A TWO-LEVEL RECONFIGURABLE ARCHITECTURE FOR DIGITAL SIGNAL PROCESSING

Mitchell J. Myjak and José G. Delgado-Frias

School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164 U.S.A.

Abstract

This paper describes a novel reconfigurable architecture for digital signal processing (DSP). The architecture consists of a two-level array of cells and interconnections. DSP algorithms are divided into 4-bit units and mapped onto the first level of cells. Each cell uses a 4x4 matrix of small elements to implement the basic operations required by the algorithm. Cells also contain pipeline latches for increased throughput. The two-level architecture features a simple VLSI implementation that combines the flexibility of memory elements with the performance of domino logic. The first prototype has been fabricated using a modest 0.5-μm CMOS technology. Circuit simulations indicate that the cell achieves a clock frequency of 100 MHz.

Keywords: Reconfigurable Systems Design, Two-Level Architecture, VLSI Systems, Digital Signal Processing.

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. Many of these applications impose critical requirements on performance, power consumption, and reliability, and thus require innovative hardware architectures to meet these specifications.

Digital systems may use several components to implement DSP, ranging from custom integrated circuits to general-purpose microprocessors. Application-specific integrated circuits (ASIC) achieve very high performance, but incur high development costs and only support one operation. General-purpose microprocessors can execute any software program, but offer relatively poor performance for DSP [1,2]. DSP-enhanced microprocessors achieve better results, but still do not approach the speed of an ASIC. Finally, reconfigurable devices attempt to combine the performance of an ASIC with the flexibility of a microprocessor. This approach has recently become practical for DSP, due to the exponentially increasing capabilities of VLSI systems.

Reconfigurable devices generally contain an array of programmable cells and interconnection structures. These devices have great flexibility and adaptability because designers can change the hardware configuration at any time, even after deployment [1,3,4]. In addition, the design process can be automated with the use of appropriate software tools. Traditional reconfigurable devices such as field-programmable gate arrays (FPGA) place little functionality in the cells. These fine-grain devices work well for implementing combinational or sequential logic. However, DSP algorithms use mathematical operations such as multiplication extensively. Mapping a multiplier onto a fine-grain device results in a complex structure that yields poor performance [5]. Recently, researches have proposed new reconfigurable devices that incorporate adders, multipliers, lookup tables, and other functional units in the cells [2]. These coarse-grain devices achieve good performance for mathematical functions, but may not provide a straightforward way to implement the control logic necessary for DSP algorithms. In addition, the fixed number of functional units limits flexibility.

This paper describes a novel reconfigurable architecture for DSP. In this approach, each cell consists of a 4x4 matrix of reconfigurable elements. Each element contains 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 a wide variety of 4-bit functions. In memory mode, the matrix of elements operates as a

64-byte memory. The resulting two-level architecture can implement the entire spectrum of operations required for DSP.

A previous paper presented a system-level overview of the reconfigurable architecture, comparing the scheme to other implementations [6]. Section 2 of this paper extends this work by showing how DSP operations can be mapped onto the device efficiently. Section 3 describes the design of the cell, explaining how cells perform basic DSP functions. Section 4 considers a simple VLSI implementation of the cell, showing how the circuit achieves high performance. Finally, Section 5 gives some concluding remarks.

2. Mapping DSP operations

This section provides a backdrop to the design of the reconfigurable architecture by considering several common DSP operations that the device must perform. The motivations for this discussion are twofold: to demonstrate that an array of 4-bit cells can implement DSP operations efficiently, and to show the functionality that each cell must have. For now, assume that each cell can perform any 4-bit operation and has unlimited interconnections to other cells.

2.1. 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. A straightforward solution, shown in Fig. 1, implements a carry-save multiplier [7] with 4-bit cells. Note that *A* and *B* are broadcast across the entire structure. This multiplier requires twenty cells: four that perform multiplication, four that perform addition, and twelve that perform both. The critical path involves eight cells.

A typical cell multiplies two four-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

$$y_{7:0} = (a_{3:0} \times b_{3:0}) + c_{3:0} + d_{3:0}. \quad (1)$$

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

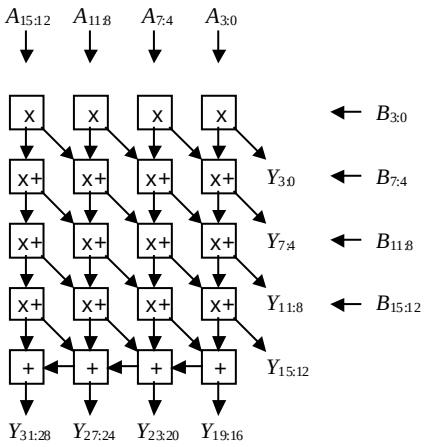

Fig. 1. Carry-save multiplier structure.

By rearranging the interconnection structure, it is possible to reduce the hardware required. Fig. 2 shows an improved multiplier that uses only sixteen cells and has a critical path of seven cells. The structure is easily scalable to form 4*n*-bit multipliers with n^2 cells.

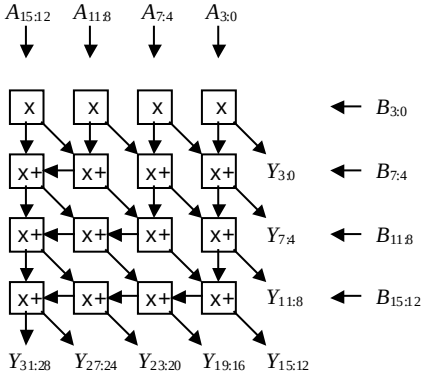

Fig. 2. Improved multiplier structure.

One benefit of dividing large operations into 4-bit units is the performance enhancement gained by pipelining [8]. Adding pipeline latches to the output of each cell enables the module to begin one multiplication per clock cycle. The clock frequency is constrained by the propagation delay of one cell plus associated interconnection structures. Fig. 3 shows the pipeline latches added to the multiplier. Although not shown, *A* and *B* need pipeline latches as well.

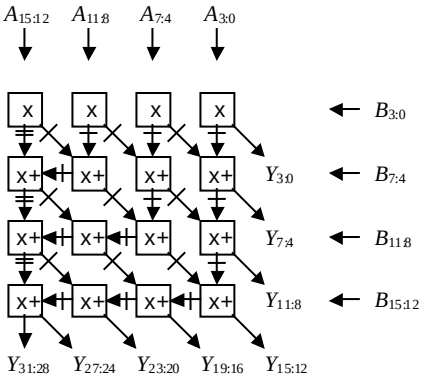

Fig. 3. Pipelined multiplier structure.

With appropriate changes to the function implemented by each cell, this structure can perform 2s-complement multiplication. Also note that two additional 16-bit terms, *C* and *D*, can be added into the top row of cells without increasing the hardware required. This modification would create a 16-bit multiply-accumulate (MAC) unit that evaluated the expression

$$Y_{31:0} = (A_{15:0} \times B_{15:0}) + C_{15:0} + D_{15:0}. \quad (2)$$

2.2. Addition

Most DSP algorithms require addition as well as multiplication. In many cases, an addition may be combined with another multiplication and implemented with the MAC unit described previously. For example, the finite impulse-response (FIR) filter equation is amenable to this simplification:

$$y[n] = b_0x[n] + b_1x[n-1] + ... + b_mx[n-m]. \quad (3)$$

However, other algorithms require separate addition units. The structure in Fig. 4 uses four cells to add two 16-bit numbers *A* and *B*. Each cell adds two four-bit portions of these numbers, named *a* and *b*, along with carry in *c*:

$$y_{4:0} = a_{3:0} + b_{3:0} + c_0. \quad (4)$$

A similar structure can be used to perform 2s-complement addition or subtraction. In general, adding or subtracting two 4*n*-bit numbers requires *n* cells.

The adder operates in a pipelined fashion for maximum performance. Note that *A* and *B* should arrive in a staggered fashion: the least-significant four bits in the first clock cycle, the next four bits in the second cycle, and so forth. Many of the structures described here have similar requirements.

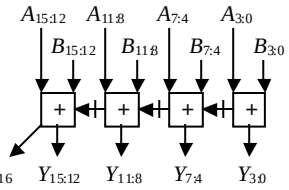

Fig. 4. Pipelined adder structure.

2.3. Memory operations

To implement DSP algorithms, the reconfigurable device must provide a way 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. This example suggests that cells should be able to perform memory operations as well as mathematical functions.

Suppose that each cell can implement an 8-bit, 64-entry memory. Fig. 5 shows the inputs and outputs of the cell in such a configuration. The lower two bits of the *e* input together with the *a* input specify the 6-bit address. Bit e_2 is a read enable, while bit e_3 is a write enable. This arrangement groups the signals into 4-bit units. Input *i* contains the 8-bit write data, while output *q* holds the 8-bit value read from the cell.

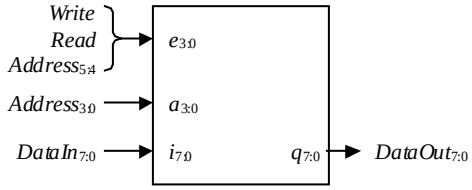

Fig. 5. 64×8-bit memory.

The cell can perform the following operations:

- Write: Memory[$e_{1:0}a_{3:0}$] ← $i_{7:0}$
- Read: $q_{7:0}$ ← Memory[$e_{1:0}a_{3:0}$]
- No-op: $q_{7:0} \leftarrow i_{7:0}$.

Passing the input data to the output data on a no-op simplifies the design of larger memory units. For example, consider the 16-bit, 256-entry memory in Fig. 6. The eight cells marked “M” store the actual data, while the four cells marked “D” decode the input address and control signals. The decoders activate one row of cells for each operation; inactive rows pass the input data to the next row. The memory operates in a pipelined fashion with a 4-cycle latency in order to not restrict the system clock frequency.

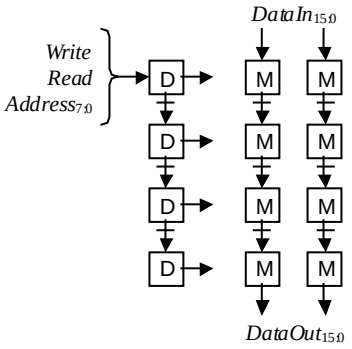

Fig. 6. 256×16-bit memory.

2.4. Logic and control operations

DSP operations are not composed exclusively 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.

Implementing control logic on a reconfigurable device requires good fine-grain flexibility. An architecture that places a fixed number of functional units in each cell may not be able to evaluate arbitrary logic expressions efficiently. However, an architecture that places only fine-grain functionality in each cell cannot execute complex mathematical operations efficiently. The next section of this paper proposes a solution to this dilemma.

3. Description of the design

At the top level, the architecture consists of an array of reconfigurable cells. Each cell performs operations in 4-bit units. The choice of 4-bit cells gives designers control over the word length and maximizes device utilization. In addition, having larger cells would increase the fan-in and fan-out of the gates, create signal integrity problems, and impede the datapath.

Fig. 7 depicts a portion of the array of cells. Actual devices might have a hundred to a thousand of these cells. Sixteen 4-bit busses connect each cell to its eight neighbors. The busses can be configured to transfer data in either direction. Analysis of DSP operations such as those in the last section indicates that this interconnection structure is sufficient.

As shown in Fig. 8, each cell contains four main components. The switch routes data between the cell and its neighbors. The interface contains buffers and pipeline latches to improve performance. The processing core implements the actual operations required for DSP. Finally, the control module handles the reconfiguration process. The following subsections discuss these components in more detail.

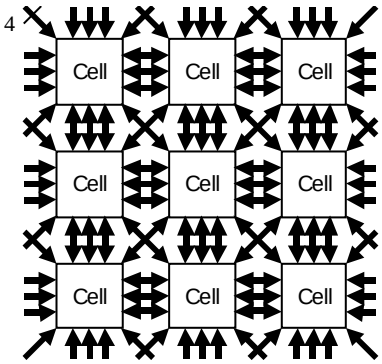

Fig. 7. Portion of reconfigurable cell array.

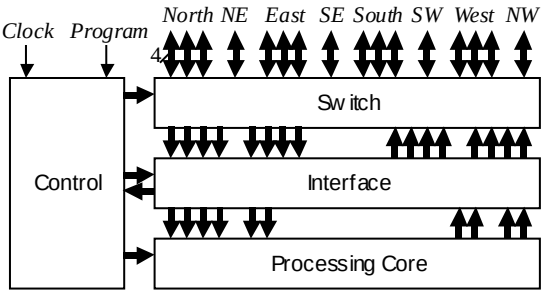

Fig. 8. Components of cell.

3.1. Switch and interface

The switch allows cells to transfer data to and from their eight neighbors in 4-bit units. Twelve busses connect cells in the horizontal and vertical directions, and four additional busses connect cells diagonally, bringing the total to sixteen. The direction of each bus is specified during the configuration process.

For maximum flexibility, the switch is a 16×16, 4-bit crossbar. The component selects up to eight incoming busses to transfer to the interface. The interface buffers the data before sending it to the core or back through the switch to another cell. The output of the core optionally travels through pipeline latches in the interface before reaching the switch. The switch then routes the data to the appropriate busses. Further details of the design of the switch appear in [9].

The pipeline latches in the interface allow DSP algorithms to execute in a pipelined or even superpipelined fashion. Without pipelining, the system clock rate would depend on the word length and operation type of each module in the device. With pipelining, the propagation delay through one cell is

the only restriction on the clock rate. In the architecture, each outgoing data line can be configured to go through zero, one, two, or three pipeline latches.

3.2. Processing core

The wide range of operations used in DSP places great demands on the capability of a cell. According to the analysis in Section 2, cells must be able to implement multiplication, addition, memory operations, and simple combinational and sequential logic. Assigning each operation to a separate functional unit would inevitably sacrifice performance or flexibility.

The core uses a novel design that consists of a 4×4 matrix of reconfigurable elements. Each element contains a 16×2-bit memory. The matrix can be configured into two structures: one optimized for memory operations, and the other optimized for mathematical functions. Both structures execute one operation per clock cycle.

In memory mode, shown in Fig. 9, the matrix of elements (E) implements a 64×8-bit memory. The operation mirrors that of the memory module discussed previously. Inputs *a*, *b*, *c*, and *d* specify a 4-bit address for each column of elements. The *e* input enables one of the columns for reading or writing. Lines *i* and *j* are the input and output data, respectively.

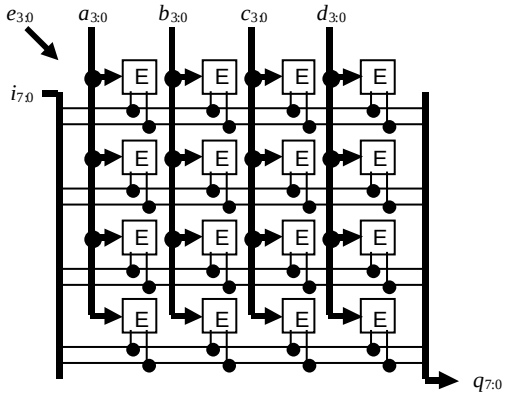

Fig. 9. Core in memory mode.

Typically, inputs *a*, *b*, *c*, and *d* have the same value. However, using separate inputs allows the same structure to implement a 4-way multiplexer.

In mathematics mode, shown in Fig. 10, the matrix of elements assumes a structure resembling a parallel multiplier. However, the elements can perform many functions besides multiplication. The 16×2-bit memory in each element now implements a lookup table for the desired function. Possible functions include signed and unsigned addition, subtraction, and multiplication, as well as rounding, shifting, and logic expressions.

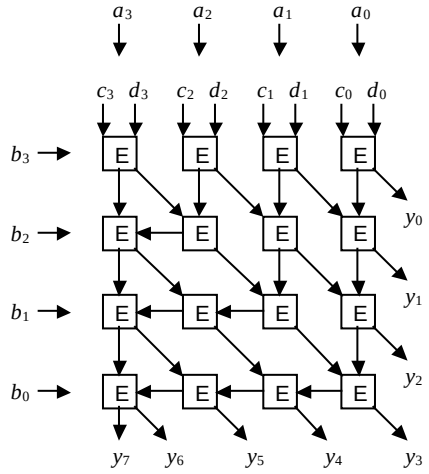

Fig. 10. Core in mathematics mode.

3.3. 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 asserts the *Program* pin of the device. Each cell enters a special mode that allows the target system to change the configuration of the switch, interface, and core. During this time, the core behaves as a random-access memory so that the end system can load information into the matrix of elements.

In all, the design of the cell combines the flexibility of a fine-grain architecture with the performance of a coarse-grain architecture. Table 1 lists some of the operations possible with suitable cell configurations.

TABLE 1. EXAMPLES OF CELL OPERATIONS

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

This discussion has assumed that the end system stores configuration data in an external memory and configures the device on power-up. Due to the regularity of DSP algorithms, many cells will have the same structure, reducing the external memory needed. However, the reconfigurable device could also contain nonvolatile memory for this purpose.

4. VLSI implementation

A notable feature of this reconfigurable 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 very high performance. For applications where reliability is also critical, error detection and correction circuitry can be added to the memory units, as described in [10]. The following discussion focuses on the basic implementation of the processing core.

4.1. Circuit design of element

Fig. 11 depicts the circuit design of one element in the processing core. Recall that each element contains a 16×2-bit memory. This memory is organized as a 4×4 array of 2-bit latches, with a row decoder and a column multiplexer. As shown in Fig. 12, each latch is a static random-access memory (SRAM) cell that only requires six transistors per bit.

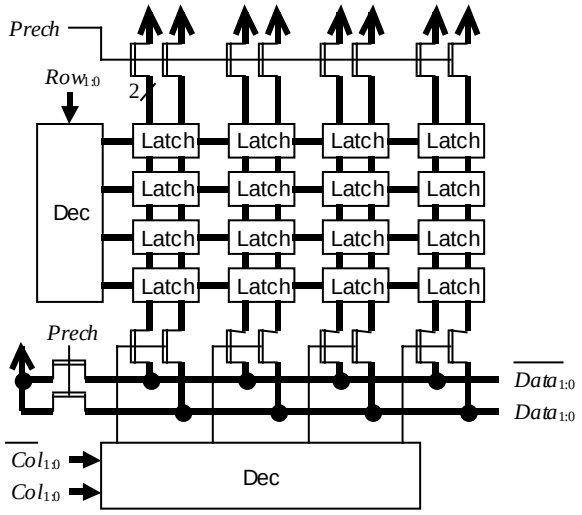

Fig. 11. Circuit design of reconfigurable element.

Each element works with a 4-bit address and a 2-bit data word. *Col* specifies the upper two bits of the address, and *Row* the lower two bits. Depending on the current mode, *Col* and *Row* may come from different sources in the matrix of elements. The *Data* and $\overline{Data}$ lines are bidirectional, and contain the input data for a write operation and the output data for a read operation.

For a read operation in memory mode, the element first precharges all internal data lines to $V_{DD} - V_{tn}$, where V_{tn} is the threshold voltage of an n-transistor. Then, the row decoder enables one row of latches, and the column decoder connects one column of latches to *Data* and $\overline{Data}$. The latch at the selected row and column discharges *Data* or $\overline{Data}$ to ground, depending on the stored value. For example, if the latch contains 01, it will discharge $Data_1$ and $\overline{Data_0}$. The cell uses these lines to set the read data *q*.

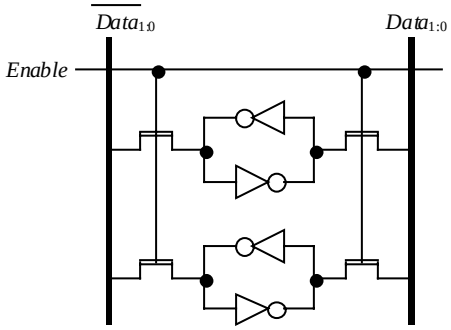

Fig. 12. 2-bit latch.

A read operation in mathematics mode works in a similar fashion, except that *Data* and $\overline{Data}$ connect to the *Col* and $\overline{Col}$ inputs of the next element. As shown in Fig. 13, the column decoder uses special circuitry to generate the column select signals. Initially, *Col* and $\overline{Col}$ are precharged to V_{DD}, turning all column select signals off. As soon as the previous element discharges *Col* or $\overline{Col}$, one of the NOR gates activates and selects the corresponding column. This circuitry creates a domino effect that allows the matrix of elements to evaluate mathematical operations rapidly.

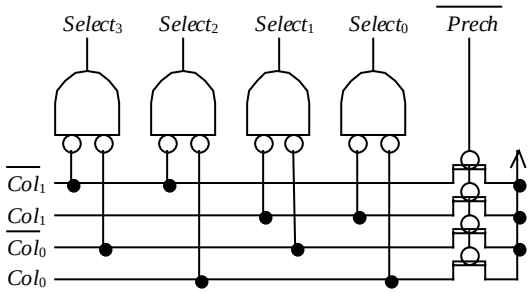

Fig. 13. Column decoder.

For a write operation, which can only occur in memory mode, the element begins by precharging all internal data lines to $V_{DD} - V_{tn}$, as before. The row and column decoder select a target latch based on the input address. However, now the element drives the write data onto the *Data* line, and the complement of the write data onto $\overline{Data}$. Note that the *i* input of the cell specifies the write data. The n-type pass transistors in the element now run in reverse, driving the new data into the selected latch.

4.2. Performance

The operation of a cell that determines the maximum clock frequency is a read operation in mathematics mode. Recall from Section 3 that the critical path of the matrix of elements involves one element in memory mode, but seven elements in mathematics mode. The circuitry that performs this critical operation has been optimized for speed. For example, the circuitry that decodes *Col* features domino logic.

Simulations of the reconfigurable cell show that it can operate with a clock period of 10 ns using a modest 0.5-μm CMOS technology. This technology, supported by MOSIS and available for academic purposes, has been chosen to build the first prototype of the cell. Due to the simplicity of the VLSI implementation and the availability of pipeline latches, the system can reach a clock frequency of 100 MHz even with this modest technology.

Fig. 14 shows an actual simulation of the matrix of elements. In this simulation, the output *y* changes from $0111\ 0000_2$ to $1000\ 1111_2$ on the falling edge of *Clock*. Bit y_0 transitions after 2 ns, followed rapidly by y_1, y_2, and so on up to y_6 and y_7. The total propagation delay through the matrix of elements is around 7 ns. Allowing 3 ns for the delay of the interconnection structures, the reconfigurable architecture can achieve a clock rate of 100 MHz. The resulting performance for several DSP benchmarks compares to the highest-performance DSP processors today [6].

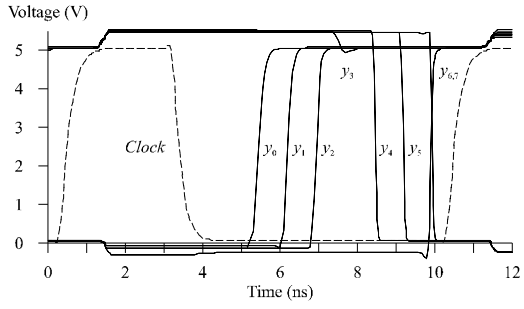

Fig. 14. Simulation of the matrix of elements.

5. Concluding remarks

This paper has presented a novel reconfigurable architecture for digital signal processing. The architecture features a two-level hierarchy of programmable cells and elements. Each cell contains a 4×4 matrix of elements that allow the cell to perform a wide variety of operations. The matrix of elements has two possible configurations optimized for mathematical functions and memory operations, respectively. Cells can then be interconnected to implement DSP algorithms. Adding pipeline latches between cells dramatically improves the performance. A prototype of the architecture has been fabricated in 0.5-μm technology. Simulations show that the prototype can operate at 100 MHz even with this modest technology.

The reconfigurable architecture has many advantages over other DSP implementations. Unlike commodity processors, system designers can customize the word length, amount of parallelism, and interconnection structure to meet the demands of the application. Unlike an ASIC, the device can be reprogrammed any number of times as the needs of the application change. The performance of the device may not approach that of an ASIC, but the reduced development costs alone make the reconfigurable architecture a viable candidate for many applications.

6. References

[1] 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.

[2] R. Hartenstein, "Coarse grain reconfigurable architectures (embedded tutorial)," in *Proc. 6th Asia South Pacific Design Automation Conference*, Yokohama, Japan, 2001, pp. 564-570.

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

[4] 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.

[5] 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.

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

[7] J. Rabaey et al, *Digital Integrated Circuits: A Design Perspective*, 2nd ed., Upper Saddle River, NJ: Pearson Education, Inc., 2003, pp. 591-592.

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

[9] F. Anderson and J. Delgado-Frias, "A reconfigurable crossbar switch for a DSP array," in *Proc. 2003 Int. Conference on VLSI*, Las Vegas, NV, 2003.

[10] D. Blum and J. Delgado-Frias, "A fault-tolerant memory-based cell for a reconfigurable DSP processor," in *Proc. 2003 Int, Conference on VLSI*, Las Vegas, NV, 2003.