satoh, T. Nishimura, M. Tatsuki, A. Ohba, S. Hine, and Y. Kuramitsu, "A 209K-transistor ECL gate array with RAM," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1275–1279, Oct. 1989.

[5] K. Kaneko, T. Okamoto, M. Nakajima, Y. Nakakura, S. Gokita, J. Nishikawa, Y. Tanikawa, and H. Kadota, "A VLSI RISC with 20-MFLOPS peak, 64-bit floating point unit," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1331–1339, October 1989.

[6] J. Rose, R. Francis, D. Lewis, and P. Chow, "Architecture of programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 1217–1225, October 1990.

[7] H.-C. Hsieh, W. S. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey, and R. Kanazawa, "Third-generation architecture boosts speed and density of field-programmable gate arrays," in *1990 Custom Integrated Circuits Conference*, pp. 31.2.1–31.2.7, 1990.

[8] R. H. Krambeck, C. Chen, and R. Tsui, "ORCA: A high speed, high density FPGA architecture," in *Compcon Spring '93*, pp. 367–372, February 1993.

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

[10] Plus Logic, *FPSL5110 Product Brief.*

[11] D. E. Smith, "Intel's FLEXlogic FPGA architecture," in *Compcon Spring '93*, pp. 378–384, February 1993.

[12] K. Kawana, H. Keida, M. Sakamoto, K. Shibata, and I. Moriyama, "An efficient logic block interconnect architecture for user-reprogrammable gate array," in *IEEE 1990 Custom Integrated Circuits Conference*, pp. 31.3.1–31.3.4, May 1990.

[13] T. Ngai, "An SRAM-programmable field-reconfigurable memory," Master's thesis, University of Toronto, 1994.

[14] S. J. E. Wilton, *Architecture of Field-Configurable Memory.* PhD thesis, University of Toronto, in progress.

[15] S. J. E. Wilton and N. P. Jouppi, "An access and cycle time model for on-chip caches," Tech. Rep. 93/5, Digital Equipment Corporation Western Research Lab, 1994.

[16] J. M. Mulder, N. T. Quach, and M. J. Flynn, "An area model for on-chip memories and its application," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 98–106, Feb. 1991.