

# ARCHITECTURAL MODIFICATIONS FOR OPTIMIZING MAPPINGS ON MORPHOSYS RC-SYSTEM

Issam Damaj, and Hassan Diab  
[issamwd@ieee.org](mailto:issamwd@ieee.org); [diab@aub.edu.lb](mailto:diab@aub.edu.lb)  
American University of Beirut  
Department of Electrical and Computer Engineering  
Faculty of Engineering and Architecture  
Beirut, Lebanon  
P.O. BOX 110236

## Abstract

*This paper presents architectural modifications to the reconfigurable part of MorphoSys, a reconfigurable computing (RC) system [2], which improves system performance [1-6]. RC is introduced, followed by the MorphoSys (M1) RC system. Moreover, two algorithms are mapped to the system based on the enhanced interconnection scheme and ALU complexity. The algorithms include a computer graphics acceleration algorithm to which ALU enhancement to M1 was considered, and an FIR filter algorithm to which interconnection enhancement to M1 was applied. The results presented indicate an improved performance. A spreadsheet model is used to present the modification and the performance improvement. The speedup achieved is explained as well as the advantages in the mapping of the application.*

**Keywords** Reconfigurable Computing, Parallel Algorithms, Graphics Acceleration, MorphoSys

## 1. Introduction

Reconfigurable computing is an intermediate solution between general-purpose processors (GPPs) and application specific integrated circuits (ASICs). Because it employs hardware that is programmable by software, RC has advantages over both general-purpose processors and ASICs. It provides a faster solution than a GPP because the function that the hardware performs and its interconnection are both specified through software that maps the application onto the reconfigurable hardware. Also, it has a wider applicability than ASICs since its configuring software makes use of the broad range of functionality supported by the reconfigurable device. It is also cheaper than an ASIC solution since it can use only one piece of hardware to perform the different sub-tasks in a heterogeneous application instead of employing an ASIC for each of those sub-tasks [3]. Achieving higher speeds than GPPs and having more flexibility than

ASICs, RC is finding its way more and more into research efforts and hardware devices.

## 2. MorphoSys Design

One of the emerging RC systems includes the MorphoSys (M1) designed and implemented at the University of California, Irvine. It has the block diagram shown in Figure 1. It is composed of: 1) an array of reconfigurable cells called the RC array, 2) its configuration data memory called context memory, 3) a control processor (TinyRISC), 4) a data buffer called the frame buffer, and 5) a DMA controller [2]. A program runs on M1 in the following manner: General-purpose operations are handled by the TinyRISC processor, while operations that have a certain degree of parallelism, regularity, or intensive computations are mapped to the RC array.

The TinyRISC processor controls, through the DMA controller, the loading of the context words to context memory. These context words define the function and connectivity of the cells in the RC array. The processor also initiates the loading of application data, such as image frames, from main memory to the frame buffer. This is also done through the DMA controller. Now that both configuration and application data is ready, the TinyRISC processor instructs the RC array to start execution. The RC array performs the needed operation on the application data and writes it back to the frame buffer. The RC array loads new application data from the frame buffer and possibly new configuration data from context memory. Since the frame buffer is divided into two sets, new application data can be loaded into it without interrupting the operation of the RC array. Configuration data is also loaded into context memory without interrupting RC array operation. This causes M1 to achieve high speeds of execution [3].

As stated earlier, the reconfigurable device in M1 is the RC array divided into four quadrants. It has the design and interconnection shown in Figure 2 [2]. The interconnection network is built on three hierarchical levels. The first is a nearest neighbor layer that connects the RCs in a 2-D mesh. The second is an intra-quadrant connection that connects a

specific RC to any other RC in its row or column in the same quadrant. The third is an inter-quadrant connection that carries data from any one cell (out of four) in a row (or column) of a quadrant to other cells in an adjacent quadrant but in the same row (or column) [4].

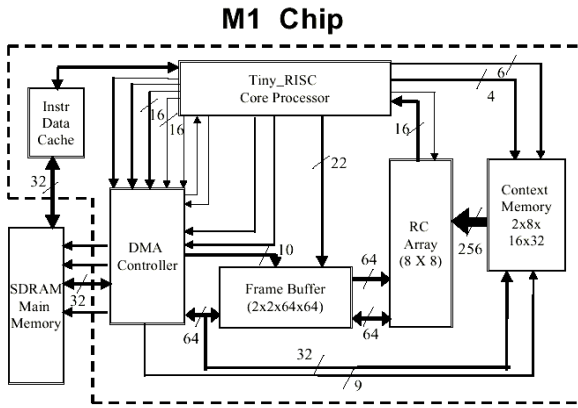


Figure 1. MorphoSys (M1) Block Diagram.

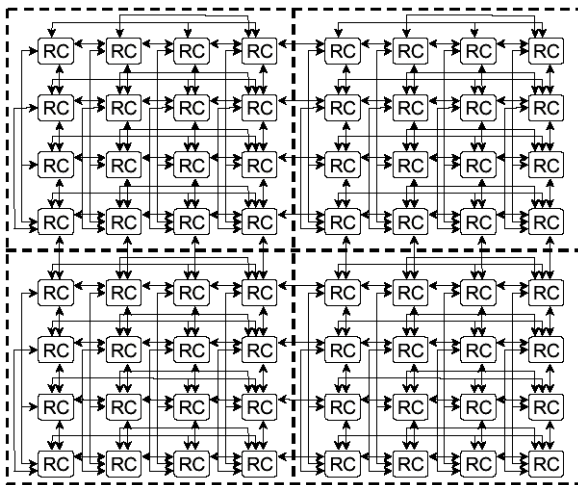


Figure 2. RC Array Interconnection.

The context words present on context memory configure the function of the RCs as well as the interconnection, thus specifying where their input is from and where their output will be written [5]. M1 is designed so that all the cells in the same row perform the same function and have the same connection scheme (in row context broadcast mode), or all the cells in the same column perform the same function and have the same connections scheme (in column context broadcast mode). All the cells of a row or of a column share the same configuration [5].

### 3. Graphics Acceleration Algorithms

Geometrical transformations are a fundamental part of computer graphics. Transformations are used to position,

shape, and change viewing positions of objects, as well as change how they are viewed. The main usage of the M1 is, as any parallel processor, to perform fast computations of intensive algorithms that need a certain computational power requirements. The Majority of algorithms that employ geometrical transformations could be classified in this area [5-6]. Translation, scaling, rotation, and shear are the basic part of any geometrical transformation algorithm. Also the composition of those transformations is quite beneficial for fast computation.

#### 3.1. Composite Scalings Algorithm

In this paper we will pinpoint the need of some architectural modifications after introducing a composite scaling algorithm. The composition of two scalings would mean either to: 1) calculate a composite scaling parameter and then to perform the scaling, or 2) to do two successive scalings i.e. to call twice the subroutine suggested in [6] (scalar-vector multiplication routine). This is similar to the case of any other geometrical transformation. Generally, to construct a subroutine, for two successive transformations, is useful for code reusability, especially, if we could obtain a major saving in execution time. To perform the first way of performing two scalings we must first calculate the composite scaling parameter. This scaling parameter is the result of multiplying the two scaling factors. Then, to perform the scaling of the points, which is also the multiplication of the scaled points with the composite scaling factor. Using the M1 ALU basic operations the steps involved are:

$$\text{Out}(t) = C1 \times C2.$$

$$\text{Out}(t+1) = \text{Out}(t) \times A.$$

Or, in another way:

$$\text{Out}(t) = C1 \times A.$$

$$\text{Out}(t+1) = C2 \times \text{Out}(t).$$

Where  $C1$  and  $C2$  are constants, while  $A$  is a vector of inputs.

#### 3.2. ALU Enhancement

Both the previous steps require two unsupported operations ( $\text{Out}(t+1) = C \times \text{Out}(t)$ ,  $\text{Out}(t+1) = A \times \text{Out}(t)$ ) in the ALU of the M1 RC-Array. This led us to start a new research that aims to try solving this problem. We divided our aims into two categories: 1) the possibility to modify the ALU operations to support the operation  $\text{Out}(t+1) = \text{Out}(t) \times A$  or  $\text{Out}(t+1) = C \times \text{Out}(t)$ . 2) The possibility to implement the operation that will lead to the operation  $\text{Out}(t) = B \times A$ .

Discussing only the first modification that assumes the presence of the operation  $\text{Out}(t+1) = C2 \times \text{Out}(t)$ , i.e. by allowing the multiply operation in the ALU per cell. The code after this modification is shown in Table 1.

0:	ldui	r1, 0x1;	R1 ← 1000 <sub>hex</sub> . This is where vector A is stored.
1:	ldfb	r1, 0, 0, 16 ;	FB ← 16 x 32 bits at set 0, bank A, address 0.
2:	add	r0, r0, r0;	No-operation.
...	...	...	...
33:	ldui	r3, 0x4;	R3 ← 4000 <sub>hex</sub> . where the context word is in mem.
34:	ldetxt	r3, 0, 0, 0, 1;	Load one context word from main memory starting at the address stored in register 3 into plane 0, block 0 and starting at word 0.
35:	Add	r0, r0, r0;	NOP
...	...	...	...
38:	ldui	r3, 0x5;	R3 ← 5000 <sub>hex</sub> . where the context word is in mem.
39:	ldetxt	r3, 0, 0, 1, 1;	Load another context word from main memory starting at the address stored in register 3 into plane 1, block 0 and starting at word 0.
40:	Add	r0, r0, r0;	NOP
...	...	...	...
42:	ldui	r4, 0x0;	R4 ← 0000 <sub>hex</sub> .
43:	sbc	1, 0, 0, 0, 0, 0, 0x0;	Single bank column broadcast causing all the cells in the RC array to perform their operations specified by the context word in context memory starting with data from frame buffer, set 0, bank A, address offset 0.
44:	sbc	1, 0, 0, 0, 0, 0, 0x40;	It sends data from bank at address 40 <sub>hex</sub> in FB.
45:	sbc	1, 0, 0, 0, 0, 0, 0x80;	It sends data from bank at address 80 <sub>hex</sub> in FB.
46:	sbc	1, 0, 0, 0, 0, 0, 0xC0;	It sends data from bank at address C0 <sub>hex</sub> in FB.
47:	sbc	1, 0, 0, 0, 0, 0, 0x100;	It sends data from bank at address 100 <sub>hex</sub> in the FB.
48:	sbc	1, 0, 0, 0, 0, 0, 0x140;	It sends data from bank at address 140 <sub>hex</sub> in the FB.
49:	sbc	1, 0, 0, 0, 0, 0, 0x180;	It sends data from bank at address 180 <sub>hex</sub> in the FB.
50:	sbc	1, 0, 0, 0, 0, 0, 0x1C0;	It sends data from bank address 1C0 <sub>hex</sub> in the FB.
51:	sbc	1, 0, 0, 1, 0, 1, 0x0;	Single bank column broadcast causing all the cells in the RC array to perform their operations specified by the new suggested context word (with ALU enhancement) in context memory causing Out (t+1) = C2 x Out (t). Where, no data is taken from the frame buffer.
52:	sbc	1, 0, 0, 1, 0, 1, 0x0;	
53:	sbc	1, 0, 0, 1, 0, 1, 0x0;	
54:	sbc	1, 0, 0, 1, 0, 1, 0x0;	
55:	sbc	1, 0, 0, 1, 0, 1, 0x0;	
56:	sbc	1, 0, 0, 1, 0, 1, 0x0;	
57:	sbc	1, 0, 0, 1, 0, 1, 0x0;	
58:	sbc	1, 0, 0, 1, 0, 1, 0x0;	
59:	wfbi	0, 0, 0, 1, 0x0;	
60:	wfbi	1, 0, 0, 1, 0x40;	of column 1 into set 1, address 64.
61:	wfbi	2, 0, 0, 1, 0x80;	of column 2 into set 1, address 128.
62:	wfbi	3, 0, 0, 1, 0xC0;	of column 3 into set 1, address 192.
63:	wfbi	4, 0, 0, 1, 0x100;	of column 4 into set 1, address 256.
64:	wfbi	5, 0, 0, 1, 0x140;	of column 5 into set 1, address 320.
65:	wfbi	6, 0, 0, 1, 0x180;	of column 6 into set 1, address 384.
66:	wfbi	7, 0, 0, 1, 0x1C0;	of column 7 into set 1, address 448.

Table 1. TinyRISC code, composite scalings (1 step).

In the suggested algorithm firstly (i.e. Out (t) = C1xA) an available vector scalar context word is used, then in the next step the suggested context word is used. Vector A represents the points to be scaled, C1 represents the first scaling factor in scalar form passed thru the context

word, C2 is the second scaling factor passed thru the new suggested vector-scalar context word.

### 3.3. Performance Analysis

Performance evaluation and comparisons is made between the new mappings/designs and repeatedly using those discussed in [6]. This performance is based on the speed of execution of the algorithms. The M1 system is considered to be operational at a frequency of 100 MHz.

The algorithm in table 1 takes 66 cycles in order to terminate. So we can finish calculating the desired output for the 64-element vector after 66 cycles under the M1. The cycle time for the M1 is known to be equal to 1/100 MHz i.e. the cycle time is equal to 10 nsec. Thus, the speed in elements per cycle of the algorithm of table 1 is equal to 0.96 elements/cycles. Accordingly, the time for the algorithm to terminate is equal to 0.66 Microseconds. Note that this algorithm performs two successive scalings in one step. Performing this transformation with calling the routines in [6] twice (i.e. applying two successive scalings in two steps), the results are as following for 64-element vectors scaling.

- Total number of cycles: 106 cycles.
- Cycles Per Elements: 1.667.

Comparing these results with the new composite routine in Table 1, we show the superiority of the new proposed algorithm, thus, the advantage of the suggested ALU-Functional modification. Tables 3 and 4 shows comparisons between the two algorithms under M1 and also with the routines in [6] under the Intel 80386 and Pentium microprocessors. The speedup factor in Table 3 is considered to be the ratio of number of cycles of the other suggested systems over the number of cycles of the M1. In addition, the percentage speedup factor in Table 4 is considered to be the ratio of number of cycles of algorithm 1 over the number of cycles of algorithms 2. The same algorithm of Table 1 is shown in Table 2, but coded with the Intel instruction set.

### 4. FIR Filter Basic Mapping

The FIR filter has the following equation:

$$y_k = \sum_{j=0}^{N-1} x_{k-j} w_j, \text{ or}$$

$$y_k = x_k w_0 + x_{k-1} w_1 + x_{k-2} w_2 + \dots + x_{k-N+1} w_{N-1}$$

N being the order of the FIR filter. The basic FIR filter mapping suggested in [1] has the model shown in Figure 4. The idea is general enough though to be applied to an FIR filter of any order and an RC array of any size.

Label			Description	Clocks		
				80486	Pentium	
	MOV	SP, C1_Loc	SP ← Location of C1 in mem.	1T	1T	
	MOV	BP, C2_Loc	BP ← Location of C2 in mem.	1T	1T	
	MOV	SI, V1_Loc	SI ← Location of V1 in memory.	1T	1T	
	MOV	DI, Result_Loc	SP ← Loc of the result in mem.	1T	1T	
	MOV	CX, Count_Value	SI ← Value to count down 64.	1T	1T	
AA:	MOV	AL, [SP]	Get the vector element addressed by the SP register.	1T	1T	
	MOV	BL, [BP]	Get the vector element addressed by the BP register.	1T	1T	
	MUL	BL	AX ← AL * BL.	42T	11T	
	MOV	BL, [SI]	Get the vector element addressed by the SI register.	1T	1T	
	MUL	BL	AX ← AL * BL.	42T	11T	
	MOV	[DI], AX	Store the value in AX in the memory location addressed by DI register.	1T	1T	
	INC	SP	Get next element of C1.	1T	1T	
	INC	BP	Get next element of C2.	1T	1T	
	INC	SI	Get next element of V1.	1T	1T	
	INC	DI	Increment the destination.	1T	1T	
	DEC	CX	Decrement the counter.	1T	1T	
	JNZ	AA	Jump of not finished to AA.	3/1T	1T	
Calculations (64-elements vectors).				TOTAL Time States	6147T	2053
				Frequency	100MHz	133MHz
				For a 64-element vector	61.47µs	15.4µs
				Cycles Per Elements		
				For a 64-element vector	96	32

Table 2. Code and analysis of composite Scalings algorithms using the 80486, and Pentium instruction set.

Algorithm	System	Measure				
		N# of Cycles	Total Time in Micro Seconds	Speedup	Element Per Cycles	Cycles Per Element
Combined Scalings "64-Elements".	M1	66	0.66		0.96	1.03
	Pentium	2053	15.4	31.1	0.031	32
	80486	6147	61.47	93	0.01	96
Combined Scalings "32-Elements".	M1	40	0.4		0.8	1.25
	Pentium	1029	7.73	25.7	0.031	32.25
	80486	3107	31.07	77.6	0.01	96

Table 3. Comparisons with other systems.

Algorithm	N# of Cycles	Measure		
		% Speedup	Elements Per Cycle	Cycles Per Element
Two Scalings in Two Steps "64 Elements" Algorithm 1	110	67 %	0.64	1.71
Two Scalings in One Steps "64 Elements" Algorithm 2	66		0.96	1.03
Two Scalings in Two Steps "32 Elements" Algorithm 1	60	50 %	0.53	1.89
Two Scalings in One Steps "32 Elements" Algorithm 2	40		0.8	1.25

Table 4. Results from the MorphoSys emulator.

#### 4.1. FIR Filter Optimized Mapping Onto Enhanced MorphoSys Interconnection

A proposed addition to the interconnection network is a simple one. Its goal is to enhance the performance by allowing diagonally connected cells in the array to exchange data. This new modification leads to enhancements in performance. The basic FIR filter mapping in [1] has the RCs using port A to access the data bus and port B to access the output of the cell to the left of the specific RC. Port B accesses the output of the cell to its top, bottom, or left. In this paper, we suggest the addition of a connection that allows port B to access the cell at its lower left corner (diagonally). This is shown in Figure 3 for only one sample cell. This will allow data to travel from a certain cell to a diagonal cell on its upper right corner. This shall prove to enhance performance for the FIR application.

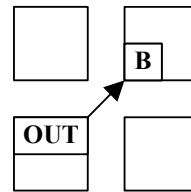


Figure 3. Suggested Interconnection Addition.

A new mapping is introduced that will take advantage of this new interconnection. The constraint here is that the maximum order that the filter can assume needs to be less than the RC length or width. In an 8x8 RC array M1 configuration, for example, an FIR filter with a maximum order of 7 can be mapped. In a 16x16 RC array configuration, a maximum order of 15 can be mapped and so on. This constraint will become clearer as the mapping is explained.

The new model is explained via spreadsheets for a 3-tap filter in a 4x4 RC array M1 for simplicity and clarity. The same concept may, however, be applied to a filter of any order and an RC array of any size. Figure 5 shows the suggested mapping. In this mapping, the last column is not made use of. So, a 4x3 RC array configuration could work, but since the RC array is usually of equal length and width, then a 4x4 RC array is considered in this specific case and an NxN RC array in general for a filter of maximum order (N-1).

Column 0 (spreadsheet column F), as in the case of the mapping in [1], multiplies the input by weight w2. Columns 1 and 2 multiply the input by weights w1 and w2 respectively and add the result to the output of the cell on its lower left corner. This is where the new interconnection is made use. The last column, column 3 in this case, is not needed. Note that the cells in the last row of columns 1 and

2 do not have a cell to their lower left corner, so the result of the addition is added to zero. Notice also that the order of the input elements is sequential with no need for rearrangement. Looking at column L in the model, i.e. column 2, one can see that the first two cells in each cycle hold the formulas for two output vector elements. The specific elements can be seen in column Z. This diagonal communication from a cell to the cell on its upper right corner can be seen clearly throughout the figure.

## 4.2. Performance Improvement

This mapping produces two results every cycle compared to the basic mapping in [1] that achieved a rate of one result per cycle. This obviously doubles the speed of producing output elements at the last column, and show the positive effect of the suggested interconnection. Assuming that the RC has only two input ports, then extending the connection to include newer cells would not enhance the performance any further because those new interconnections would not be made use of. With an extended interconnection, further enhancement on the speed can be achieved if each cell is provided with three input ports and an ALU-multiplier adder of also three inputs that can perform a multiplication of one input and addition with the two other inputs.

## 4.3. Performance Analysis

For the basic mapping of the FIR filter in [1], N results are obtained every N cycles. Thus, the rate of output calculation is 1 per cycle. In the case of the improved mapping, 2 results are obtained every cycle giving a rate of 2 samples per cycle. Thus, a speedup of 2 is achieved in the improved design over the basic design. This speedup is in the production of output samples in the contents of the RCs of the last column in the RC array. It does not take into consideration the time taken to write back the results to the frame buffer. For the basic FIR filter mapping in [1], write back is made once every N cycles, thus producing N sample outputs every (N+1) cycles. The speed obtained is  $N/(N+1)$ . For the FIR filter mapping on the improved design, write back is made once every cycle, thus producing 2 output samples every 2 cycles with a speed of 1 sample per cycle. Table 5 shows the number of samples obtained for a specific number of cycles for both the FIR filter basic mapping (BM) in [1] and the mapping on the improved design (IM) proposed above. Table 5 also includes the speeds achieved for the FIR filter of orders 8, 16, 32 and 64. The speed is in Million Samples per Cycle (MSPS) assuming that M1 is running at a frequency of 100 MHz. The FIR filter mapping on the improved design achieves a fixed speed of 100 MHz for a FIR filter of any order. Thus, it

has better performance than the basic FIR filter mapping achieving speeds between 88 and 98 MSPS for filter orders between 8 and 64. The speedup factor in Table 5 is the ratio of the MSPS value of the improved mapping over the basic mapping.

	BM		IM		MSPS		
	Samples (N)	Cycles (N+1)	Samples (2)	Cycles (2)	BM	IM	% Speedup
8-tap FIR	8	9	2	2	88.89	100	12.4%
16-tap FIR	16	17	2	2	94.12	100	6.2%
32-tap FIR	32	33	2	2	96.97	100	3.1%
64-tap FIR	64	65	2	2	98.46	100	1.5%

Table 5. Speeds of Basic and Improved Mappings.

## 5. Conclusion

The suggested enhancements allow more flexibility in mapping not only the presented algorithms, but also potentially on many others. The presence of a multiplication operation in the ALU of each RC gives the capability of processing multiplication operation in parallel, instead of relying on the TinyRISC single processor. Moreover, the diagonal interconnection modification provides more flexibility in mapping algorithms, which would lead also to a better overall performance. All in all, an optimized mapping of the FIR filter technique was proposed. This mapping is tested and compared to the basic mapping with respect to speedup, speed of output sample production, and speedup factor. Good speedup factors were achieved by the optimized mapping over the basic mapping reaching 12.4% for an 8-tap FIR. Furthermore, a new mapping for computer graphics was introduced showing the benefits and the superiority of the suggested modifications. High speedup factors were achieved reaching 66.7 % for comparisons within the same system (M1).

## References

- [1] E. Abdenmour, H. Diab, and F. Kurdahi. FIR Filter Mapping and Performance Analysis on MorphoSys. *In the Proceedings of the 7<sup>th</sup> IEEE International Conference on Electronics, Circuits and Systems*, Lebanon, Dec. 2000.
- [2] N. Bagherzadeh, F. Kurdahi, H. Singh, G. Lu, M. Lee, and E. Filho. MorphoSys: A Parallel Reconfigurable System. *In the Proceedings of Euro-Par 99*, Toulouse France 1999.
- [3] R. Maestre, F. Kurdahi, N. Bagherzadeh, H. Singh, R. Hermida, and N. Fernandez. Kernel Scheduling in Reconfigurable Computing. *In the Proceedings of Design and Test in Europe (DATE'99)*, Munich, Germany, March 1999.
- [4] N. Bagherzadeh, F. Kurdahi, H. Singh, G. Lu, M. Lee and E. Filho. MorphoSys: A Reconfigurable

Architecture for Multimedia Applications. *In the Proceedings of XI Brazilian Symposium on Integrated Circuit Design*, Rio De Janeiro, Oct. 1998.

[5] I. Damaj, H. Diab. Performance Analysis Of Some Geometrical Transformation Algorithms Using Reconfigurable Computing. *In the proceedings of the International Symposium on Innovation in*

*Information & Communication Technology ISIICT 2001*, Amman, Jordan September 26-27, 2001.

[6] I. Damaj, H. Diab. Performance Analysis Of Extended Vector-Scalar Operations Using Reconfigurable Computing. *In the Proceedings of the ACS International Conference of Computer Systems and Applications*, Beirut, Lebanon June 26-29 2001.

	C	D	E	F	G	H	I	J	K	L
3	cycle 0	Data bus		RC(0,0) = 0	Data bus		RC(0,1) = 0	Data bus		RC(0,2) = 0
4				RC(1,0) = 0			RC(1,1) = 0			RC(1,2) = 0
5				RC(2,0) = 0			RC(2,1) = 0			RC(2,2) = 0
6	cycle 1	x -2		x-2w2	x -2		x-2w1	x -2		x-2w0
7		x -1		x-1w2	x -1		x-1w1	x -1		x-1w0
8		x 0		x0w2	x 0		x0w1	x 0		x0w0
9	cycle 2	x -1		x-1w2	x -1		x-1w1+x-2w2	x -1		x-1w0+x-2w1
10		x 0		x0w2	x 0		x0w1+x-1w2	x 0		x0w0+x-1w1
11		x 1		x1w2	x 1		x1w1+x0w2	x 1		x1w0+x0w1
12	cycle 3	x 0		x0w2	x 0		x0w1+x-1w2	x 0		x0w0+x-1w1+x-2w2
13		x 1		x1w2	x 1		x1w1+x0w2	x 1		x1w0+x0w1+x-1w2
14		x 2		x2w2	x 2		x2w1+x1w2	x 2		x2w0+x1w1+x0w2
15	cycle 4	x 1		x1w2	x 1		x1w1+x0w2	x 1		x1w0+x0w1+x-1w2
16		x 2		x2w2	x 2		x2w1+x1w2	x 2		x2w0+x1w1+x0w2
17		x 3		x3w2	x 3		x3w1+x2w2	x 3		x3w0+x2w1+x1w2
18	cycle 5	x 2		x2w2	x 2		x2w1+x1w2	x 2		x2w0+x1w1+x0w2
19		x 3		x3w2	x 3		x3w1+x2w2	x 3		x3w0+x2w1+x1w2
20		x 4		x4w2	x 4		x4w1+x3w2	x 4		x4w0+x3w1+x2w2
21	cycle 6	x 3		x3w2	x 3		x3w1+x2w2	x 3		x3w0+x2w1+x1w2
22		x 4		x4w2	x 4		x4w1+x3w2	x 4		x4w0+x3w1+x2w2
23		x 5		x5w2	x 5		x5w1+x4w2	x 5		x5w0+x4w1+x3w2
24	cycle 7	x 4		x4w2	x 4		x4w1+x3w2	x 4		x4w0+x3w1+x2w2
25		x 5		x5w2	x 5		x5w1+x4w2	x 5		x5w0+x4w1+x3w2
26		x 6		x6w2	x 6		x6w1+x5w2	x 6		x6w0+x5w1+x4w2
27	cycle 8	x 5		x5w2	x 5		x5w1+x4w2	x 5		x5w0+x4w1+x3w2
28		x 6		x6w2	x 6		x6w1+x5w2	x 6		x6w0+x5w1+x4w2
29		x 7		x7w2	x 7		x7w1+x6w2	x 7		x7w0+x6w1+x5w2
30	cycle 9	x 6		x6w2	x 6		x6w1+x5w2	x 6		x6w0+x5w1+x4w2
31		x 7		x7w2	x 7		x7w1+x6w2	x 7		x7w0+x6w1+x5w2
32		x 8		x8w2	x 8		x8w1+x7w2	x 8		x8w0+x7w1+x6w2

Figure 4. 3-Tap FIR Filter Mapping

	C	D	E	F	G	H	I	J	K	L	M	N	O	P	S	V	Y	Z				
3	cycle 0	Data bus		RC(0,0) = 0	Data bus		RC(0,1) = 0	Data bus		RC(0,2) = 0	Data bus		RC(0,3) = 0	w2	w1	w0						
4				RC(1,0) = 0			RC(1,1) = 0			RC(1,2) = 0			RC(1,3) = 0	w2	w1	w0						
5				RC(2,0) = 0			RC(2,1) = 0			RC(2,2) = 0			RC(2,3) = 0	w2	w1	w0						
6				RC(3,0) = 0			RC(3,1) = 0			RC(3,2) = 0			RC(3,3) = 0	w2	w1	w0						
7	cycle 1	x -2	x-2w2	x -2	x-2w1	x -2	x-2w0	x -2			x -2			w2	w1	w0						
8		x -1	x-1w2	x -1	x-1w1	x -1	x-1w0	x -1						w2	w1	w0						
9		x 0	x0w2	x 0	x0w1	x 0	x0w0	x 0						w2	w1	w0						
10		x 1	x1w2	x 1	x1w1	x 1	x1w0	x 1						w2	w1	w0						
11	cycle 2	x 0	x0w2	x 0	x0w1+x-1w2	x 0	x0w0+x-1w1	x 0			x 0			w2	w1	w0		y0				
12		x 1	x1w2	x 1	x1w1+x0w2	x 1	x1w0+x0w1	x 1			x 1			w2	w1	w0		y1				
13		x 2	x2w2	x 2	x2w1+x1w2	x 2	x2w0+x1w1	x 2			x 2			w2	w1	w0						
14		x 3	x3w2	x 3	x2w1	x 3	x3w0	x 3			x 3			w2	w1	w0						
15	cycle 3	x 2	x2w2	x 2	x1w1+x0w2	x 2	x2w0+x1w1+x0w2	x 2			x 2			w2	w1	w0		y2				
16		x 3	x3w2	x 3	x2w1+x1w2	x 3	x3w0+x2w1+x1w2	x 3			x 3			w2	w1	w0		y3				
17		x 4	x4w2	x 4	x4w1+x3w2	x 4	x4w0+x2w1	x 4			x 4			w2	w1	w0						
18		x 5	x5w2	x 5	x5w1	x 5	x5w0	x 5			x 5			w2	w1	w0						
19	cycle 4	x 4	x4w2	x 4	x4w1+x3w2	x 4	x4w0+x2w1+x1w2	x 4			x 4			w2	w1	w0		y4				
20		x 5	x5w2	x 5	x5w1+x4w2	x 5	x5w0+x4w1+x3w2	x 5			x 5			w2	w1	w0		y5				
21		x 6	x6w2	x 6	x6w1+x5w2	x 6	x6w0+x5w1	x 6			x 6			w2	w1	w0						
22		x 7	x7w2	x 7	x7w1	x 7	x7w0	x 7			x 7			w2	w1	w0						
23	cycle 5	x 6	x6w2	x 6	x6w1+x5w2	x 6	x6w0+x5w1+x4w2	x 6			x 6			w2	w1	w0		y6				
24		x 7	x7w2	x 7	x7w1+x6w2	x 7	x7w0+x6w1+x5w2	x 7			x 7			w2	w1	w0		y7				
25		x 8	x8w2	x 8	x8w1+x7w2	x 8	x8w0+x7w1	x 8			x 8			w2	w1	w0						
26		x 9	x9w2	x 9	x9w1	x 9	x9w0	x 9			x 9			w2	w1	w0						

Figure 5. Mapping of the Improved Design.