A TWO-LEVEL RECONFIGURABLE CELL ARRAY FOR DIGITAL SIGNAL PROCESSING

By

MITCHELL JOHN MYJAK

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2004

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of MITCHELL JOHN MYJAK find it satisfactory and recommend that it be accepted.

Chair

Acknowledgment

This research was supported in part by the EECS Alumni Fellowship from Washington State University and by the U.S. Department of Homeland Security Graduate Fellowship.

Special thanks to my advisor, Dr. José Delgado-Frias, for all his help and support.

A TWO-LEVEL RECONFIGURABLE CELL ARRAY FOR DIGITAL SIGNAL PROCESSING

Abstract

by Mitchell John Myjak, M.S.
Washington State University
May 2004

Chair: José G. Delgado-Frias

Reconfigurable hardware has become an attractive option for implementing digital signal processing, especially in systems that require both high performance and flexibility. This thesis presents a novel two-level reconfigurable architecture targeted toward systems with these requirements. The architecture supports a large orthogonal design space whereby designers can customize the word length, amount of parallelism, number of functional units, and functional unit connectivity to meet the needs of the application.

On the upper level, algorithms are mapped onto an array of 4-bit cells and a hierarchical interconnection fabric. The interconnection structure contains a mesh of 4-bit busses for local data transfer, as well as an H-tree for communicating results between functional units. On the lower level, each cell contains a small matrix of elements that collectively implement all necessary operations. The matrix of elements has only two configurations: one optimized for mathematical functions such as multiply-accumulates, and the other optimized for memory operations. The system also contains pipeline latches to maximize clock rate and throughput.

Circuit simulations indicate that the architecture achieves a clock frequency of 200 MHz in a modest 0.25-μm CMOS technology. An initial prototype of the reconfigurable cell has been fabricated in 0.5-μm CMOS and tested for functionality. The estimated execution time for a 16-bit, 256-point Fast Fourier Transform shows a speedup ranging from 1.6 to 14 compared to contemporary digital signal processors.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many digital systems rely on digital signal processing (DSP) to achieve their functionality. For example, cellular phones use sophisticated compression and encryption algorithms to transmit data securely over a wireless link. Digital multimedia devices such as video cards and CD players translate a stream of bits into images or music. Even hearing aids may implement complex digital filters to enhance speech.

Reconfigurable hardware has become an attractive option for implementing DSP, especially in applications that must combine high performance and flexibility. Specialized applications that require low power consumption and/or fault tolerance can also use this approach to meet specifications. The following sections discuss the main requirements of DSP systems and compare reconfigurable hardware to other alternatives.

1.1 Design Metrics of DSP Hardware

Although DSP encompasses a wide range of applications, a number of common metrics for DSP hardware can be identified:

- **Performance:** DSP places great demands on the processing power of any hardware implementation. For example, a 512-point Fast Fourier Transform (FFT) requires around 16,000 multiplications and 9,000 additions [1]. Algorithms typically work with data in vector or matrix form, so the hardware must apply the same basic operation to multiple data points. Hence, the standard metric of performance is not latency, but rather total execution time or its reciprocal, throughput. Hardware implementations that exploit the parallelism of DSP algorithms will achieve much higher throughput.

- **Flexibility:** For commercial products, the total cost clearly influences the implementation strategy chosen. Using commodity devices eliminates the need to design, fabricate, and test custom hardware. Devices with high flexibility can be used in a large number of applications to lower development costs.

- **Power consumption:** In recent years, the application space of DSP has shifted to include wireless and mobile computing. As a result, power consumption is a crucial design metric for many DSP systems today. This evolution requires novel hardware architectures to meet the new demands and challenges.

- **Fault tolerance:** Hardware used for mission-critical applications, such as communication satellites and real-time monitoring equipment, must contain mechanisms to detect and handle faults. Radiation-induced errors, such as latch-up, burn-out, and single event upsets, are of major concern in environments with high background radiation, such as space [2]. Memory elements are particularly vulnerable to single event upsets, which occur when a charged subatomic particle causes a transient voltage spike that subsequently changes the state of the circuit.

Most applications require a balance between two or more of these metrics. Hence, the ability of DSP hardware to meet the particular needs of an application is another key factor influencing the design choice.

Table 1.1: Comparison of DSP implementations

Device	Performance	Flexibility	Power	Fault Tolerance
General-purpose processor	Low	High	Medium	None
Digital signal processor	Medium	Medium	Medium	None
Configurable processor	Medium	Medium	Med-Low	Possible
Reconfigurable hardware	Med-High	High-Med	Med-Low	Inherent
ASIC	High	Low	Low	Possible

1.2 DSP Implementations

Digital systems may use a variety of components to perform DSP, ranging from application-specific integrated circuits (ASIC) to general-purpose microprocessors [3]. Table 1.1 provides a comparison of these approaches in terms of the four metrics described above [4].

General-purpose processors can execute a wide variety of software programs, including DSP algorithms. However, their performance may not meet the requirements of the application [4]. Specialized digital signal processors include some instructions tailored for DSP computations. They generally achieve better performance than their general-purpose counterparts, but their architecture may not be optimized for the different requirements that DSP applications may have, such as speed, power, and word length. In addition, fault-tolerant processors are generally not commercially available.

Configurable processors have a customizable instruction set, datapath, and memory organization. Devices of this type are configured for a particular application prior to fabrication [5]. However, each configuration requires a new compiler to generate optimal code. In addition, the use of such a processor may be limited to a specific application, so this approach does not achieve as high flexibility as other alternatives.

Reconfigurable hardware allows designers to change the configuration of the hardware at any time. This approach provides an excellent alternative for performance, flexibility, power, and fault tolerance [6]. Users may also select between different trade-offs, such as performance versus fault tolerance, depending on the application at hand. The following section discusses reconfigurable hardware in greater detail.

Finally, application-specific integrated circuits are optimized for a particular DSP algorithm. These devices can achieve maximum performance and minimum power consumption, but incur high development costs. Due to the cost and limited applicability of an ASIC, this approach is only feasible for high-volume applications in general.

1.3 Reconfigurable Hardware

Reconfigurable hardware attempts to combine the performance of an ASIC with the flexibility of a microprocessor. This approach has recently become practical for DSP, due to the increasing capabilities of VLSI systems. In general, reconfigurable devices contain an array of programmable cells and interconnections. DSP algorithms are divided into small portions and mapped onto the structure. Unused portions of the hardware can be disabled to lower the total power consumption. Since the hardware configuration can be changed at any time, even after deployment, reconfigurable hardware achieves great flexibility [6]. In addition, the design process can be automated using appropriate software tools [7],[8]. Finally, reconfigurable hardware possesses a certain degree of fault tolerance, in that DSP algorithms can be remapped around faulty cells if the circuit is damaged.

Traditional reconfigurable devices such as field-programmable gate arrays (FPGA) place little functionality in the cells [9]. These fine-grain devices work well for implementing combinational or sequential logic. However, DSP uses mathematical operations such as multiplication extensively. Unless the architecture contains dedicated hardware for this purpose, mapping a multiplier onto a fine-grain device creates a complex structure that yields poor performance [10],[11].

Recently, researches have proposed new reconfigurable devices that incorporate adders, multipliers, lookup tables, and other functional units in the cells [12],[13],[14]. In some respects, these coarse-grain devices are successors to the older reconfigurable systolic array architectures, as in [15]. In general, coarse-grain reconfigurable hardware achieves good performance for mathematical functions, but may not implement all the control logic necessary for DSP. The fixed number of functional units also limits flexibility.

This thesis describes a novel medium-grain reconfigurable architecture for DSP [16], [17], [18], [19]. In this approach, each cell contains a 4×4 matrix of reconfigurable elements. Each element, in turn, consists of a small random-access memory. The matrix of elements can be configured into two structures: one optimized for mathematical functions and the other for memory operations. In mathematics mode, each element acts as a lookup table that allows the cell to implement many 4-bit functions. In memory mode, the matrix of elements operates as a 64-byte memory. The resulting two-level architecture can perform the wide range of operations required for DSP.

The remainder of this thesis is organized as follows. Chapter 2 describes the upper level of the architecture and explains how various operations can be mapped onto the array of cells. Chapter 3 covers the lower level, showing how the matrix of elements allows cells to implement various 4-bit functions. Chapter 4 considers the circuit schematic of an element and its corresponding VLSI implementation. Circuit simulations of the reconfigurable cell appear in Chapter 5, along with measurements taken from a prototype device. Chapter 6 focuses on the interconnection structure used in the architecture to group cells into functional units and functional units into algorithms. In Chapter 7, the execution times of several benchmark algorithms are computed and compared to current digital signal processors. Finally, Chapter 8 provides some concluding remarks.

Chapter 2

Upper-Level Organization

At the upper level, the two-level architecture consists of an array of reconfigurable cells and interconnection structures. This chapter describes the array of cells and demonstrates how cells can be grouped into functional blocks to implement basic operations, such as multiplication and addition. The motivation for this discussion is twofold: to demonstrate that the architecture can implement these operations efficiently, and to identify the functionality that each cell must contain. For now, assume that each cell can implement any 4-bit operation and has unlimited communication bandwidth to neighboring cells.

2.1 Cell Array

Figure 2.1 illustrates a portion of the reconfigurable cell array. Each cell performs operations in 4-bit units. The use of 4-bit cells gives designers control over the word length and maximizes the utilization of the device [20]. Having larger cells would increase the fan-in and fan-out of the gates, create signal integrity problems, and impede the datapath. As described in Chapter 6, a mesh of 4-bit busses connects neighboring cells horizontally and vertically. Additional busses allow data to be routed between non-adjacent cells.

As shown in Figure 2.2, each cell contains four components. The processing core imple-

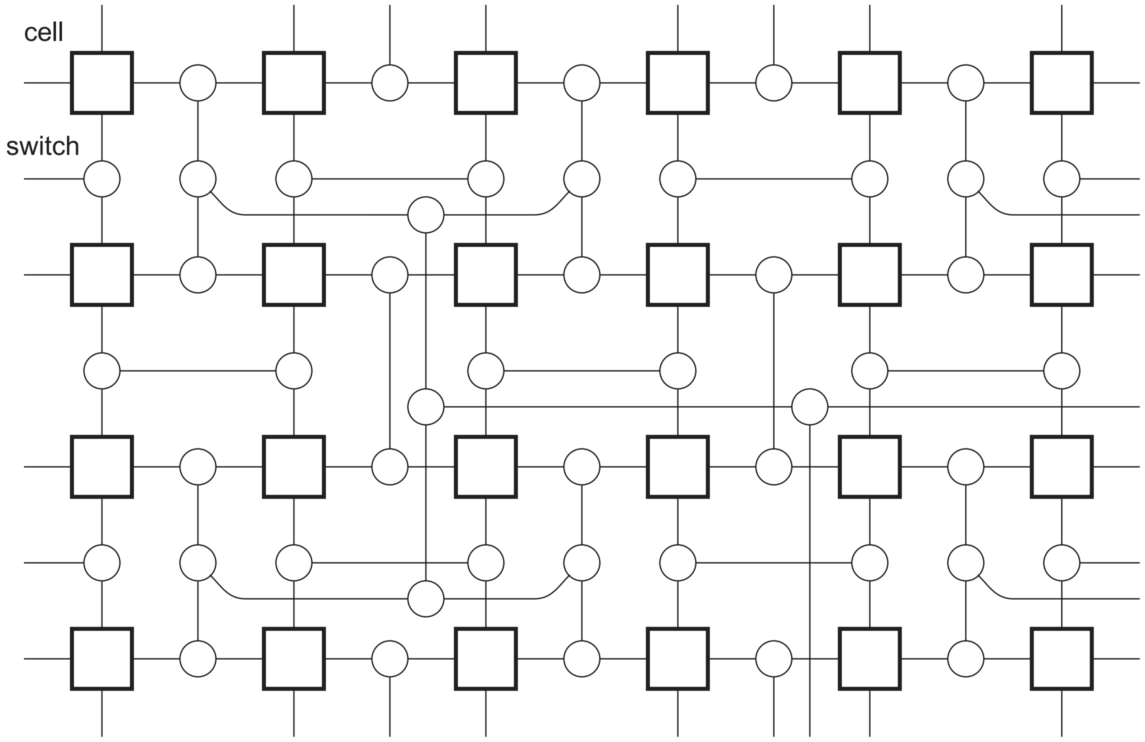


Figure 2.1: Array of cells in reconfigurable architecture

ments the 4-bit operations required for DSP. This component can perform both mathematics and memory operations. The two switches connect the inputs and outputs of the processing core to the interconnection network. Data latches between the switches and the processing core pipeline the execution cycle. Finally, the control module generates control signals for the processing core and manages the reconfiguration process.

2.2 Clock Approach

Figure 2.3 summarizes the clocking scheme used in the cell. In the first clock phase, the cell precharges the processing core and enables the two switches. The values in the output latches flow through the output switch onto the interconnection network. At the destination cell, the values pass through the input switch and are stored in the input latches. In the second clock phase, the cell precharges the two switches and enables the processing core.

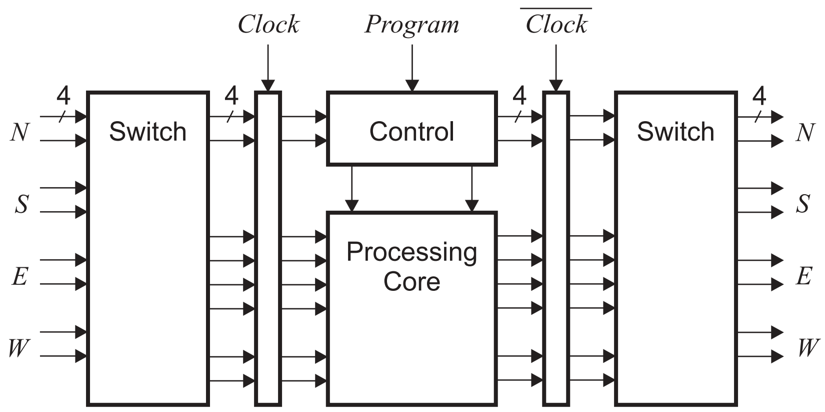


Figure 2.2: Components of cell

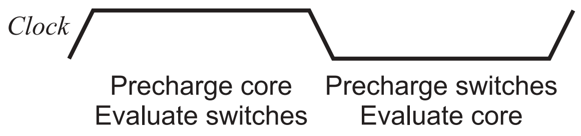


Figure 2.3: Operations performed during each phase of clock

The processing core evaluates the desired operation, and the results are placed in the output latches.

Besides isolating the two phases of the clock, the latches in the cell allow DSP algorithms to exploit the benefits of pipelining [21]. Without pipelining, the system clock rate would depend on the word length and operation type of each part of the algorithm. With pipelining, the only restriction on the clock rate is the propagation delay through one cell. Depending on the requirements of the algorithm, each data line can be configured to go through one or several latches.

The remainder of this chapter describes how groups of cells can implement the basic operations required in DSP, including multiplication, addition, memory operations, and control logic.

2.3 Unsigned Multiplication

Almost all DSP algorithms use multiplication of some form. Depending on the target application, the algorithm may require unsigned or signed multiplication of 16-bit, 20-bit, 24-bit, 32-bit, or larger numbers. The use of 4-bit cells enables applications to implement a multiplier of the precise size required, while exploiting the inherent parallelism of the operation.

Suppose the reconfigurable device must multiply two unsigned 16-bit numbers A and B to generate a 32-bit output Y. The unit is to operate in parallel for maximum performance. Two options for mapping the multiplier onto the array of cells are now discussed.

2.3.1 Carry-Save Multiplier

A straightforward solution, outlined in Figure 2.4, implements a carry-save multiplier [22] with 4-bit cells. Note that A and B are transferred across entire columns and rows of cells, respectively. This multiplier requires twenty cells: four that perform multiplication, four that perform addition, and twelve that perform both operations. The critical path involves eight cells. A typical cell multiplies two 4-bit portions of the inputs, say a and b, and may add two 4-bit terms to the result, say c and d. Denoting the result as y, each cell performs the operation

$$y_{7:0} = (a_{3:0} \times b_{3:0}) + c_{3:0} + d_{3:0}. \tag{2.1}$$

The upper and lower halves of the result connect to the c and d inputs of neighboring cells.

2.3.2 Improved Multiplier

By rearranging the interconnection structure, it is possible to reduce the hardware required. Figure 2.5 illustrates an improved multiplier that uses sixteen cells and has a critical path of seven cells. The interconnection scheme scales easily to form n-bit multipliers with $(n/4)^2$ cells (assuming n is a multiple of 4). Although the clock scheme used in the reconfigurable

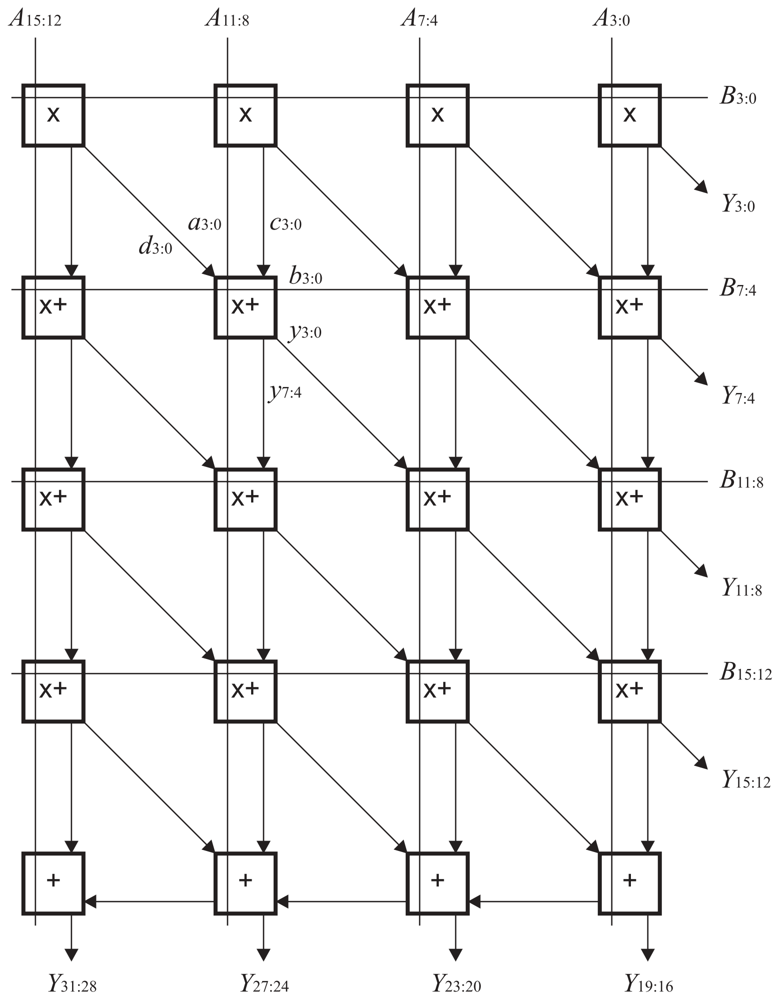


Figure 2.4: Diagram of 16-bit carry-save multiplier

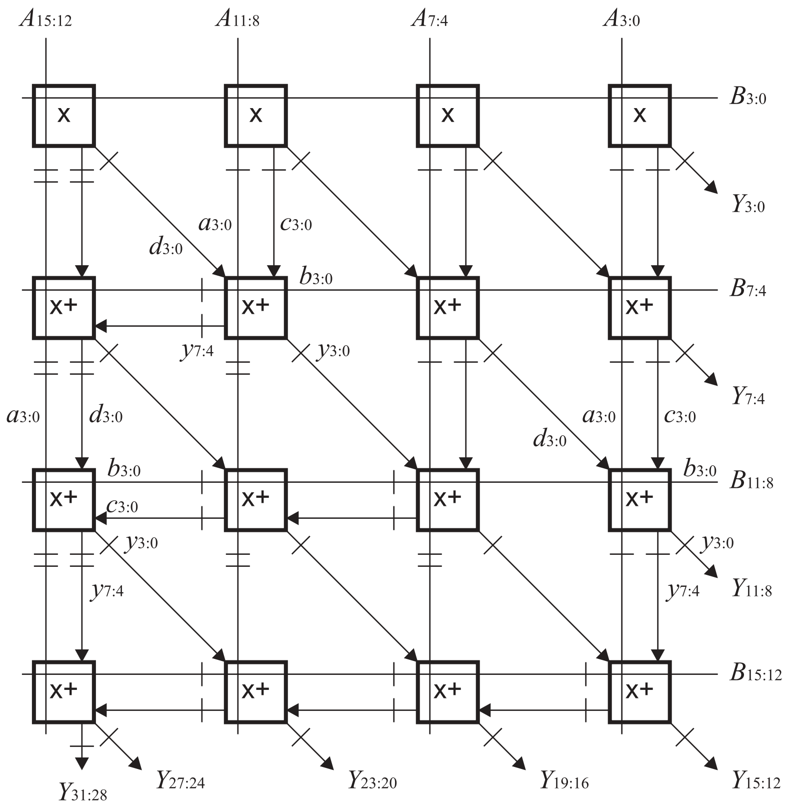


Figure 2.5: Diagram of modified 16-bit multiplier

architecture automatically pipelines the multiplier into 4-bit portions, some minor adjustments should be made so that the structure fully exploits the benefits of pipelining. The hash marks in the figure indicate the number of pipeline stages that must separate each cell so that intermediate results arrive at the next cell at the proper times. With these modifications, the multiplier has a latency of seven clock cycles, but can initiate one operation per cycle. The least significant four bits of the output are generated during the first clock cycle, the next four bits in the second cycle, and so forth.

2.4 Multiply-Accumulate

The top row of cells in Figure 2.5 performs multiplication but not addition. If these cells also evaluated the expression in (2.1), the multiplier could add two additional 16-bit terms

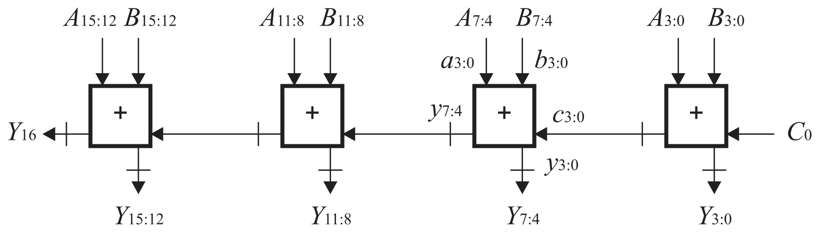


Figure 2.6: Diagram of 16-bit adder

to the result. This modification would create a powerful multiply-accumulate (MAC) unit that calculated the formula

$$Y_{63:0} = (A_{31:0} \times B_{31:0}) + C_{31:0} + D_{31:0}. \tag{2.2}$$

2.5 Addition

Most DSP algorithms require addition as well as multiplication. In many cases, an addition may be combined with a multiplication and implemented with the MAC unit described previously. For example, the difference equation used in digital filters is amenable to this simplification. However, some algorithms still require dedicated adders.

The structure in Figure 2.6 uses four cells to add two 16-bit numbers A and B. Each cell adds two four-bit portions of the inputs as well as a carry in:

$$y_{4:0} = a_{3:0} + b_{3:0} + c_0. \tag{2.3}$$

The carry out of the last stage is discarded for simplicity. In general, adding or subtracting n-bit numbers requires $n/4$ cells (again assuming that n is a multiple of four).

The adder uses pipelining for maximum performance. Note that the inputs must arrive in a staggered fashion, starting with the least significant four bits. Many of the units described in this chapter impose similar requirements on the inputs.

2.6 Two's-Complement Multiply-Accumulate

DSP algorithms generally work with both positive and negative numbers, so it is reasonable to expect that applications may require two's-complement multiplication and addition. As described in this section, the same multiplier structure can be used to perform this operation, except that some cells use different data formats.

First, recall from (2.1) that each cell in the unsigned MAC unit evaluates the 4-bit MAC function

$$y_{7:0} = (a_{3:0} \times b_{3:0}) + c_{3:0} + d_{3:0}.$$

Figure 2.5 illustrates how these 4-bit terms are defined for various cells in the design. For consistency, c always appears to the left of d in the diagram.

Now consider a two's-complement MAC unit that handles 16-bit inputs in 4-bit portions. From the properties of two's-complement numbers, the most significant 4-bit portion has two's-complement format, but the remaining portions have unsigned format. Hence, many of the cells will still operate on unsigned inputs. Figure 2.7 depicts the data formats in the two's-complement MAC unit. Solid lines denote unsigned data; dashed lines denote two's-complement data.

Observe that some cells generate two's-complement outputs, whereas other cells do not. In fact, the two's-complement MAC unit contains seven types of cells, labeled A through H in the figure (G is missing for technical reasons). The A cells simply evaluate the unsigned MAC function in 2.1. However, the B cell must multiply the two's-complement portion of A with an unsigned portion of B. The cell also adds two's-complement portions of C and D to the result. In order to represent the entire range of valid outputs, the B cell must generate an 8-bit output y whose upper 4 bits and lower 4 bits are both two's-complement numbers. This data format is unusual, but is the best choice for representing the result. In fact, one

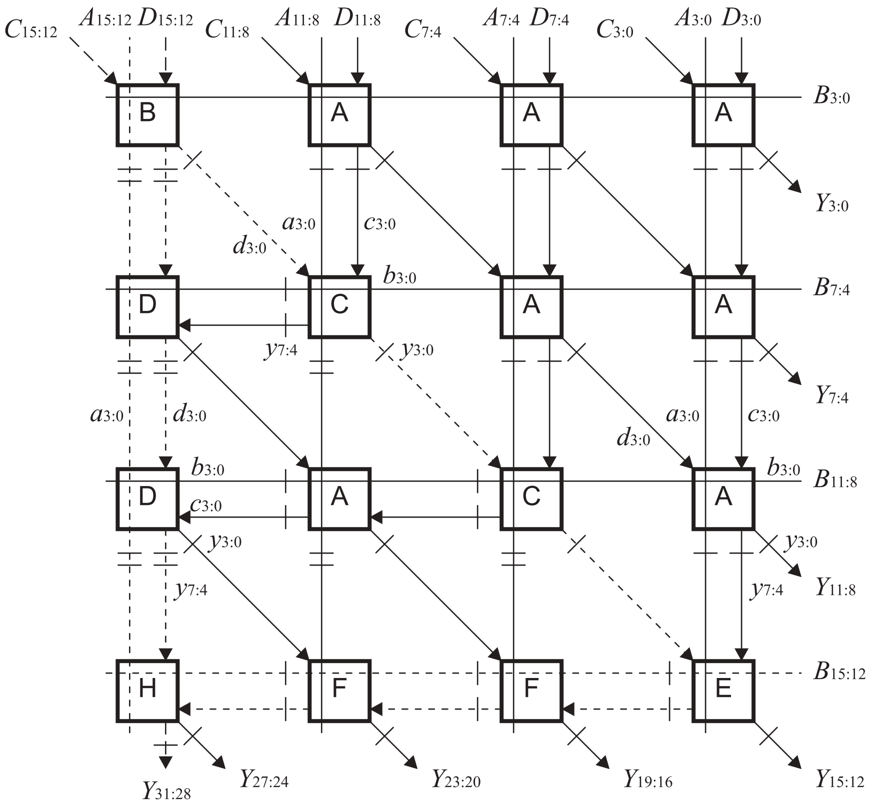


Figure 2.7: Diagram of 16-bit two's-complement MAC unit

Table 2.1: Example calculations of B cell in two's-complement MAC unit

$a_{3:0}$	$b_{3:0}$	$c_{3:0}$	$d_{3:0}$	$y_{7:4}$	$y_{3:0}$	$y_{7:0}$
5	5	5	–5	2	–7	25
5	10	5	5	4	–4	60
–5	5	5	5	–2	7	–25
–5	10	5	5	–4	4	–60
7	15	7	7	7	7	119
–8	15	–8	–8	–8	–8	–136

can think of the cell as generating two 4-bit outputs that satisfy the expression

$$16y_{7:4} + y_{3:0} = (a_{3:0} \times b_{3:0}) + c_{3:0} + d_{3:0}, \tag{2.4}$$

where $y_{7:4}$, and $y_{3:0}$, $a_{3:0}$, $c_{3:0}$, and $d_{3:0}$ all have two's-complement format. Table 2.1 lists several example calculations for the B cell. Recall that 4-bit two's-complement numbers range from –8 to 7, whereas 4-bit unsigned numbers range from 0 to 15.

A similar analysis can be performed for the remaining cells used in the multiplier. For example, the C cells generate an unsigned output $y_{7:4}$ and a two's-complement output $y_{3:0}$. With the data formats shown in Figure 2.7, the 32-bit multiplier can generate a two's-complement output Y without additional hardware. Table 2.2 lists the input and output formats of each type of cell (including the G type used later). A "+" sign denotes unsigned format, and a "–" sign denotes two's-complement format.

2.7 Memory Operations

When mapping DSP algorithms onto hardware, memory is needed to store intermediate results. For example, the Fast Fourier Transform (FFT) requires a working buffer approximately the size of the input data. Most adaptations of the algorithm also use a lookup table of multiplication coefficients. It follows that the reconfigurable device should implement random-access memory of some form to fulfill the requirements of DSP algorithms.

Table 2.2: Data format requirements in two's-complement MAC

Type	$a_{3:0}$	$b_{3:0}$	$c_{3:0}$	$d_{3:0}$	$y_{7:4}$	$y_{3:0}$
A	+	+	+	+	+	+
B	−	+	−	−	−	−
C	+	+	+	−	+	−
D	−	+	−	+	−	+
E	+	−	−	+	−	+
F	+	−	+	−	−	+
G	+	+	−	+	+	−
H	−	−	−	−	−	+

Table 2.3: Memory operations of cell

Operation	we	re	Data flow
No-op	0	0	$q_{7:0} \leftarrow i_{7:0}$
Read	0	1	$q_{7:0} \leftarrow Mem[a_{5:0}]$
Write	1	0	$q_{7:0} \leftarrow i_{7:0}$
			$Mem[a_{5:0}] \leftarrow i_{7:0}$
Read-Write	1	1	$q_{7:0} \leftarrow Mem[a_{5:0}]$
			$Mem[a_{5:0}] \leftarrow i_{7:0}$

Other reconfigurable devices typically embed memory blocks within the main array of cells [12]. The two-level cell array is unique in that each cell can implement a 64×8-bit memory. The inputs and outputs of the cell in such a configuration include a 6-bit address a, 8-bit input data i, and 8-bit output data q. Depending on the read enable re and write enable we, the cell can perform the operations shown in Table 2.3.

Passing the input data to the output data on a no-op enables large memory units to be constructed easily. Consider the 512×64-bit memory diagrammed in Figure 2.8. The rightmost column of "D" cells decodes the 9-bit address A, whereas the main 8×8 block of "M" cells implements the memory. The entire module operates in a pipelined fashion. As an access request travels through the pipeline, each decoder cell determines whether A falls within the address range for the corresponding row of memory cells. If so, the re or we

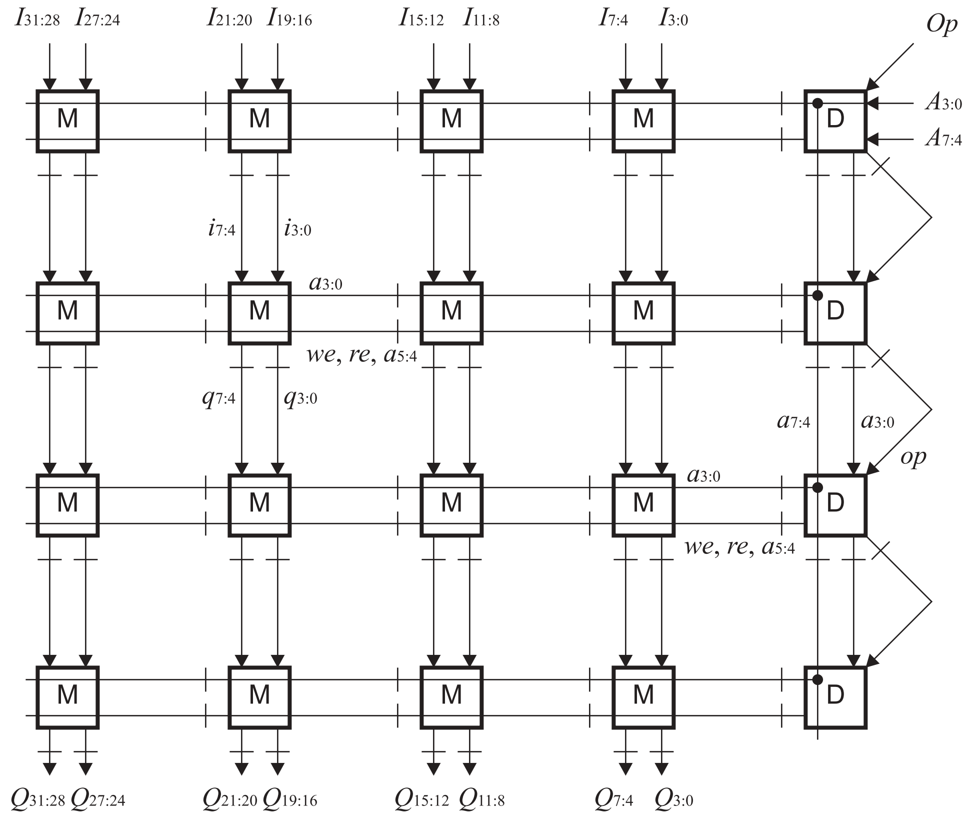


Figure 2.8: Diagram of 256×16-bit memory

signals of these cells asserted. If not, a no-op occurs and the memory cells pass the data unchanged to the next row.

2.8 Logic and Control Operations

DSP operations are not composed entirely of mathematical functions, but also require a certain amount of control logic for proper operation. This control logic may include AND-OR expressions, decoders, multiplexers, and simple state machines. For example, the FFT requires a mechanism to load data into the computational stage at the proper time.

Figure 2.9 depicts a structure that includes both combinational and sequential logic. The two "M" cells implement an 8-bit, two-way multiplexer. The "C" cell acts as a counter that records the number of cycles elapsed since the beginning of the operation. The cell uses this

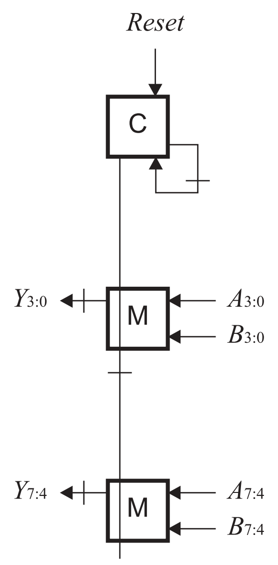


Figure 2.9: Diagram of control logic block

value to generate the control signal for the multiplexer. This module could be useful for DSP algorithms that contain several phases of execution.

Implementing control logic presents a problem for many reconfigurable devices tailored for DSP. Architectures that place a fixed number of functional units in each cell may not be able to evaluate arbitrary logic expressions efficiently. Some systems work around this problem by supplementing the reconfigurable device with a separate microprocessor: the microprocessor handles the control operations, while the reconfigurable device executes the mathematical functions [3]. In contrast, the two-level cell array has both coarse-grain and fine-grain flexibility, as demonstrated in Chapter 3.

Chapter 3

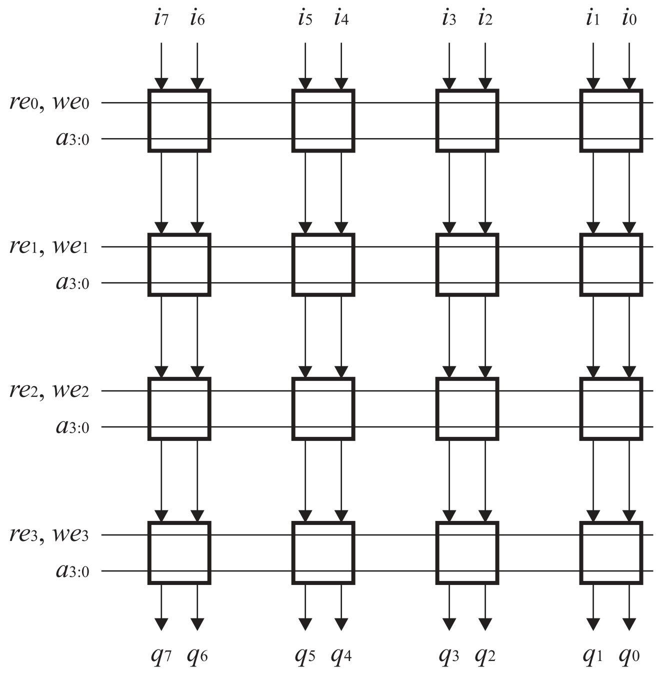
Lower-Level Organization

At the lower level, the processing core consists of a 4×4 matrix of reconfigurable elements. Each element contains a 16×2-bit memory. The processing core can be configured into two structures: one optimized for memory operations, and the other optimized for mathematical functions. Both structures execute one operation during the evaluation phase of the clock. The following sections illustrate both modes of operation and demonstrate how the matrix of elements can implement various functions.

3.1 Memory Mode

In memory mode, shown in Figure 3.1, the processing core implements a 64×8-bit memory. The lower four bits of the address a connect to every element. The control module uses the upper two bits of a to generate read and write signals for each row of elements. Lines i and q are the input data and output data, respectively. Each column of elements handles 2 bits of the data. Thus, this structure can implement the memory operations described in Chapter 2.

Possible uses of memory mode include storing intermediate results, creating a table of constant coefficients, and implementing multivariable logic functions. All of this functionality



i_7 i_6 i_5 i_4 i_3 i_2 i_1 i_0
re_0, we_0
$a_{3:0}$
re_1, we_1
$a_{3:0}$
re_2, we_2
$a_{3:0}$
re_3, we_3
$a_{3:0}$
q_7 q_6 q_5 q_4 q_3 q_2 q_1 q_0

Figure 3.1: Processing core in memory mode

is vital to implementing the control logic required in DSP algorithms.

3.2 Mathematics Mode

In mathematics mode, shown in Figure 3.2, the processing core reuses the same memory elements to implement mathematical functions. The matrix of elements now assumes a structure resembling the MAC unit presented in Chapter 2. In fact, this structure is optimized for the MAC equation in (2.1):

$$y_{7:0} = (a_{3:0} \times b_{3:0}) + c_{3:0} + d_{3:0}.$$

Although other, more sophisticated structures can perform this function, they offer little performance advantage for 4-bit word lengths. Moreover, the carry-save structure can implement many functions besides multiplication, as each element now acts as a 16×2-bit lookup table.

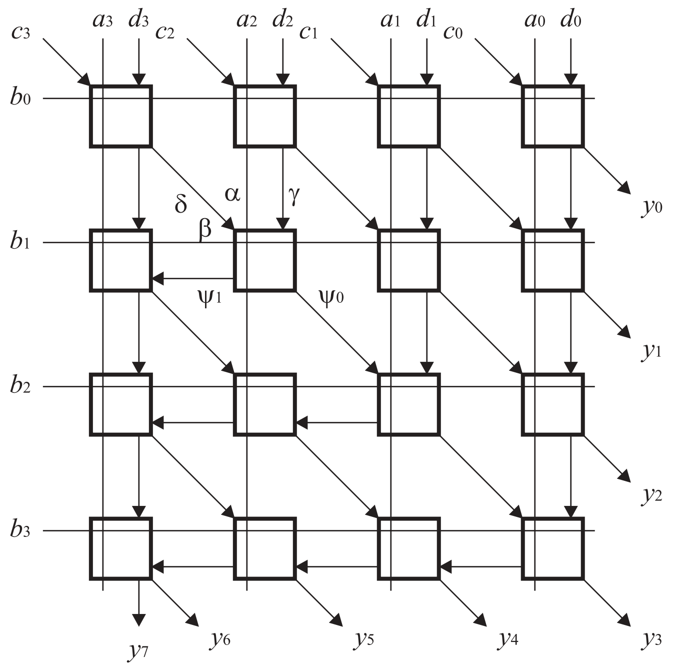


Figure 3.2: Processing core in mathematics mode

3.3 Unsigned Arithmetic

In mathematics mode, the processing core can readily perform the MAC function above. Recall from Chapter 2 that this function appears in the design of large multipliers. The memory inside each element implements a lookup table for the 1-bit MAC function.

To perform addition, the lookup tables are configured to assume the b input is unity. Now the processing core can add three four-bit numbers with the same choice of data format as before.

3.4 Two's-Complement Arithmetic

Chapter 2 demonstrated that MAC units in general require seven types of cells, denoted A, B, C, D, E, F, and H. Each cell performs the MAC function on 4-bit inputs, but different cells use different data formats. A natural question is how each cell can compute the required 4-bit operations.

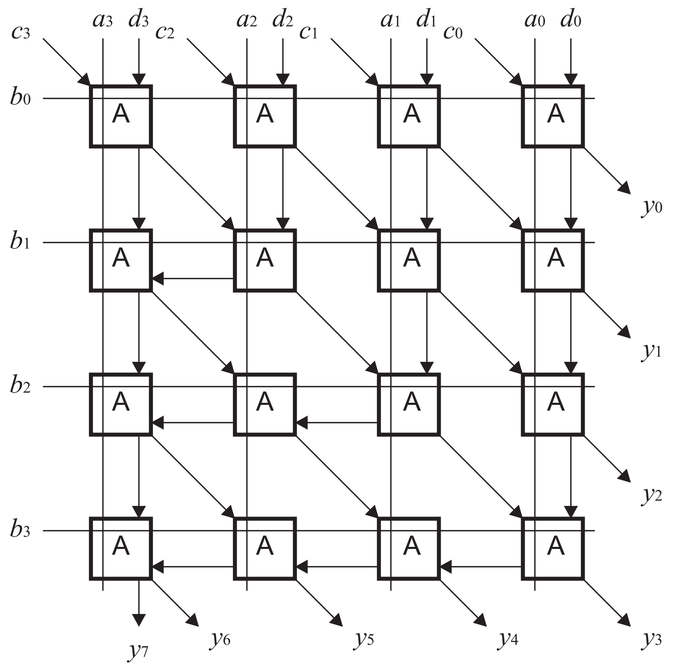


Figure 3.3: Type A cell for two's-complement MAC

3.4.1 Implementations of Two's-Complement Cells

For type A cells, the solution is simple: use mathematics mode to implement an unsigned MAC unit. As shown in Figure 3.3, each of the elements works with data in unsigned form. Hence, one can classify the elements as type A as well. Each element computes the 1-bit MAC function

$$\psi_{1:0} = (\alpha \times \beta) + \gamma + \delta, \tag{3.1}$$

where α, β, γ, and δ denote the inputs to the element, and ψ signifies the 2-bit output. Note that multiplication reduces to the logical AND operation, denoted by $\wedge$, in the 1-bit case. Each bit of ψ can be expressed in terms of the combinational logic functions

$$\psi_1 = \text{MAJ}(\alpha \wedge \beta, \gamma, \delta) \tag{3.2}$$

$$\psi_0 = \text{XOR}(\alpha \wedge \beta, \gamma, \delta), \tag{3.3}$$

where

$$\mathrm{MAJ}(P, Q, R) = (P \wedge Q) \vee (P \wedge R) \vee (Q \wedge R) \tag{3.4}$$

$$\mathrm{XOR}(P, Q, R) = P \oplus Q \oplus R. \tag{3.5}$$

For type B cells, inputs a, c, and d have two's-complement format, and b has unsigned format. Knowing the data format for each input to the cell, one can determine the format of every internal line using the information in Table 2.2. The procedure closely parallels the analysis for the two's-complement multiplier in Chapter 2, except that the signal names are Greek symbols instead of lowercase letters. As shown in Figure 3.4, the implementation of the type B cell requires elements of types A, B, and C. Note that both the upper and lower portions of the y output have two's-complement format, as required.

Continuing on, cells of types C and D have straightforward implementations, as shown in Figure 3.4. Type E cells require five types of elements, including type G. Type F cells are similar. Finally, type H cells have the same formatting assignments as the twos-complement multiplier. This property holds because all the inputs and outputs of a type H cell have two's-complement format.

3.4.2 Reduction of Element Types

Now consider the MAC function computed by type B elements. From Table 2, the α, β, δ, ψ_1, and ψ_0 signals of type B elements all have two's-complement format. For these signals, logic 0 denotes 0 and logic 1 denotes -1. Hence, type B elements compute the expression

$$-2\psi_1 - \psi_0 = (-\alpha \times \beta) - \gamma - \delta, \tag{3.6}$$

which simplifies to

$$2\psi_1 + \psi_0 = (\alpha \wedge \beta) + \gamma + \delta. \tag{3.7}$$

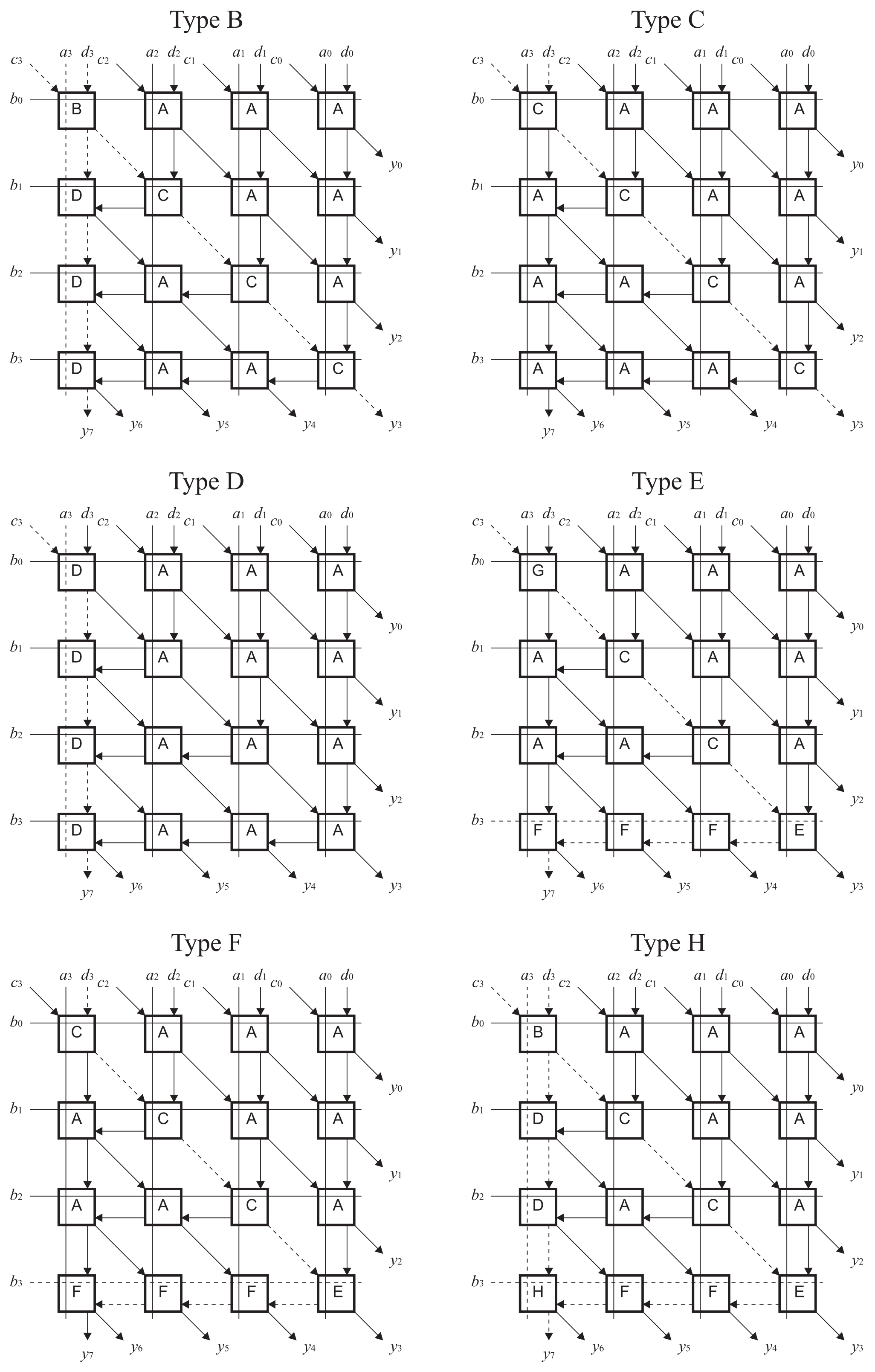


Figure 3.4: Other types of cells for two's-complement MAC

Table 3.1: Reduction of element types for two's-complement multiplication

Type	ψ_1	ψ_0	Same as
A	$\mathrm{MAJ}(\alpha \wedge \beta, \gamma, \delta)$	$\mathrm{XOR}(\alpha \wedge \beta, \gamma, \delta)$	A
B	$\mathrm{MAJ}(\alpha \wedge \beta, \gamma, \delta)$	$\mathrm{XOR}(\alpha \wedge \beta, \gamma, \delta)$	A
C	$\mathrm{MAJ}(\alpha \wedge \beta, \gamma, \neg\delta)$	$\mathrm{XOR}(\alpha \wedge \beta, \gamma, \delta)$	C
D	$\mathrm{MAJ}(\alpha \wedge \beta, \gamma, \neg\delta)$	$\mathrm{XOR}(\alpha \wedge \beta, \gamma, \delta)$	C
E	$\mathrm{MAJ}(\alpha \wedge \beta, \gamma, \neg\delta)$	$\mathrm{XOR}(\alpha \wedge \beta, \gamma, \delta)$	C
F	$\mathrm{MAJ}(\alpha \wedge \beta, \neg\gamma, \delta)$	$\mathrm{XOR}(\alpha \wedge \beta, \gamma, \delta)$	F
G	$\mathrm{MAJ}(\alpha \wedge \beta, \neg\gamma, \delta)$	$\mathrm{XOR}(\alpha \wedge \beta, \gamma, \delta)$	F
H	$\neg\mathrm{MAJ}(\alpha \wedge \beta, \neg\gamma, \neg\delta)$	$\mathrm{XOR}(\alpha \wedge \beta, \gamma, \delta)$	H

Since (3.1) and (3.7) are equivalent, elements of types A and B implement the same combinational logic expressions.

Performing a similar analysis on the remaining types of cells reveals that only four distinct types of elements are required. In fact, each element implements the same expression for ψ_0; the only difference is the expression used to compute ψ_1. Table 3.1 lists the functions corresponding to each type of element. (Here $\neg$ denotes the logical complement.) A reconfigurable architecture could exploit these similarities to implement all necessary operations efficiently.

3.5 Shifting

Another operation that frequently appears in floating-point arithmetic is bit shifting. Figure 3.5 shows how the processing core can implement a universal bit shifter in mathematics mode. Under this configuration, the y output can be assigned any 4-bit subsequence of the string $c_3c_2c_1c_0d_3d_2d_1d_0$, such as $c_0d_3d_2d_1$. The a and b inputs control the operation of the shifter. The top row of elements act as two-way multiplexers between the bits of c and d. The remaining elements route these bits to the proper output positions. Note that the light gray elements are configured to pass data on without modification. All these operations are

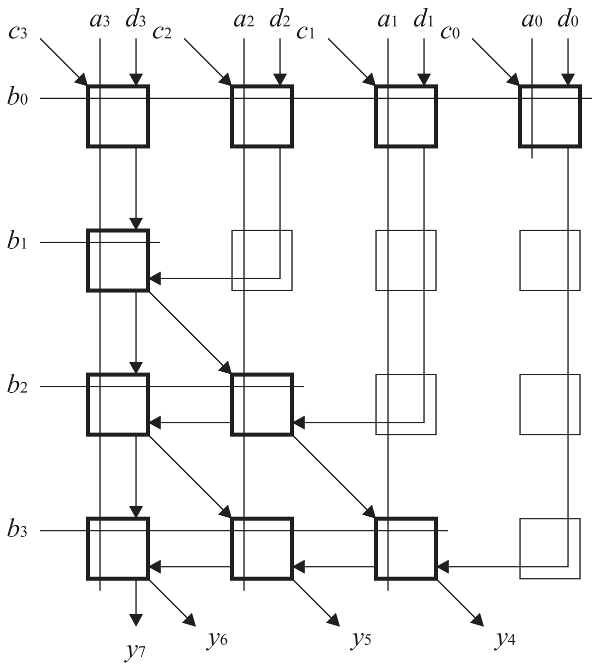


Figure 3.5: Mathematics mode used for bit shifter

possible with suitable configuration of the lookup tables.

3.6 Logic and Control Functions

The processing core can also use mathematics mode to evaluate simple logic functions, as illustrated in Figure 3.6. Observe that the 16×2-bit memory inside each element can define two functions of up to four variables. In the figure, the elements in the top row implement the desired functions. The remaining elements pass the results to the outputs. By using pipeline latches, the cell can evaluate sequential as well as combinational logic.

Another way to implement logic functions is to change the processing core to memory mode and use the 64×8-bit memory as a large lookup table.

Finally, the processing core can implement a 4-bit, 4-way multiplexer using a special feature of memory mode. As shown in Figure 3.7, the four inputs are placed onto lines a, b, c, and d and passed to each column of elements. For regular memory operations, all these

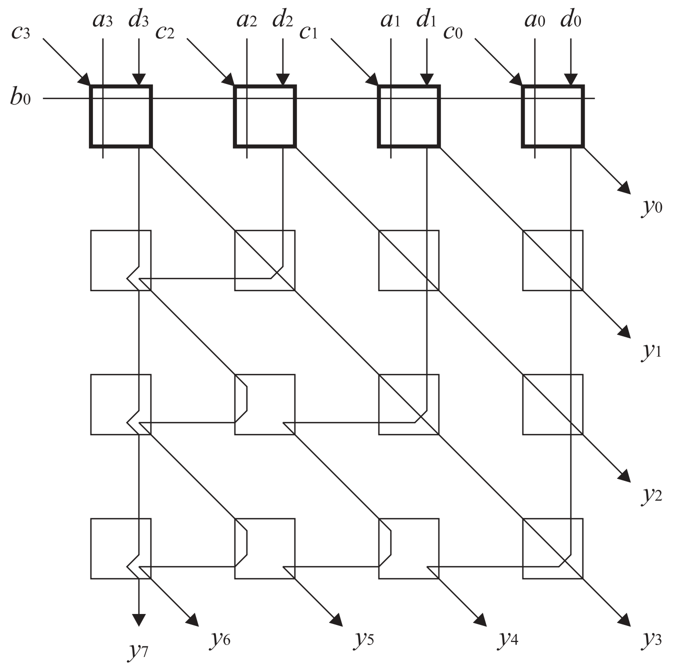


Figure 3.6: Mathematics mode used as logic unit

lines are tied to the lower 4 bits of the address. The control module uses the upper two bits of the address to enable one row of elements. The selected elements simply copy the selected input to the q output.

3.7 Configuration

Before using the reconfigurable device to perform DSP, each cell must be programmed to implement the desired operation. The process begins when the target system places the cell in programming mode. The system can then change the configuration of the processing core, as well as the switch and interface. During this time, the processing core behaves as a random-access memory so that the end system can load information into the matrix of elements using normal write operations. Further details of this process appear in [23].

In all, the design of the cell combines the flexibility of a fine-grain architecture with the performance of a coarse-grain architecture. Table 3.2 lists some of the operations possible

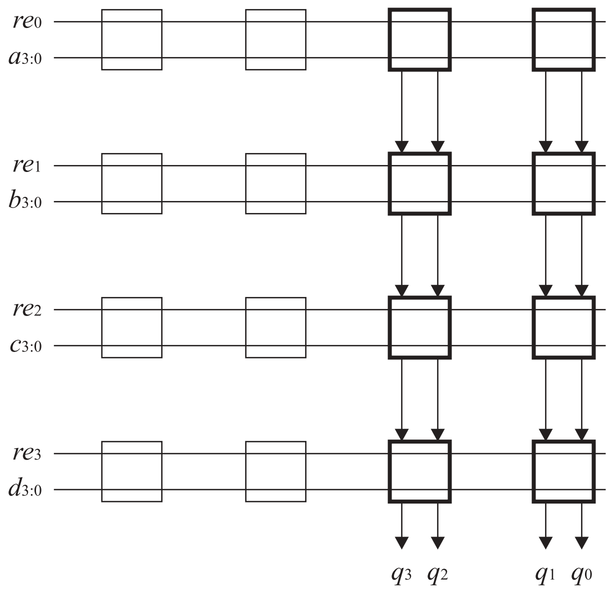


Figure 3.7: Memory mode used as 4-way multiplexer

with suitable cell configurations. By using lookup tables, the processing core can work with different data formats easily.

Table 3.2: Examples of cell operations

Operation	Remarks
$y = (a \times b) + c + d$	Unsigned or signed multiply-accumulate
$y = a + b + c$	Unsigned or signed addition/subtraction
$y = (a \text{ AND } b) \text{ OR } c$	Function specified by lookup table
$y = \text{MUX}(a, b, c, d)$	Use $a_{5:4}$ in memory mode to select input
$y = \text{SHIFT}(c, d)$	Shift $c_3c_2c_1c_0d_3d_2d_1d_0$ right or left
Memory	64×8-bit capacity
Lookup table	Read-only memory
State machine	Read-only memory with pipelined feedback

Chapter 4

Implementation

This chapter completes the presentation of the lower-level organization by describing the circuit design of an element. A notable feature of this architecture is the absence of functional units such as adders. The entire design consists of a hierarchy of memory units with some simple glue logic. This strategy leads to a simple and compact VLSI implementation that achieves high performance with moderate power consumption. For applications where reliability is also critical, error detection and correction circuitry can be added to the memory units, as described in [26]. The following discussion focuses on the circuit schematics and transistor layout of the reconfigurable element.

4.1 Circuit Design

Figure 4.1 depicts the organization of one element in the processing core. Each element contains a 16×2-bit memory. This memory is arranged into a 4×4 array of 2-bit latches, together with additional glue logic. In memory mode, the element has a 4-bit address a, 2-bit input data i, and 2-bit output data q. In mathematics mode, the four address bits are pre-empted by inputs α, β, γ, and δ. The lower two bits control a row decoder, and the upper two bits control a column decoder.

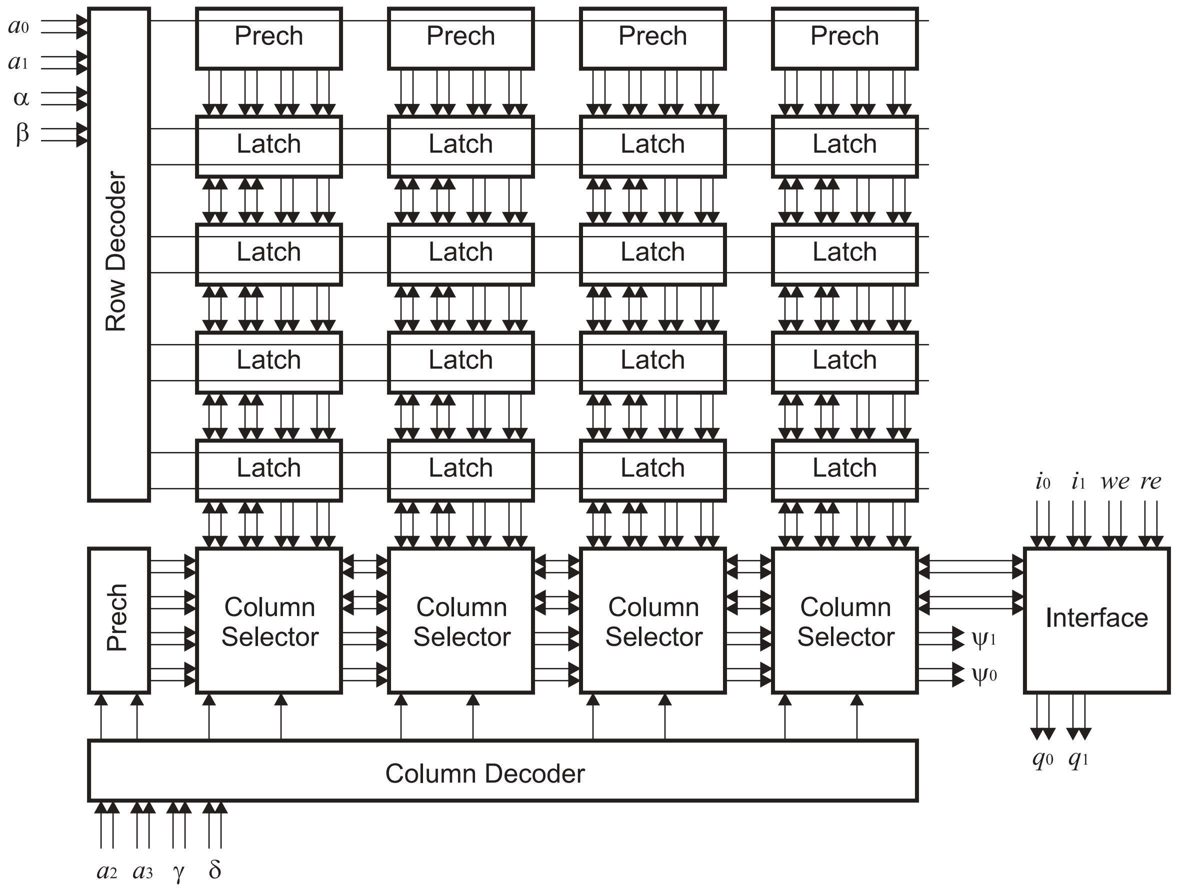


Figure 4.1: Organization of reconfigurable element

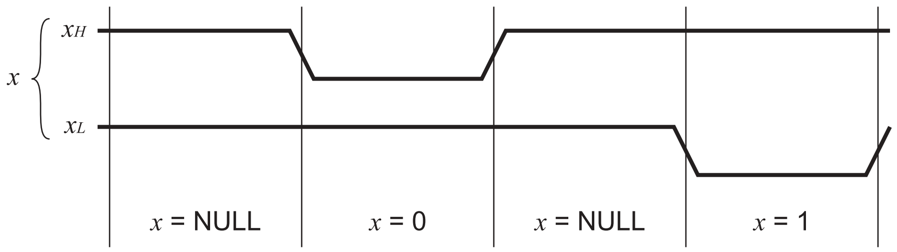


Figure 4.2: Data format used in reconfigurable element

The element uses a special data format to achieve high performance. As shown in Figure 4.2, each bit x is represented by two signals, x_H and x_L. Initially, both signals are precharged to V_{DD}, indicating a NULL condition. Discharging x_H specifies logic 0; discharging x_L specifies logic 1. Under normal operation, both signals are never low at the same time. The components in the element do not require a separate clock signal since the data itself contains all necessary timing information.

This data format is especially suited to the design of the latch, illustrated in Figure 4.3. Each 2-bit latch contains two static random-access memory (SRAM) cells. The circuit provides separate paths for memory mode and mathematics mode. For a read operation in memory mode, the element first precharges $MemLine_H$ and $MemLine_L$ to V_{DD}. Then, the row decoder asserts the $MemEn$ input, allowing the latch to discharge one of these signals to ground. The latch contains strong n-transistors to expedite this operation. A read operation in mathematics mode proceeds in a similar fashion. For a write operation in memory mode, the element drives the new data into $MemLine_H$ and $MemLine_L$. When $MemEn$ is asserted, the data overwrites the value stored in the latch.

The other components in the element are very simple. The column selector, depicted in Figure 4.4, connects one column of data lines to the main data lines of the element. The component contains n-type pass transistors, so it can pass a strong logic 0 or a weak logic 1 in either direction. The precharger units, shown in Figure 4.5, charge the internal lines to V_{DD} when the element is not performing any operation. The column decoder in Figure 4.6 enables one of the column selectors based on the upper two bits of the input address. The

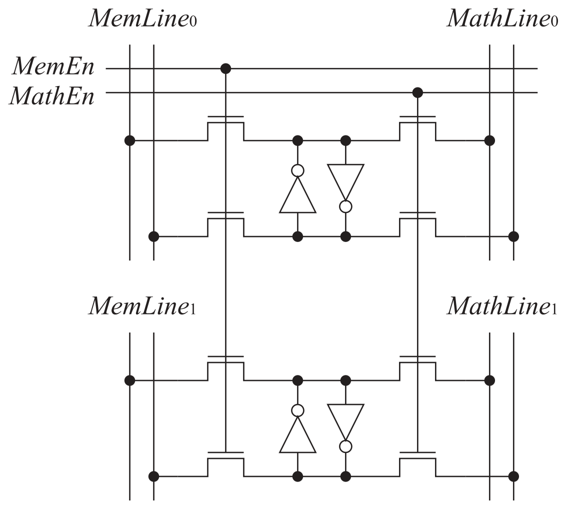


Figure 4.3: 2-bit latch with separate paths for memory mode and mathematics mode

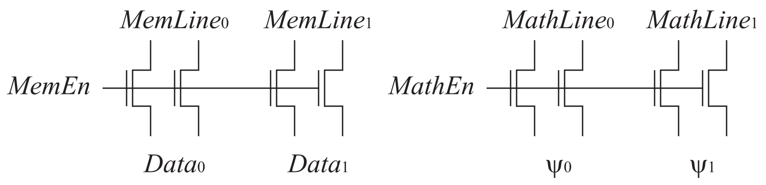


Figure 4.4: Column selector

row decoder, which has a similar structure, enables one row of latches based on the lower two address bits. Both decoders generate separate signals for memory mode and mathematics mode. The decoders also turn on the appropriate precharger units when the input address is NULL.

The final component of the element, the interface module, controls the read and write operations in memory mode. As shown in Figure 4.7, the module contains a three-way switch between the main data line *Data*, input data i, and output data q. A data-driven

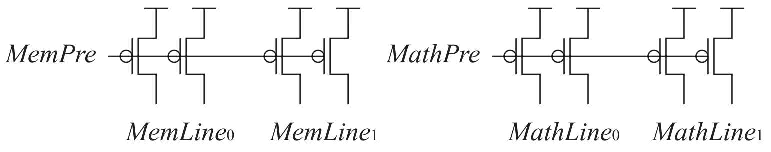


Figure 4.5: Precharger unit

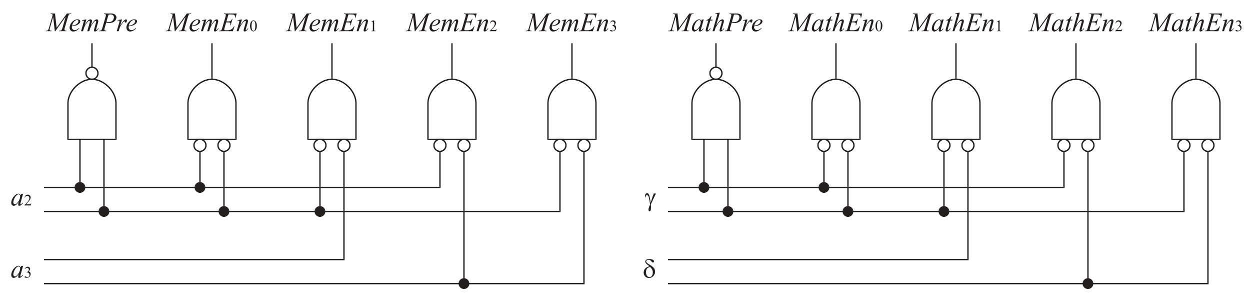


Figure 4.6: Column decoder

state machine enables the n-type pass transistors at the appropriate times. Intially, all q lines are precharged to V_{DD}. The action of the interface module during the evaluation phase depends on the type of memory operation being performed on the element:

- **No-op:** If no memory operation is being performed on the element, i is connected to q so that data can flow from one element to the next. This step is necessary because other elements may need the data for reading or writing.
- **Read:** For a read operation, *Data* is connected to q through the top two transistors.
- **Write:** A write operation consists of two phases. First, i is connected to q and allowed to discharge one of the output lines. The external circuitry senses that the output data has evaluated and asserts the *Ready* signal. Then, i is connected to *Data* instead and drives the data into the memory.
- **Read-Write:** A read-write combines the two above operations. First, *Data* is connected to q and allowed to discharge the output. When the external circuitry asserts *Ready*, *Data* is connected to i instead.

4.2 Operation

The matrix of elements operates on a single clock signal, *Clock*. While *Clock* is high, the elements precharge their internal nodes to V_{DD}. When *Clock* falls low, the matrix of elements

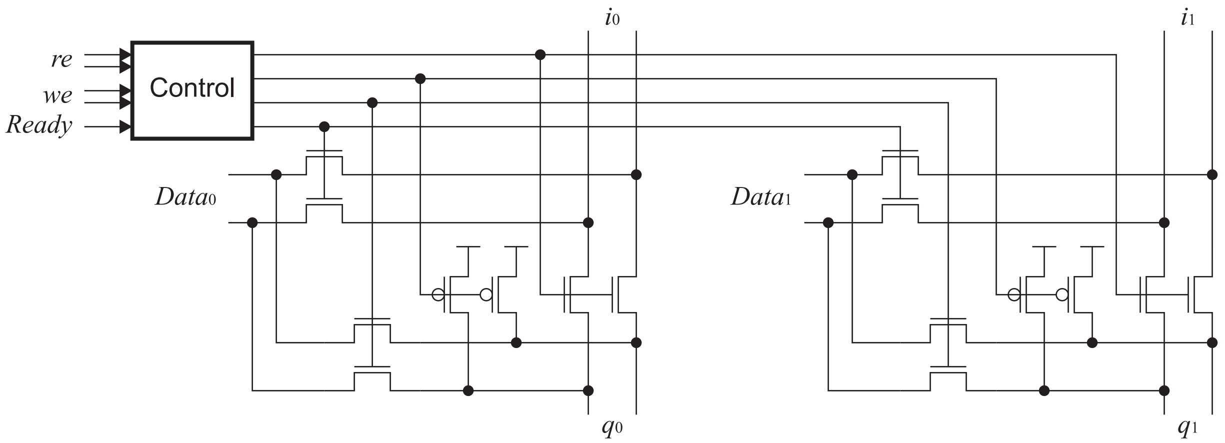


Figure 4.7: Interface module

performs the required operation. In mathematics mode, the output of one element propagates to the inputs of neighboring elements, placing an upper limit on the clock frequency. The chain of seven elements illustrated in Figure 4.8 comprises the critical path.

To reduce the total propagation delay, elements use a circuit style similar to DOMINO logic, shown in Figure 4.9. Each element implements a pull-down network followed by a CMOS decoder. The outputs of the latches travel through the n-network of the column selector and reach the *Data* outputs. These outputs connect to the column decoder in the next element. The decoder uses CMOS logic to drive the gates of the column selector. This series of pull-down networks and CMOS decoders repeats for all elements in the chain.

For a mathematics operation, all the address inputs are initially charged to V_{DD}, causing the decoders to turn on the precharge transistors and disable the latches. Then, inputs α and β are broadcast to all elements simultaneously. When these inputs evaluate, the row decoder turns off the column precharge transistors and enables the appropriate row of latches. The data in the latches begins to propagate to the ψ outputs. When the previous elements cause γ and δ to evaluate, the column decoder turns off the precharge transistors on the output and enables one column. The ψ outputs then evaluate, affecting the γ and δ inputs of the next elements. This process creates a domino effect that allows the matrix of elements to perform mathematical operations rapidly.

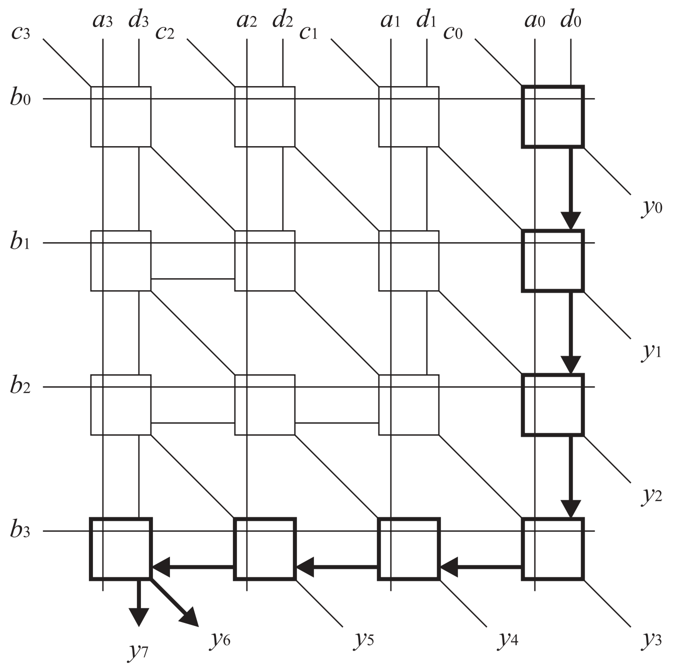


Figure 4.8: Critical path in mathematics mode

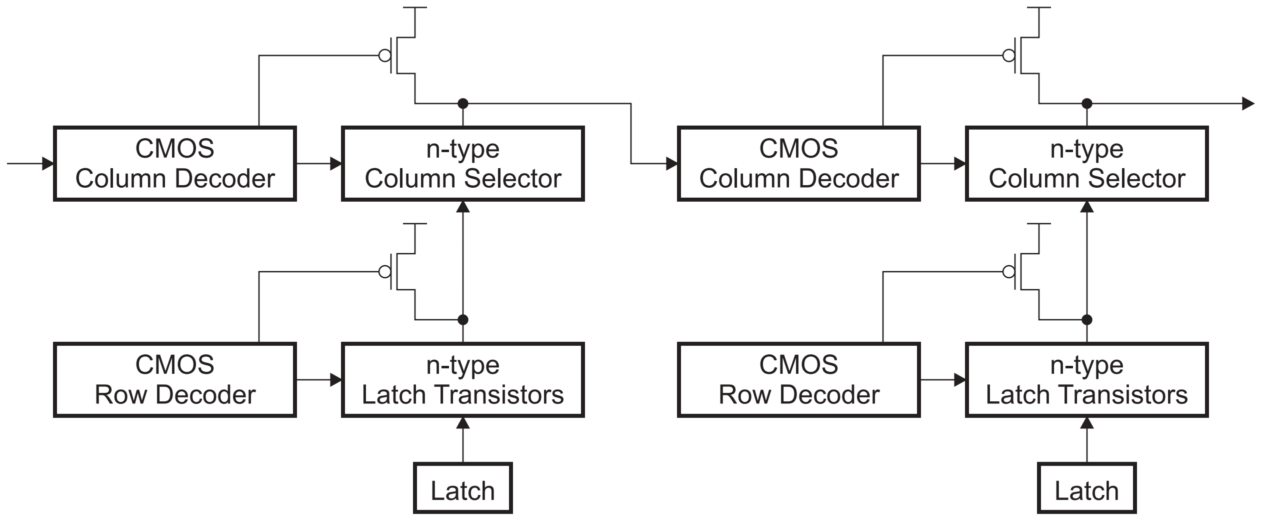


Figure 4.9: DOMINO logic blocks

A read operation in memory mode operates in a similar fashion, except that each element receives all four bits of the input address a at the same time. The latch at the selected row and column discharges $Data_H$ or $Data_L$ to ground, depending on the stored value. The cell uses these lines to set the read data q.

To perform a write operation, which can only occur in memory mode, the element first executes a read operation at the input address, storing the resulting value if necessary. After $Data$ evaluates, the element drives value of the i input onto the same lines. The n-type pass transistors in the datapath now run in reverse, storing the new data in the selected latch.

4.3 Transistor Layout

Figure 4.10 depicts a sample layout of the reconfigurable element in a 0.5-μm CMOS technology. This layout was used in the prototype of the reconfigurable cell described in Chapter 5. The 4×4 array of latches resides in the center, with horizontal enable lines and vertical data lines. Although the circuit design used in the prototype was slightly different than the design just presented, the element has a compact layout that fits inside a rectangular area.

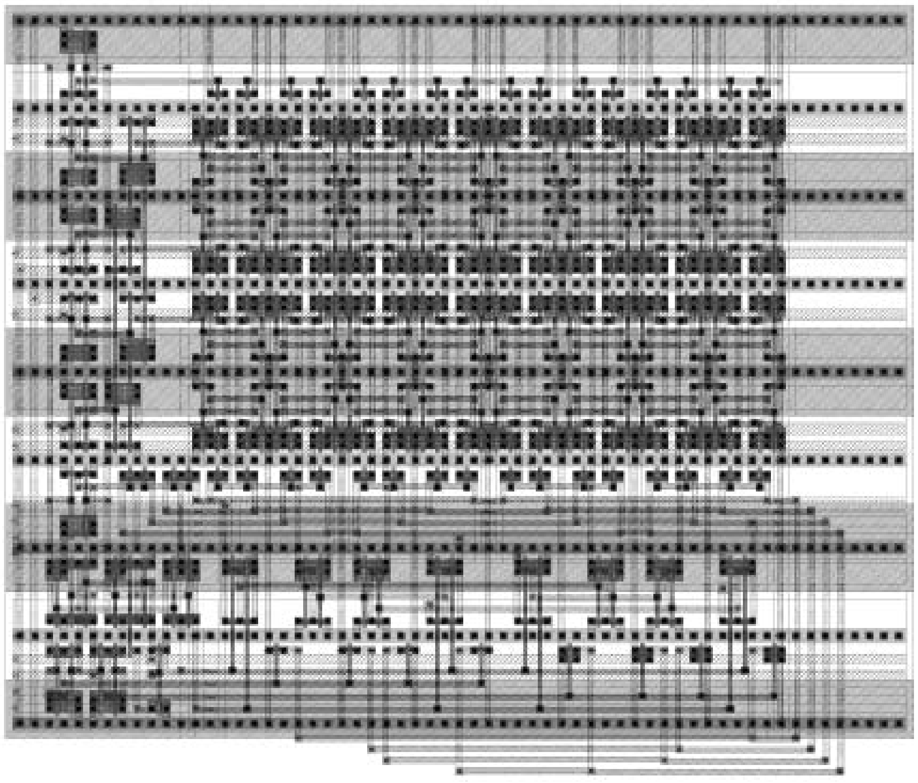


Figure 4.10: Layout of reconfigurable element in prototype

Chapter 5

Simulations and Prototype

The operation of a cell that determines the maximum clock frequency is a read operation in mathematics mode. As shown in Chapter 4, the critical path in the processing core involves one element in memory mode, but seven elements in mathematics mode. However, the circuitry that performs this critical operation has been optimized for speed. The transistor-level simulations presented in this chapter demonstrate that the reconfigurable cell can operate with a clock period of 5 ns using 0.25-μm CMOS technology.

5.1 Memory Mode

The first simulation demonstrates that the reconfigurable cell can read and write data in memory mode. As listed in Table 5.1, the system first writes 00 into the 2-bit latch at address 0. In the next clock cycle, the system performs a read-write operation at that address and changes its value to 11. The output of the simulation appears in Figure 5.1.

As shown in the simulation, bit 0 of the latch is initially at logic 1, but transitions to logic 0 when the first write operation occurs. The read-write operation contains two distinct steps. First, the output data line q_{0H} falls to logic 0, indicating that a logic 0 has been read from the element. When the read completes, the system drives the new data into the latch.

Table 5.1: Operations performed in memory mode simulation

Time	Operation
Initial condition	Address 0 stores 11
1.0 ns – 3.5 ns	Precharge
3.5 ns – 6.0 ns	Write 00 to address 0
6.0 ns – 8.5 ns	Precharge
8.5 ns – 11.0 ns	Read contents of address 0 and replace with 11

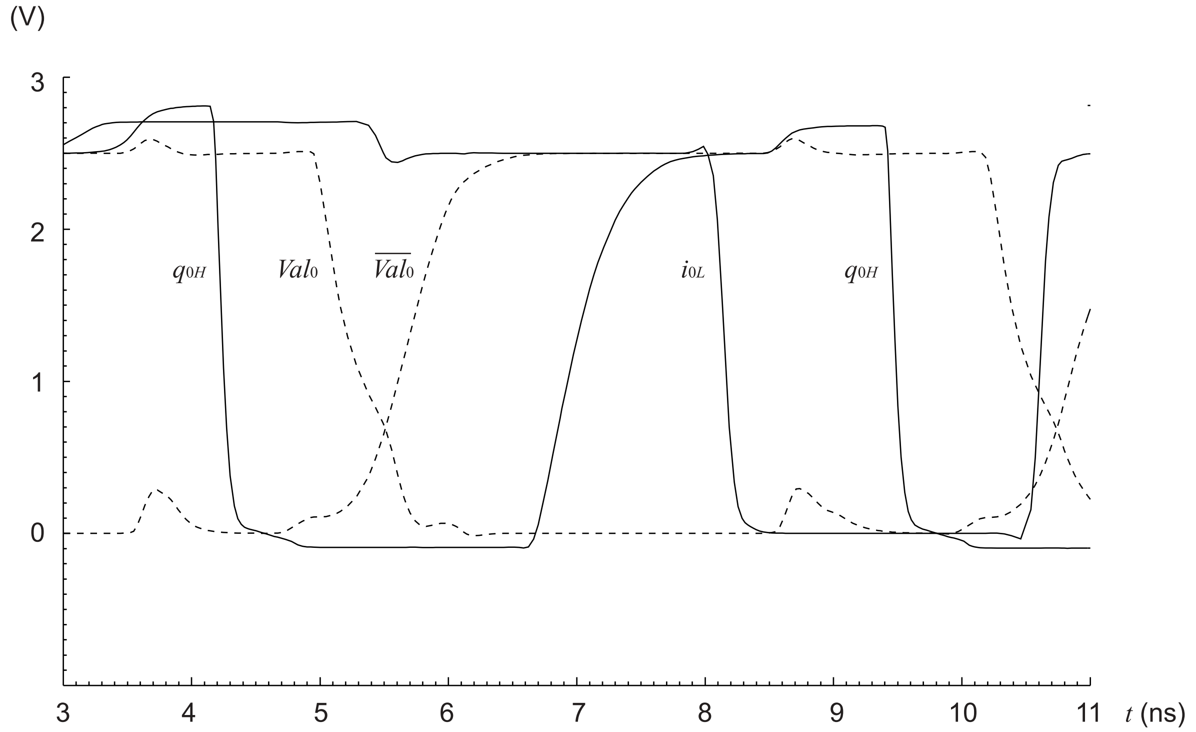


Figure 5.1: Simulation of processing core in memory mode

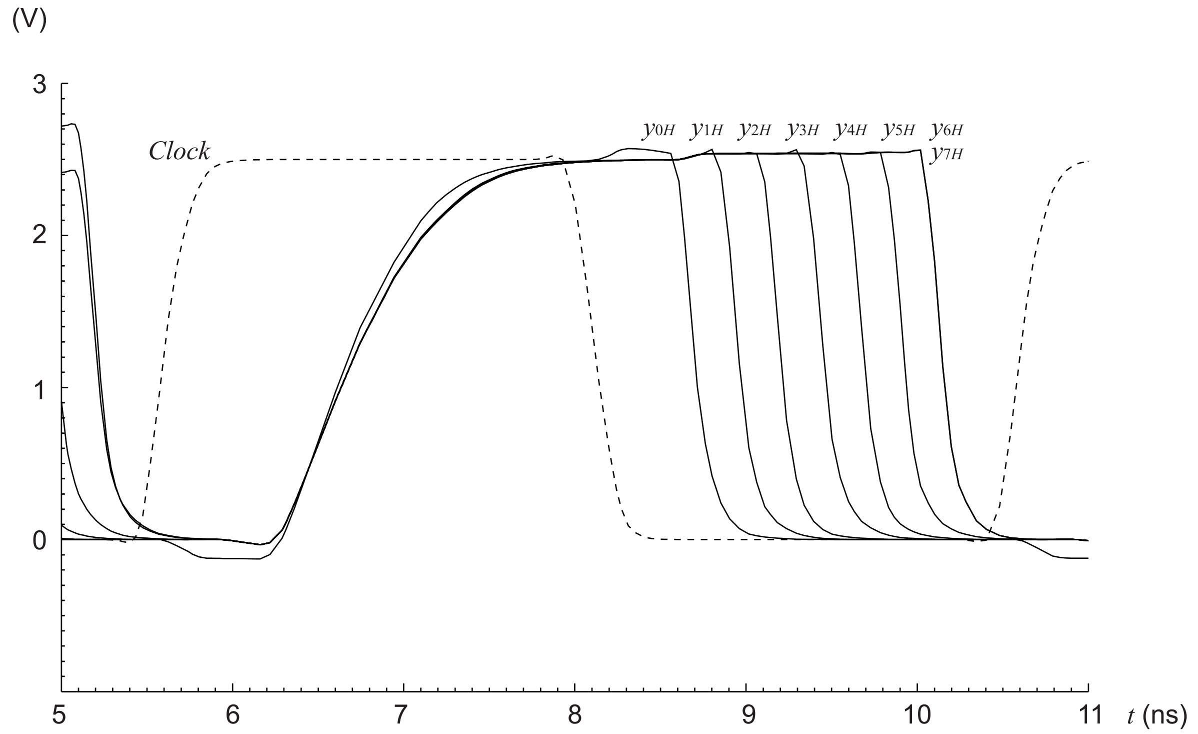


Figure 5.2: Simulation of processing core in mathematics mode

The latch changes state just before the 2.5-ns limit of the clock phase.

5.2 Mathematics Mode

Figure 5.2 contains a simulation of the processing core during a worst-case mathematics operation. When *Clock* is high, the processing core precharges all internal data lines and allows the switches to route new data to the inputs. When *Clock* falls low, the evaluation phase begins. The calculated result is zero for this example, so y_{0H} through y_{7H} all fall to ground. Bit 0 evaluates first, followed by bits 1, 2, 3, and so on.

The behavior of the processing core agrees with the analysis of the critical path in Chapter 4. Outputs y_{6H} and y_{7H} evaluate together because the last element in the chain generates both simultaneously. The total propagation delay through the processing core is just under 2.5 ns.

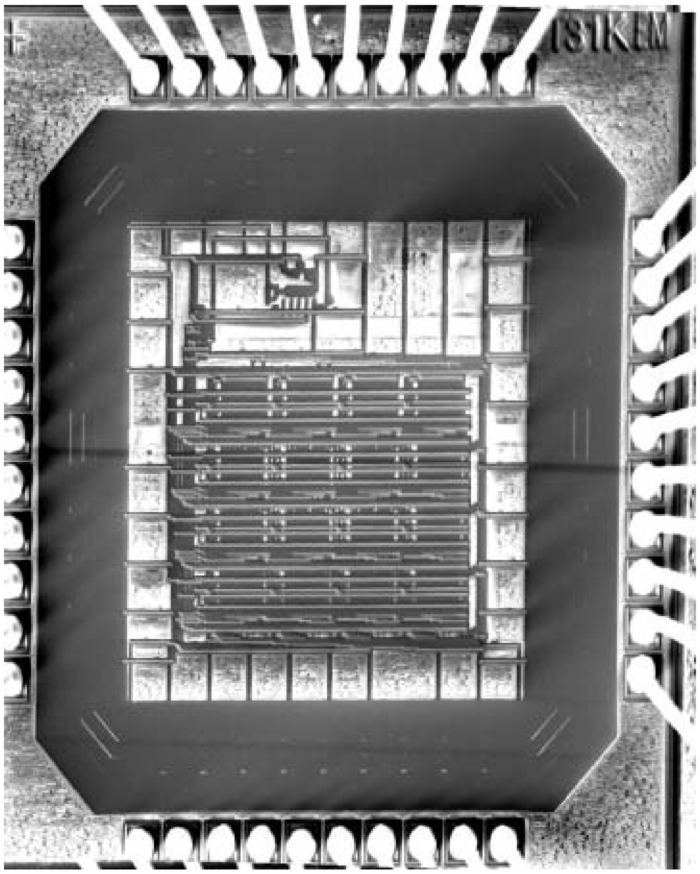


Figure 5.3: Photomicrograph of prototype chip

5.3 Functional Verification

An initial prototype of the processing core has been fabricated by the MOSIS Prototyping Service in 0.5-μm technology. Figure 5.3 depicts a photomicrograph of the chip. The large block in the center contains the 4×4 matrix of elements, whereas the other small blocks implement control circuitry. Due to the simplicity of the VLSI implementation, the layout of the processing core is very compact and scalable.

Figure 5.4 contains a series of waveforms that demonstrate the functionality of the prototype. In this test, the circuit is first configured to implement a 4-bit multiplier. Recall that the processing core is placed into memory mode to load values into the lookup tables. The circuit then calculates all perfect squares from 1×1 to 15×15, as listed in Table 5.2. The

Table 5.2: Test cases to verify prototype

Cycle	Operation	Output
1–65	Configuration	
66	$0 \times 0 = 0$	0000 0000
67	$1 \times 1 = 1$	0000 0001
68	$2 \times 2 = 4$	0000 0100
69	$3 \times 3 = 9$	0000 1001
70	$4 \times 4 = 16$	0001 0000
71	$5 \times 5 = 25$	0001 1001
72	$6 \times 6 = 36$	0010 0100
73	$7 \times 7 = 49$	0011 0001
74	$8 \times 8 = 64$	0100 0000
75	$9 \times 9 = 81$	0101 0001
76	$10 \times 10 = 100$	0110 0100
77	$11 \times 11 = 121$	0111 1001
78	$12 \times 12 = 144$	1001 0000
79	$13 \times 13 = 169$	1010 1001
80	$14 \times 14 = 196$	1100 0100
81	$15 \times 15 = 225$	1110 0001

device functions correctly for all inputs, showing that both memory mode and mathematics mode work correctly.

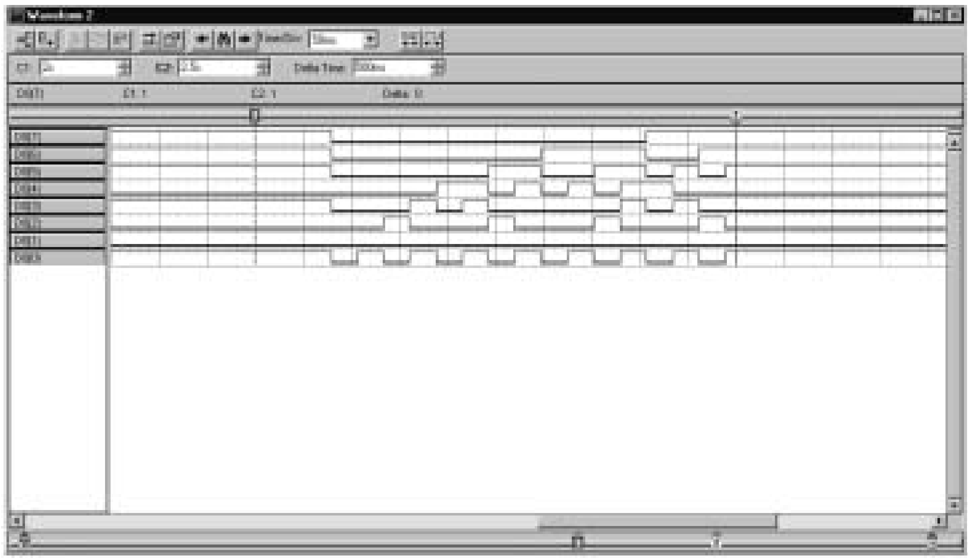


Figure 5.4: Verification of prototype chip

Chapter 6

Interconnection Structure

A natural way to implement DSP on reconfigurable hardware is to partition the target algorithm into discrete functional units, such as multipliers, adders, memories, and control logic. Each unit can then be mapped onto a block of cells, as described in Chapter 2. However, the data transfer required within a block differs from that required between functional units. For instance, adjacent functional units typically exchange data in units of words, whereas cells inside a unit handle data in smaller portions.

The reconfigurable cell array uses a novel interconnection structure that expedites data transfer both within and between functional units. As depicted in Figure 6.1, a mesh of 4-bit busses connects neighboring cells horizontally and vertically. Superimposed onto the mesh is a structure known as the “H-tree”. Each level of this global binary tree contains a fixed number of busses; however, the number of bits per bus increases at higher levels. In this way, the H-tree resembles a fat-tree, which has been recognized as an efficient routing structure for parallel processing applications [24]. However, the bandwidth of the H-tree does taper off after a certain level.

The complete interconnection structure has a compact layout in which cells are surrounded by switches in almost all directions. Although not covered here, one could use techniques similar to those presented in [25] to fold the H-tree into an even more regular

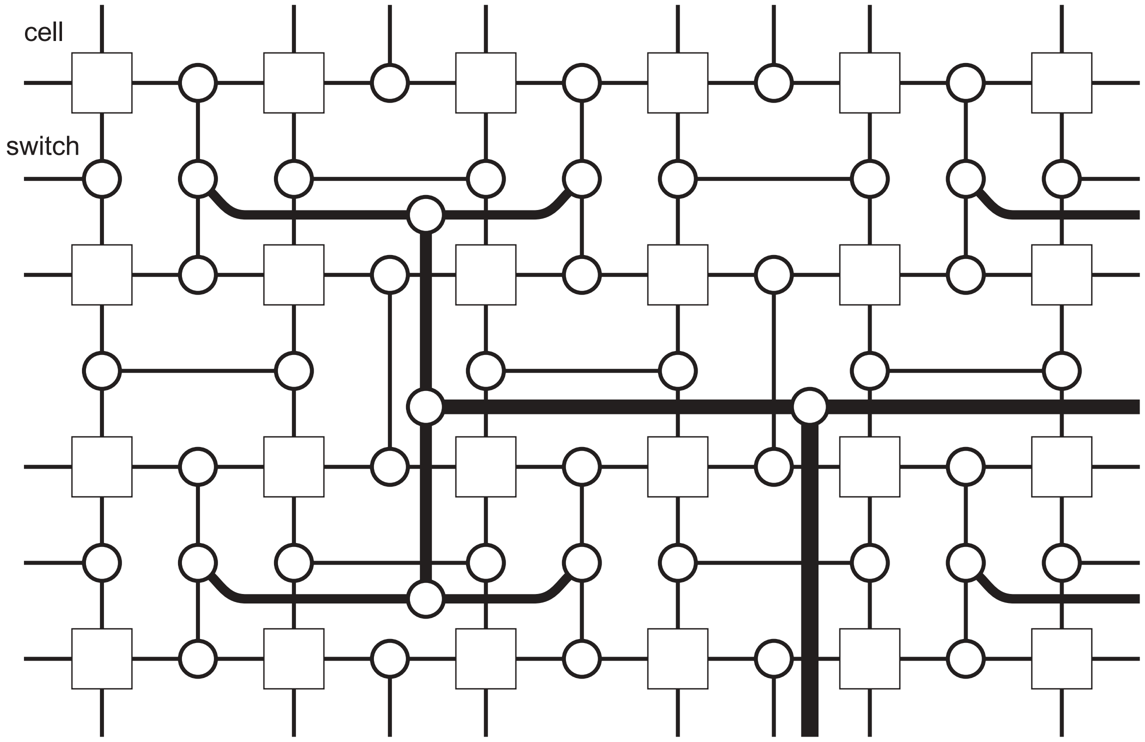


Figure 6.1: Interconnection structure in reconfigurable architecture

layout. The remainder of this chapter describes the local and global interconnection schemes, and then illustrates how several functional units can be mapped onto the structure.

6.1 Local Mesh

The local interconnect, shown in Figure 6.2, allows cells to transfer intermediate results within a functional unit. A mesh of 4-bit busses connects cells horizontally and vertically; additional "center beams" permit data to be routed in other directions. All busses are unidirectional. The regularity of the structure supports functional units of any size and shape.

Figure 6.3 illustrates one of the switches in the local mesh. As shown, these switches manipulate each 4-bit bus separately. Incoming data from a cell can either be routed to the cell opposite the switch, or through the center beam to the two more distant cells.

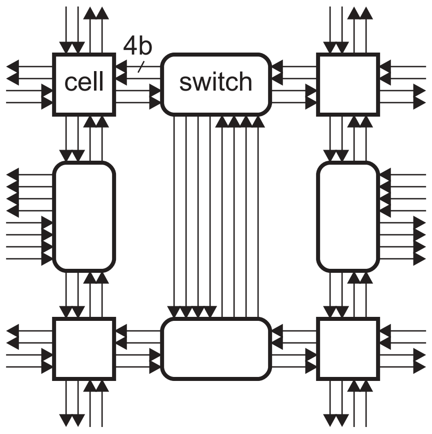


Figure 6.2: Local mesh of 4-bit busses with additional “center beams”

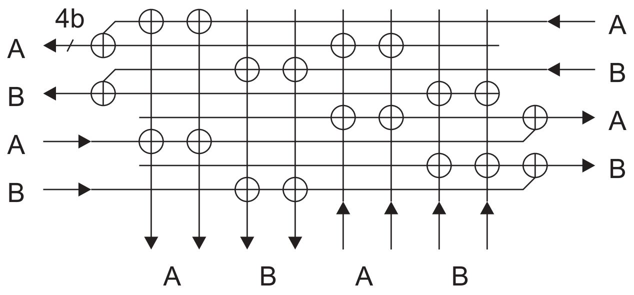


Figure 6.3: Switch in local mesh

All data transfers occur in a single clock phase while the processing core is precharging. These switches divide all data busses into two groups, designated A and B in the figure. As explained in [19], partitioning the busses in this manner does not sacrifice flexibility.

6.2 Global H-tree

The global H-tree, depicted in Figure 6.4, routes the inputs and outputs of functional units across the reconfigurable cell array. The two lowest levels of the tree are shared with the local interconnect, so cells that only interact with other cells in the functional unit do not waste the capacity of the global interconnect. However, each cell can access the H-tree in one direction. The root of the tree could connect to an internal memory or to the external pins of the device.

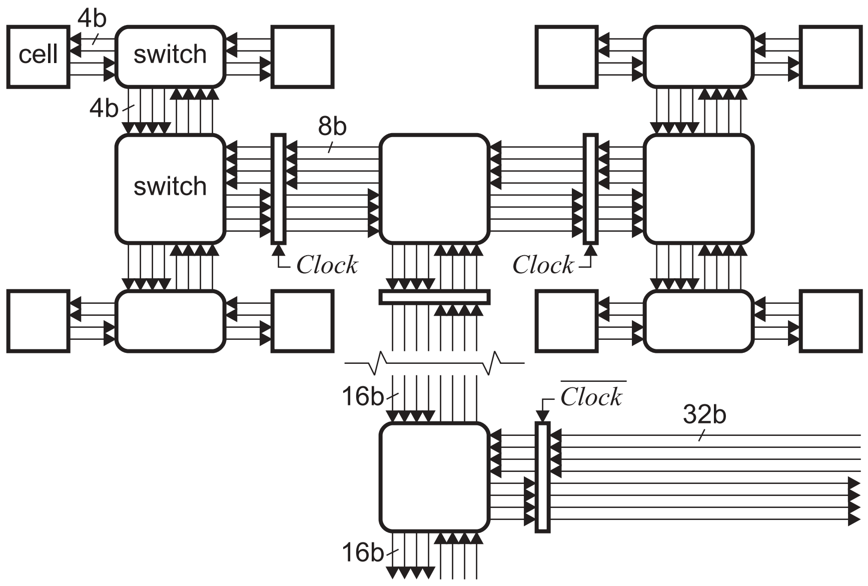


Figure 6.4: Global H-tree

Each level of the H-tree contains four input busses and four output busses. Data originating from a cell travels up the output path until it reaches the highest level required. The data then cuts through to the input path and descends to its destination. Both directions of travel on the global interconnect require routing through multiple levels of switches. Hence, the H-tree includes pipeline latches to enable higher clock frequencies. To allow for simultaneous data transfer to and from each cell, the number of lines in each bus doubles at each level, up to a maximum of 64 bits.

Figure 6.5 details a typical switch in the H-tree. Like the switches in the local mesh, busses are divided into two groups. However, the switches route data in units of 8, 16, 32, or 64 bits. The architecture of each switch is similar; only the number of bits per bus changes on each level. On the input path, the $2n$-bit busses from the upper level can be routed onto the n-bit busses of the same group in the two lower levels. The least significant and most significant n bits of the input are handled separately. On the output path, each n-bit bus from the lower level can be copied onto an outgoing $2n$-bit bus of the same group. Alternatively, the switch can transfer data from the output path to the input path on the same level; the

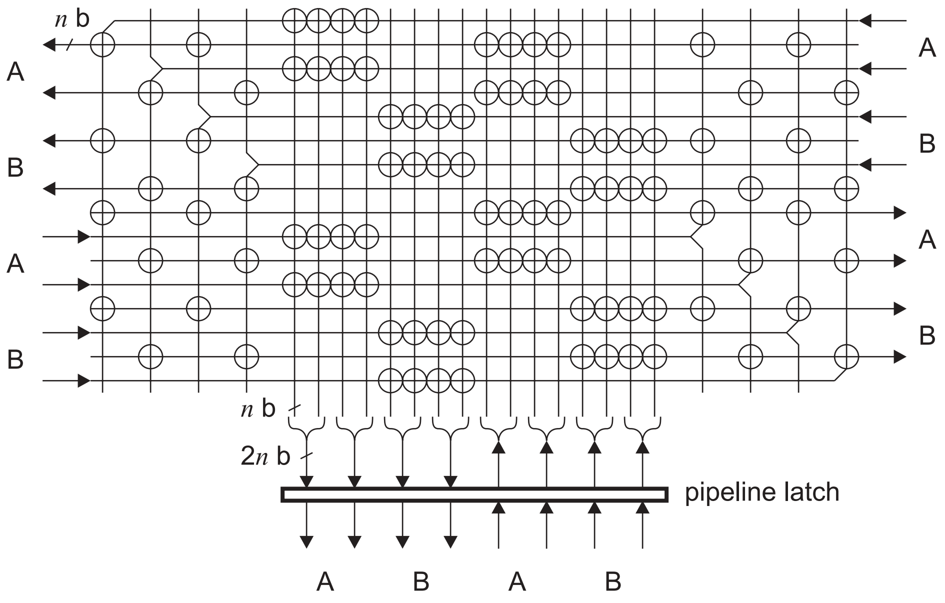


Figure 6.5: Typical switch in H-tree

group designations are not observed in this case. This approach allows designers to create libraries of functional units that can be connected easily without conflicts.

6.3 Merging Operation

The switches in the upper levels of the H-tree contain an additional provision when two n-bit busses on the output path are routed to the same destination. As shown in Figure 6.6, each 4-bit portion of the busses can be manipulated separately to avoid collisions. This merging operation is useful when collecting the outputs of a functional unit onto a single bus. Note that the configuration complexity of the switch increases somewhat, but the number of connections to each data bus remains the same. Hence, the merging operation does not affect the capacitance on the data lines.

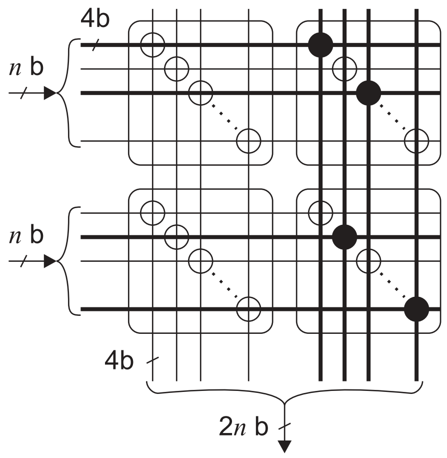


Figure 6.6: Switches can merge n-bit busses onto the same lines

6.4 Multiplication and Multiply-Accumulate

Recall from Chapter 2 that a 16-bit multiplier or multiply-accumulate unit required a 4×4 block of cells. Figure 6.7 illustrates how this functional unit can be mapped onto the interconnection structure. The thick lines denote the inputs and outputs of the unit, whereas the thin lines depict the local connections between cells. As shown, the H-tree passes A to the bottom row of cells, and passes B to the rightmost column. The seven cells along the top and right edges generate the product Y.

Due to the nature of the H-tree, all the 4-bit portions of the inputs and outputs incur the same latency as they are transferred from one functional unit to another. In other words, the 4-bit portions of Y will arrive at another functional unit in the same order as they were generated. This property gives the H-tree an advantage over other structures, in which the latency may be different for each portion of data.

6.5 Addition

Figure 6.8 illustrates the result of mapping the 16-bit adder. The linear chain of elements is folded into a square structure for greater compatibility with the H-tree. As shown, the

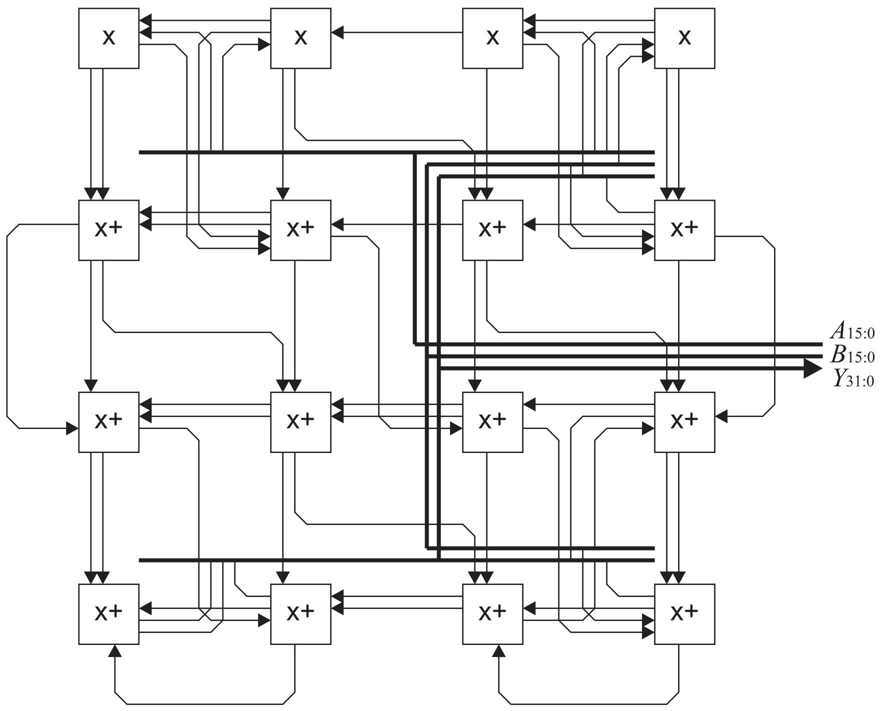


Figure 6.7: Mapped 16-bit multiplier

majority of data transfer is dedicated to the inputs and outputs of the adder. However, the interconnection structure still has enough flexibility to route the carry-out signals.

Observe how the top and bottom busses detour around the edges of the module. These busses would not conflict with any neighboring functional units, since local interconnections are only used for communication within blocks of cells.

6.6 Memory Operations

Mapping the memory module to the reconfigurable cell array produces the structure in Figure 6.9. As shown, the address A and main write enable signal W should be routed to the topmost cell in this column. The local interconnect transfers the generated control signals are transferred horizontally, along with the least significant 6 bits of A. For a write operation, the H-tree transfers the write data I to the top row of cells; the data then propagates downward along the local interconnect to the selected row. For a read operation,

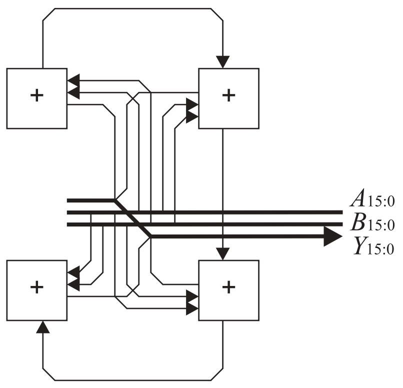


Figure 6.8: Mapped 16-bit adder

the selected row places the read data onto the local interconnect, which carries the data to the Q output on the H-tree. The interconnection structures have sufficient capacity to transfer all required data for this large memory.

Aside from some minor control logic, the functional units described here are sufficient to implement common DSP algorithms. As shown in Chapter 7, these functional units can be connected together easily using the hierarchical nature of the H-tree.

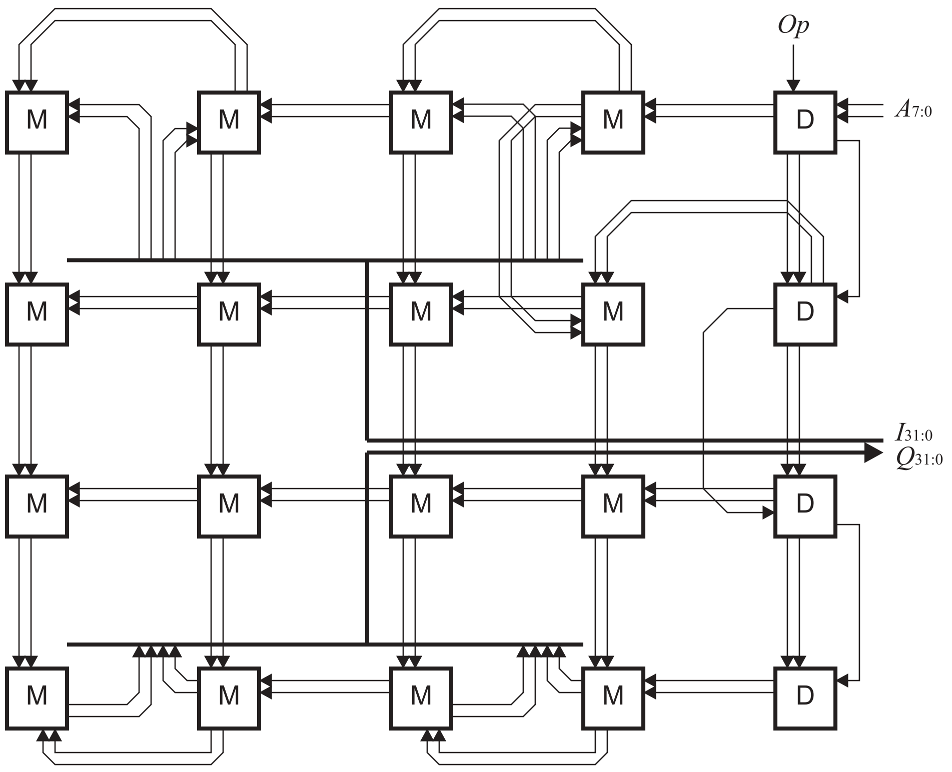


Figure 6.9: Mapped 256×16-bit memory

Chapter 7

Performance Comparison

This chapter analyzes the performance and flexibility of the reconfigurable cell array for several DSP benchmarks. In particular, a 32-bit, 512-point Fast Fourier Transform (FFT) is mapped onto the architecture using many of the functional units described previously. The last section compares the execution time of a smaller, 256-point FFT with the reported results of contemporary digital signal processors.

7.1 Mapping the Fast Fourier Transform

One of the most popular benchmarks in DSP is the Fast Fourier Transform (FFT). This algorithm is widely used to convert a discrete-time signal to the frequency domain and vice versa. The kernel of the classic decimation-in-frequency FFT appears in Figure 7.1. This "butterfly" operation includes an adder, a subtracter, and a multiplier, all of which operate on complex numbers. Initially, the input data is loaded into the memory on the left. Each pair of points is then processed by the butterfly stage and stored in the memory on the right. Then, the two memories are reversed and the process repeats. In all, a $2n$-point data set requires n processing stages.

Figure 7.2 illustrates a 512-point FFT mapped onto the reconfigurable cell array. The

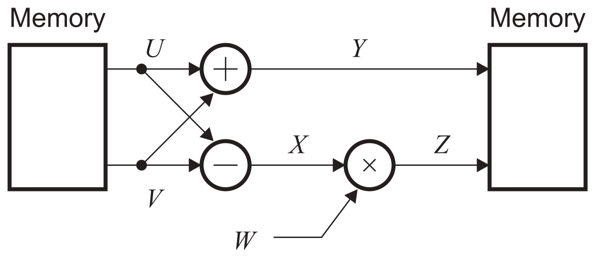


Figure 7.1: Kernel of decimation-in-frequency FFT

structure operates on fixed-point data with a 32-bit real portion and 32-bit complex portion. The "butterfly" uses many of the basic functional units described in the previous section, and occupies a 32×16 block of cells. Figure 7.3 outlines the arrangement of the functional units on this structure.

On the left side of the array are four 32-bit multipliers. The multipliers are connected together to form a 32-bit complex multiplier. Due to the properties of the FFT, the most significant bits of the output can be discarded without loss of accuracy. Hence, one corner of each multiplier is truncated and two adders are placed in the available space. The outputs of the multiplier connect to the inputs of the adders in accordance with the basic equations for complex multiplication:

$$\mathrm{Re}\{Z\} = (\mathrm{Re}\{X\} \times \mathrm{Re}\{W\})(\mathrm{Im}\{X\} \times \mathrm{Im}\{W\}) \tag{7.1}$$

$$\mathrm{Im}\{Z\} = (\mathrm{Re}\{X\} \times \mathrm{Im}\{W\}) + (\mathrm{Im}\{X\} \times \mathrm{Re}\{W\}). \tag{7.2}$$

On the right side of the array are the complex adder and subtracter used to generate X and Y. Two real 32-bit adders comprise each of these units, since the real terms and imaginary terms must be added separately. The two memories used for the FFT appear in the center. Each memory is a slightly modified version of the memory unit presented earlier, in that two words can be read or written simultaneously. Interchanging the roles of the two memories can be performed by simply rerouting the main input and output lines on the H-tree after each processing stage. Finally, the small group of 8 cells in the bottom right

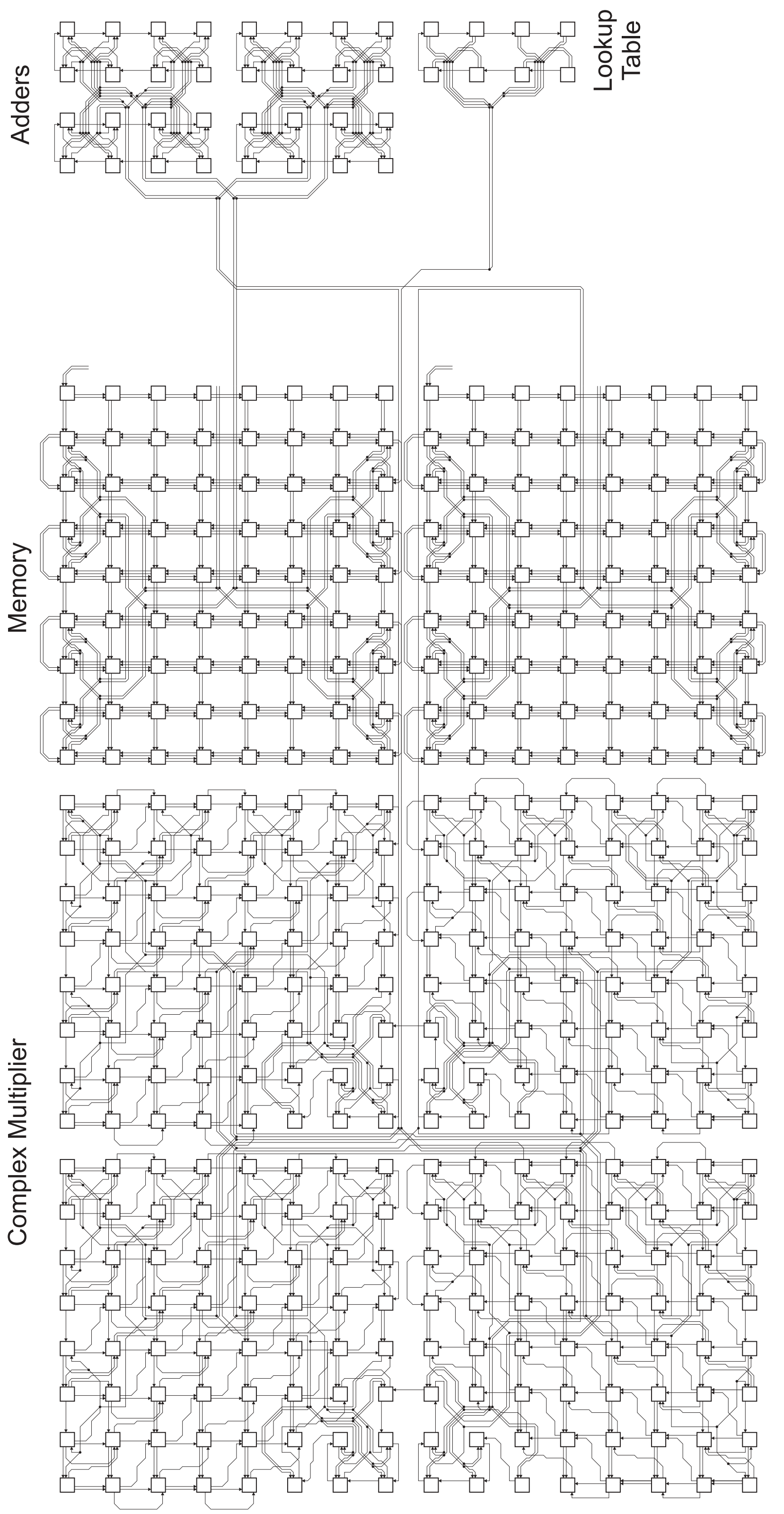


Figure 7.2: Complete mapping of 512-point FFT

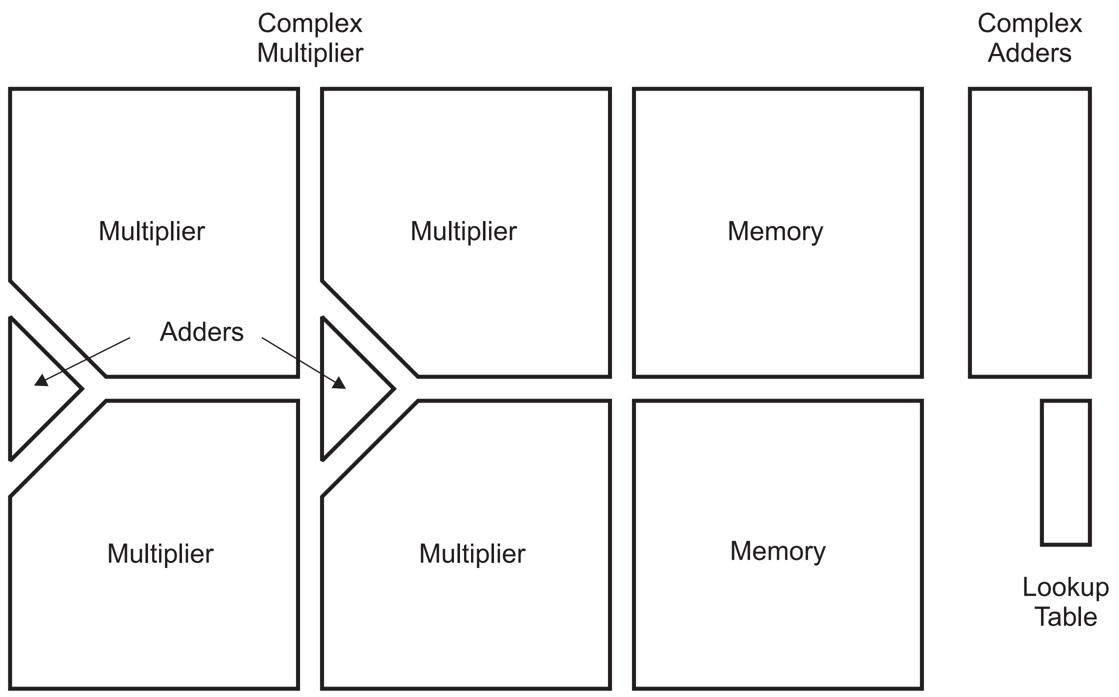


Figure 7.3: Functional units in 512-point FFT

stores the constant factors used in the multiplication stage. Not shown is the control logic necessary to generate the addresses for the memory units and reroute the memory lines at runtime. This logic could easily fit between the memory units and the adders.

In all, this implementation of the FFT uses 440 cells. It is estimated that each stage of the algorithm requires 316 cycles: 49 to send the first pair of points through the pipeline, 255 to process the remaining points, 7 to receive the most significant portion of the result, and 5 to partially reconfigure the memory units. Hence, 2844 cycles would be required for the nine stages.

Decreasing the word length and/or the number of samples used in the FFT would reduce the area and time requirements. Although not illustrated here, a 256-point FFT that operates on 16-bit data can fit inside a 16×8 block cells and requires approximately 1320 cycles. Lowering the word length in particular decreases the total area by a factor of four.

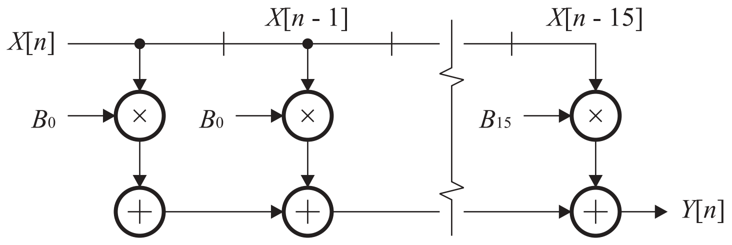


Figure 7.4: Structure of FIR filter

7.2 Mapping Finite Impulse Response Filters

Another common DSP benchmark is the finite impulse response (FIR) filter. Consider a 16-tap FIR filter that operates on 32-bit fixed-point data:

$$Y[n] = B_0 X[n] + B_1 X[n-1] + \ldots + B_{15} X[n-15]. \tag{7.3}$$

Figure 7.4 depicts the structure of the filter. The algorithm is ideally suited to the multiply-accumulate unit presented in Chapter 2, since each multiplication is paired with an addition.

The flexibility of the reconfigurable architecture allows system designers to trade off area and speed. A parallel implementation of the FIR filter using sixteen MAC units would require a large area, but achieve high processing power. Figure 7.5 illustrates the functional units for this case. Since a 32-bit MAC unit occupies 64 cells, the sixteen MAC units require 1024 cells. The estimated execution time for a 256-point data set is 691 cycles: 436 to fill the pipeline, and one cycle for each data point thereafter.

An alternative approach is to use the serial implementation diagrammed in 7.6. Here, the input data is initially loaded into a memory. The algorithm then computes

$$Y_0[n] = B_0 X[0] \tag{7.4}$$

for each sample n. The working memories on the input and output of the adder are then exchanged, similar to the operation of the FFT. In the next phase, the incremental result

$$Y_1[n] = B_1 X[1] + Y_0[n] \tag{7.5}$$

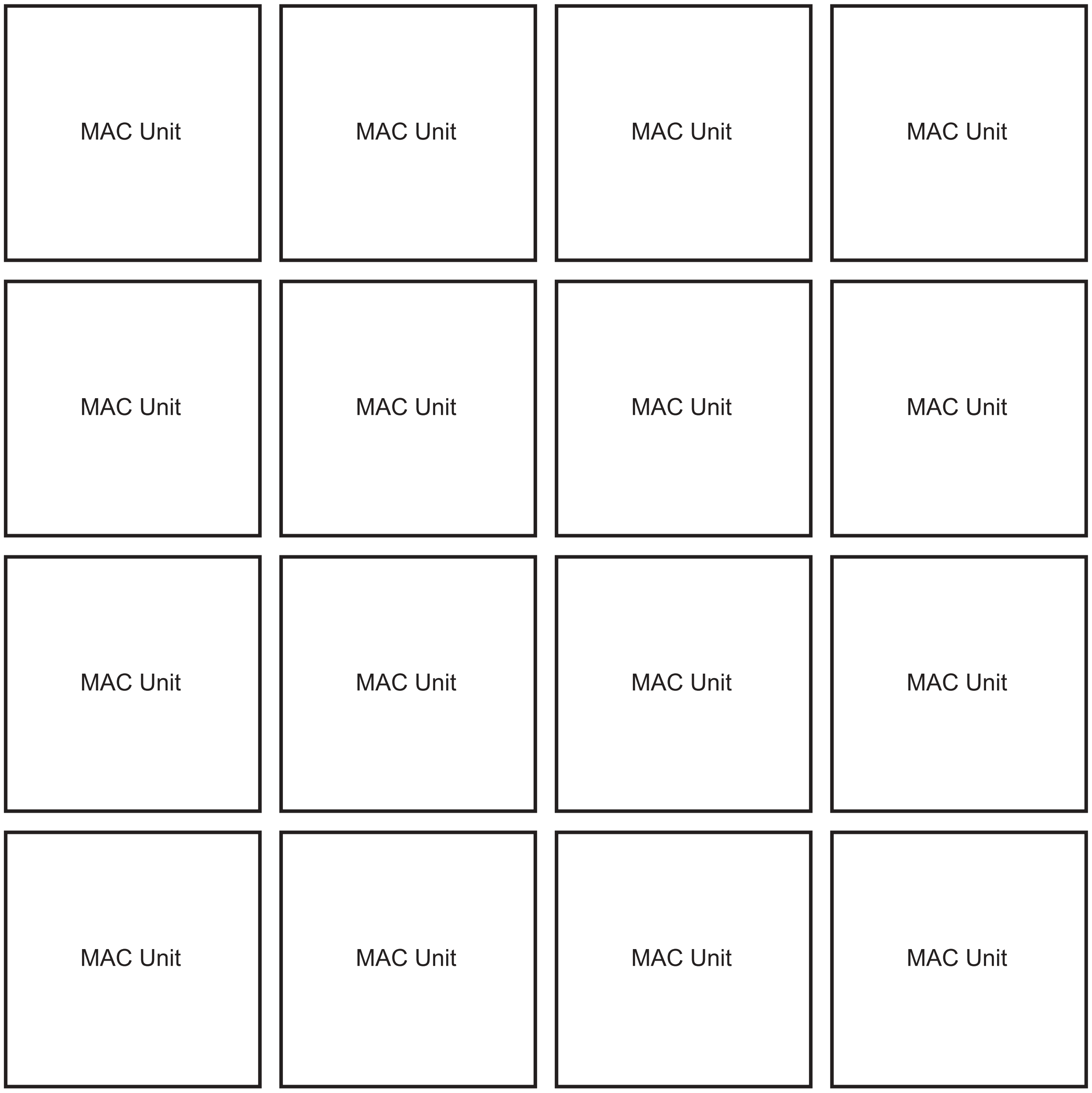


Figure 7.5: Functional units in parallel implementation of FIR filter (shown to scale)

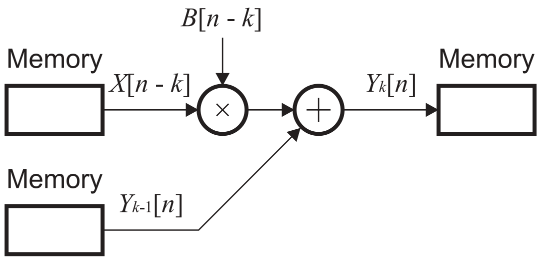


Figure 7.6: Diagram of serial implementation of FIR filter

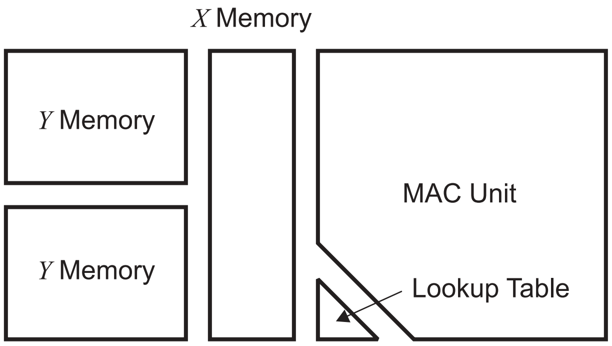


Figure 7.7: Functional units in serial implementation of FIR filter

is calculated for every n. Repeating this produce a total of sixteen times produces the desired results.

The functional units required for the serial implementation of the FIR filter are mapped out in 7.7. As in the FFT, the most significant bits of the multiplier can be discarded, assuming the coefficients of the filter are not large. The lookup table used to store these coefficients is placed in the available space. This implementation only requires 128 cells rather than 1024 for the first alternative. However, the execution time for a 256-sample input is 4704 cycles, becuase the algorithm must process the entire data set sixteen times.

7.3 Performance Evaluation

Table 7.1 shows the hardware requirements and estimated execution times for the FFT and the FIR filter. With a 200-MHz clock, the total latency for the 256-point FFT is only 6.6μs.

Table 7.1: Hardware Requirements and Execution Times of DSP Benchmarks

Benchmark	Word length	Cells	Cycles	Time
256-point FFT	16-bit real, 16-bit imag	124	1320	6.6 μs
512-point FFT	32-bit real, 32-bit imag	440	2844	14.2 μs
16-tap, 256-point FIR filter (serial)	32-bit real	128	4704	23.5 μs
16-tap, 256-point FIR filter (parallel)	32-bit real	1024	691	3.5 μs

Table 7.2: Execution Time of 256-Point FFT

Device	Cycles	Frequency	Time
ADSP-2188N [27]	7423	80 MHz	92.8 μs
ADSP-21532 [28]	3176	300 MHz	10.6 μs
TMS320VC5416-160 [29]	8542	160 MHz	53.4 μs
TMS320VC5502-300 [30]	4786	300 MHz	16.0 μs
Reconfigurable cell array	1320	200 MHz	6.6 μs

Table 7.2 compares the estimated execution time for the 256-point FFT with four commercial digital signal processors that operate on 16-bit fixed-point data. As shown, the reconfigurable architecture shows a speedup ranging from 1.6 to 14, demonstrating its great potential for DSP applications.

Chapter 8

Conclusion

This thesis has presented a novel two-level reconfigurable architecture for DSP. On the upper level, the architecture features an array of 4-bit cells and interconnection structures. On the lower level, each cell contains a 4$\times$4 matrix of elements that allows the cell to perform a wide variety of operations. The matrix of elements has two possible configurations: one optimized for mathematical functions and the other optimized for memory operations. Using the hierarchical interconnection structure, cells can be grouped into discrete functional units, such as adders, multipliers, and memory modules. Functional units can then be connected to implement DSP algorithms.

A prototype of the reconfigurable cell has been fabricated and tested for functionality. Transistor-level simulations indicate that the cell achieves a clock frequency of 200 MHz in a modest 0.25-μm technology. The theoretical execution times of several benchmark algorithms have been computed by manually mapping these algorithms onto the architecture and determining the total number of cycles required. For example, a 16$\times$8 array of cells can perform a 256-point FFT on 16-bit data in 6.6 μs. Contemporary digital signal processors require between 16.0 μs and 92.8 μs for the same operation.

8.1 Contributions

This research encompasses a variety of architectural innovations, including the following:

- **Two-level organization:** The reconfigurable architecture contains a two-level array of 4-bit cells and 1-bit elements [16], [17]. This approach allows the design to achieve the high performance required for binary arithmetic, as well as the high flexibility required for control logic. The fine-grain flexibility also permits cells to manipulate data in various data formats, such as unsigned and two's-complement.

- **Two-mode cell configurations:** Traditional fine-grain reconfigurable devices suffer from complex interconnection structures. In contrast, the 4×4 matrix of elements inside each cell can only assume two configurations [16], [17]. Mathematics mode is optimized for the 4-bit multiply-accumulate operation, and thus encompasses multiplication and addition as well. Memory mode allows embedded random-access memory and lookup tables to be distributed throughout the array of cells. Both modes can implement various logic and control functions.

- **Coarse-grain multiplication:** This research also incorporates a novel parallel multiplier structure that uses coarse-grain processing elements rather than 1-bit combinational logic blocks [18]. In this way, large multipliers and multiply-accumulate units can be implemented on the array of 4-bit cells. Unsigned multipliers require one universal element configuration; two's-complement multipliers require three additional element types.

- **Hierarchical interconnection structure:** The interconnection fabric used in the reconfigurable architecture recognizes that DSP algorithms are composed of discrete functional blocks [19]. Hence, the architecture provides a mesh of busses for data transfer within a functional unit, as well as a global H-tree for connecting functional units together. The higher levels of the H-tree manipulate data in larger units than 4 bits, allowing the inputs and outputs of functional units to be routed as a group.

- **Highly pipelined organization:** This is the first known study that applies super-pipelining to reconfigurable cell arrays. Each 4-bit cell pipelines all input and output data. This approach allows functional units to initiate one operation per clock cycle, dramatically increasing throughput. Pipeline latches are included in the H-tree as well so that interconnection latencies do not adversely affect the maximum clock frequency.

- **Flexible word length:** The array of 4-bit cells enables the target system to implement functional units of the precise size required. In addition, different functional units in the same algorithm are not constrained to using the same word length.

- **Orthogonal design space:** The reconfigurable architecture supports a large orthogonal design space whereby system designers can customize the word length, amount of parallelism, number of functional units, and functional unit connectivity to meet the needs of the application. For example, designers can choose a parallel or serial implementation of a digital filter. In this manner, systems can balance performance and flexibility requirements while minimizing development costs.

8.2 Future work

Further research on the reconfigurable architecture will focus on several areas. The primary goal will be to compare the performance and flexibility of the design to other implementations. This analysis will encompass work on both the hardware and software level. On the hardware level, additional prototype chips will be fabricated and tested to evaluate the performance of a small array of cells. On the software level, computer-aided design (CAD) tools will be developed to automate the placement and routing of DSP algorithms. The execution times of benchmark algorithms can then be calculated, similar to the initial results in Chapter 4.

Another direction for additional research will involve various enhancements to the reconfigurable architecture. For example, one alternative to the matrix of elements inside the cell would be to evaluate the 4-bit operations in a bit-serial manner. This approach would dramatically lower the area requirements, but may adversely affect the performance. The tradeoffs of such design changes will be explored.

Finally, the design space of DSP applications has expanded in recent years to include devices with specialized requirements, such as low power consumption and high reliability. Low power consumption is vital to wireless communication devices, whereas high reliability is crucial for many real-time monitoring systems. Hence, methods to lower the power requirements and increase the resilience of the device to faults will be developed.

Bibliography

[1] J. McClellan, R. Schafer, and M. Yoder, *DSP First: A Multimedia Approach*, Upper Saddle River, NJ: Prentice Hall, 1998, pp. 373–374.

[2] A.M. Chugg, "Ionising radiation effects: a vital issue for semiconductor electronics," *Engineering Science and Education Journal*, vol. 3, Jun 1994, pp. 123–130.

[3] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, Jun 2002, pp. 171–210.

[4] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: a survey," in *Programmable Digital Signal Processors*, Y. Hu, ed., Marcel Dekker Inc., 2001.

[5] N. Dutt and K. Choi, "Configurable processors for embedded computing", *IEEE Computer*, vol. 36, no. 1, Jan. 2003, pp. 120–123.

[6] M.A. Wahad and D.J. Puckey, "Reconfigurable DSP systems," in *Proc. IEE Colloquium on Applications Specific Integrated Circuits for Digital Signal Processing*, London, UK, Jun 1993, pp. 3/1–3/6.

[7] Altera Corporation, "Design Software & Development Kit Selector Guide," `http://www.altera.com/literature/sg/sg_dsdk.pdf`, Jun 2003.

[8] R. Hartenstein et al, "Mapping applications onto reconfigurable KressArrays," in *Proc. 9th International Workshop on Field Programmable Logic and Applications*, Glasgow, UK, Aug 1999.

[9] N.W. Bergmann and J.C. Mudge, "An analysis of FPGA-based custom computers for DSP applications," in *Proc. 1994 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Adelaide, Australia, vol. 2, Apr 1994, pp. 513–516.

[10] K. Rajagopalan and P. Sutton, "A flexible multiplication unit for an FPGA logic block," in *Proc. 2001 IEEE International Symposium on Circuits and Systems*, 2001, pp. 546–549.

[11] S.D. Haynes and P.Y.K. Cheung, "Configurable multiplier blocks for embedding in FPGAs," *Electronics Letters*, vol. 34, iss. 7, Apr 1998, pp. 638–639.

[12] R. Hartenstein, "Coarse grain reconfigurable architectures," in *Proc. 6th Asia South Pacific Design Automation Conference*, Yokohama, Japan, 2001, pp. 564–570.

[13] J. Smit et al, "Low cost and fast turnaround: reconfigurable graph-based execution units," in *Proc. 7th BELSIGN Workshop*, Enschede, Netherlands, 1998.

[14] P. Heysters et al, "A reconfigurable function array architecture for 3G and 4G wireless terminals," in *Proc. World Wireless Congress*, San Francisco, USA, 2002, pp. 399–405.

[15] A. Gunzinger, S. Mathis, and W. Güggenbuhl, "A reconfigurable systolic array for real-time image processing," in *Proc. 1988 International Conference on Acoustics, Speech, and Signal Processing*, New York, NY, Apr 1988, vol. 4, pp. 2054–2060.

[16] J. Delgado-Frias, M. Myjak, F. Anderson, and D. Blum, "A medium-grain reconfigurable cell array for DSP applications," in *Proc. 3rd IASTED International Conference on Circuits, Signals, and Systems*, Cancun, Mexico, May 2003, pp. 231–236.

[17] M. Myjak and J. Delgado-Frias, "A two-level reconfigurable architecture for digital signal processing," in *Proc. 2003 International Conference on VLSI*, Las Vegas, NV, Jun 2003, pp. 21–27.

[18] M. Myjak and J. Delgado-Frias, "Pipelined multipliers for reconfigurable hardware," in *Proc. 11th Reconfigurable Architectures Workshop*, Santa Fé, NM, Apr 2004, to be published.

[19] M. Myjak, F. Anderson, and J. Delgado-Frias, "H-tree interconnection structure for reconfigurable DSP hardware," in *Proc. 2004 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, Jun 2004, to be published.

[20] K. Leijten-Nowak and A. Katoch, "Architecture and implementation of an embedded reconfigurable logic core in CMOS 0.13 μm," in *Proc. 15th Annual IEEE International ASIC/SOC Conference*, Sep 2002, pp. 3–7.

[21] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed., San Francisco: Elsevier Science, 2003, pp. A-2–4.

[22] J. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, 2nd ed., Upper Saddle River, NJ: Pearson Education, Inc., 2003, pp. 591–592.

[23] J. Delgado-Frias and A. Widjaja, "An H-tree based configuration scheme for reconfigurable DSP hardware," in *Proc. 2004 International Conference on VLSI*, Las Vegas, NV, Jun 2004, to be published.

[24] C. Leiserson, "Universal networks for hardware efficient supercomputing," *IEEE Trans. on Computers*, vol. 34, iss. 10, 1985, pp. 892–901.

[25] A. DeHon, “Compact, multilayer layout for butterfly fat-tree,” *Proc. 12th ACM Symposium on Parallel Algorithms and Architectures*, Bar Harbor, ME, 2000, pp. 206–215.

[26] D. Blum, *VLSI implementation of cross-parity and modified DICE fault-tolerant schemes for reconfigurable hardware*, M.S. thesis, May 2004.

[27] Analog Devices Inc., “ADSP-21xx DSP Benchmarks,” `http://www.analog.com`.

[28] Analog Devices Inc., “DSP Benchmark Comparison,” `http://www.analog.com`.

[29] Texas Instruments Inc., “C54x DSP Benchmarks,” `http://www.ti.com`.

[30] Texas Instruments Inc., “C55x DSP Benchmarks,” `http://www.ti.com`.