

Homogeneous VLSI Structures for Finite Ring Computations¹

G.A. Jullien and W. C. Miller

VLSI Research Group, University of Windsor
Windsor, Ontario, Canada N9B 3P4 Tel. (519) 253-4232

Abstract

This paper discusses the basic concepts, and presents some preliminary results, of an approach towards the development of homogeneous arrays for massively parallel VLSI computation of Digital Signal Processing algorithms. The generation of such arrays is made possible by computing over finite rings, rather than weighted magnitude systems, such as binary, and by performing the computations at the bit-level. The architectures are systolic, or semi-systolic, in that local communication between bit-level cells is preserved, in the main, with pipelining operating at the bit level. The homogeneity of the arrays results from the use of a single generic cell structure that allows a pipelined bit-level inner product step to be computed using only twice the silicon area of a pipelined full-adder, and operating at a slightly higher throughput rate. The cell can be used for implementing a wide variety of DSP algorithms, including coding and decoding between conventional systems and the finite rings used for the computations. Since the use of an inner product step processor requires only twice the area of a binary adder, the complexity normally associated with multiplication (compared to that for addition) has now been removed, without compromising array structure and throughput. This means that algorithms can now be chosen for their VLSI structure rather than reduction of complexity of multiplication. The paper illustrates this concept by considering the homogeneous array implementation of a 2-D radix-2 DFT computational element with twiddle factor multipliers, a structure normally implemented with an irregular architecture.

Introduction

Since their introduction [Cap'81], [McC'82], bit-level systolic arrays have been a major factor in the implementation of very high speed, computationally intensive digital signal processing (DSP) algorithms. The first reported VLSI implementation of such arrays computed the digital correlation function, at a data rate of 20MHz [Cor'83], a perfect example of the use of such arrays. The bit-level systolic array has all the characteristics of a systolic array, but uses cells that operate with a single bit of data. This means that for an Nth-dimensional array, a bit-level realization would operate over N+1 dimensions.

Since the cell only operates with a single bit of the data, it is usually of a very small size has low critical path and so can operate at a very high clock rate. Usually only a small number of different cell types are used (often only one type), and the optimization of the set of cells provides a large payback when considering the implementation of large bit-level arrays [Iwa'85]. When large arrays are built on a single chip, the local communication feature can be used to advantage, in that long interconnects can be avoided for data communication. Clock distribution is the one element of the array construction that can be a problem, since the same set of signals is being distributed to the entire array. The problem is compounded in that all the signals ideally should arrive at each cell at the same time; i.e. no skew should be present between clock signals arriving at different arrays. This restriction can be lifted, if one limits the skew or arranges it to occur in a benign fashion [Hat'86]. For example, skew occurring in the opposite direction to the data flow will not introduce race conditions.

Bit-level cell structure can also be developed to allow the implementation of systolic arrays using finite ring/field arithmetic. Several such arrays running independently, and using relatively prime ring moduli, forms a residue number system (RNS). The RNS approach removes the connectivity across the dynamic range, and so the 2-D array reverts back to a linear systolic array. The RNS approach has potential advantages in minimizing the clock-skew problem, and making the array easier to test based on the much lower connectivity compared to equivalent dynamic range binary arrays. The most powerful advantage is

¹This research work was carried out under financial support from the Natural Sciences and Engineering Research Council of Canada.

the ability to compute fixed coefficient multiplication at no extra cost, and for massively parallel DSP systems, such as fixed coefficient FIR filters and DFTs, this potentially translates into large savings of silicon area.

We will first discuss the mathematical structure behind the bit-level cells, the VLSI implementation of a very efficient cell structure, and finally an example of a homogeneous array for a 2-D radix 2 butterfly.

Notation

We will define the ring (field) of computation by:

Base Ring(Field):

$R(m_k)$ or $GF(m_k) = \{S: \oplus_{m_k}, \otimes_{m_k}\}; S = \{0, 1, \dots, m_k-1\}$
 where the symbols \oplus_{m_k} and \otimes_{m_k} correspond to modulo m_k addition and multiplication, respectively.

We will define the ring over which the computation is finally mapped to as:

Direct Sum Ring:

$$R(M) = \{ \overline{S} : \overline{\oplus}, \overline{\otimes} \}; \overline{S} = \{0, 1, \dots, M-1\}; M = \prod_{k=1}^L m_k.$$

Where it is not obvious by context which ring the operation is being performed over, a subscript will be employed. Thus \oplus_m indicates addition over the ring with modulus m. Note that direct sum rings are not rings over which actual implementation of the digital signal processing algorithm takes place; they are isomorphic to the direct sum of the corresponding implementation rings. We have therefore used the symbols $\overline{\oplus}$ and $\overline{\otimes}$ rather than the expected symbols \oplus_M and \otimes_M . Results become available over these rings following the appropriate isomorphic mapping (e.g Chinese Remainder Theorem).

Residue Number System

A digit in the residue number system is represented by an L-tuple of residues [Sza'67]: $X = (x_0, x_1, \dots, x_{L-1})$
 where $x_i = (X) \text{Mod } m_i$ is the ith residue and m_i is the ith modulus. Closed computations (addition, multiplication) over the implementation rings map to the direct sum ring via the Chinese Remainder Theorem (CRT). We will write this succinctly as:

$$\overline{A} \oplus \overline{B} \Leftrightarrow \{a_0 \oplus b_0, a_1 \oplus b_1, \dots, a_L \oplus b_L\};$$

$$\overline{A} \otimes \overline{B} \Leftrightarrow \{a_0 \otimes b_0, a_1 \otimes b_1, \dots, a_L \otimes b_L\}$$

with $A, B \in R(M); a_k, b_k \in R(m_k)$ and $R(M) \sim \oplus R(m_k)$ (direct sum).

The final mapping is found from the CRT:
$$X = \sum_{k=1}^L \overline{M} \left\{ \hat{m}_k \otimes_M \left[x_k \otimes_{m_k} (\hat{m}_k)^{-1} \right] \right\}$$

with $\hat{m}_k = M / m_k, X \in R(M), x_k \in R(m_k)$ and $(\cdot)^{-1}$ the multiplicative inverse operator. We have also used the notation \sum_M to indicate summation over the ring R(M).

Finite Ring Bit-Level Cell (BIPSP_m)

The cell is based on a fixed multiplier Inner Product Step Processor (IPSP), and uses a small, fast, ROM as the computational element[Tah'87a]. For an IPSP operating over a finite ring, R(m), with a fixed multiplier (we will modify the notation to IPSP_m), the relationship is given in equation (1):

$$Y_{out} = Y_{in} \oplus_m [A_{in} \otimes_m X_{in}] \tag{1}$$

All the inputs and outputs are B bit ring elements, so that $Y, A, X \in R(m)$ with $B = \lceil \log_2 m \rceil$. The generic cell is based on a

bit-sliced implementation of the IPSP_m which we denote as BIPSP_m. The cell is shown in Fig 1, with fault detection circuitry[Tah'87b].

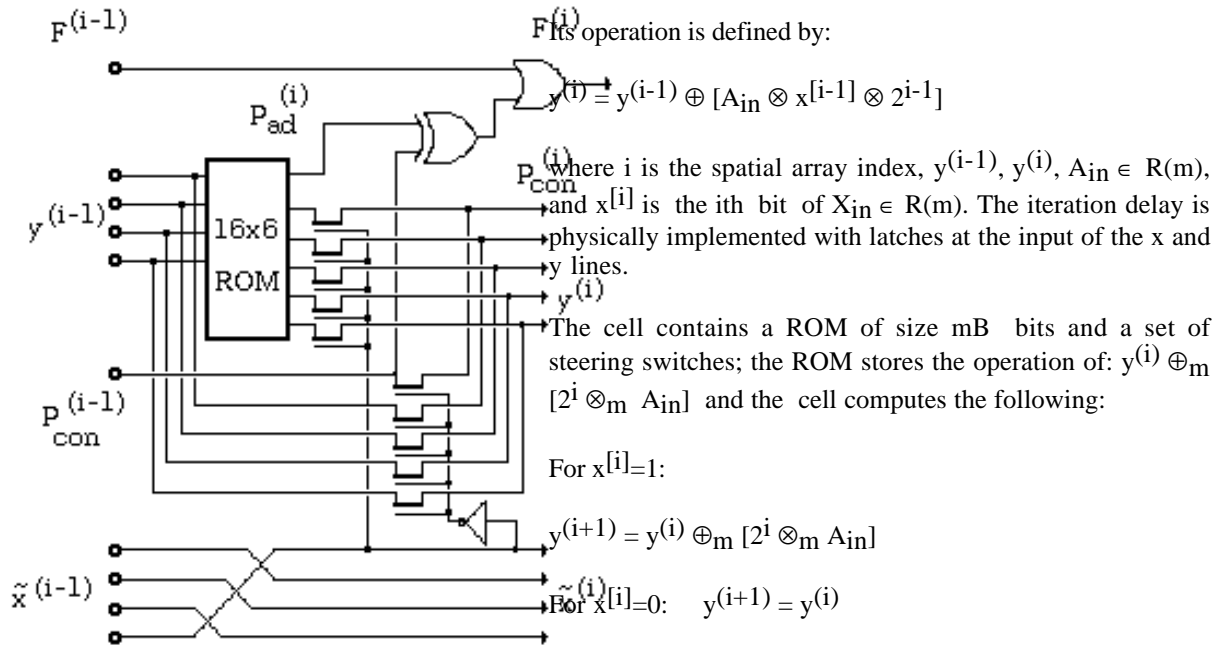


Fig 1 BIPSP_m with Fault detection

The fault detection is a simple parity check that occurs across the entire cell[Tah'87b], much the same way that an entire communication channel is tested by transmitter and receiver generated parity bits. This is possible because the cell is not used in isolation but is always connected in a linear array. In Fig 1, the X sequence has been modified, notationally, to \tilde{X} to indicate that a cyclic shift of one bit takes place. Note that the entire X sequence is available at the output and that it is in its original bit configuration after passing through B cells; i.e. $X^{(0)} = \tilde{X}^{(B-1)}$.

Cell design details

The block diagram of the finite ring cell implementation is shown in Figure 2[Jul'88]. The bit line, $\tilde{x}^{(i)}[0]$, of the $\tilde{X}^{(i)}$ input is used to control the flow of the $y^{(i)}$ input, either through the ROM or around it. Note that the implementation of the latches is distributed, rather than lumped at the input, as in Figure 1. This change was necessary for an efficient cell layout; they do not affect the functionality of the cell.

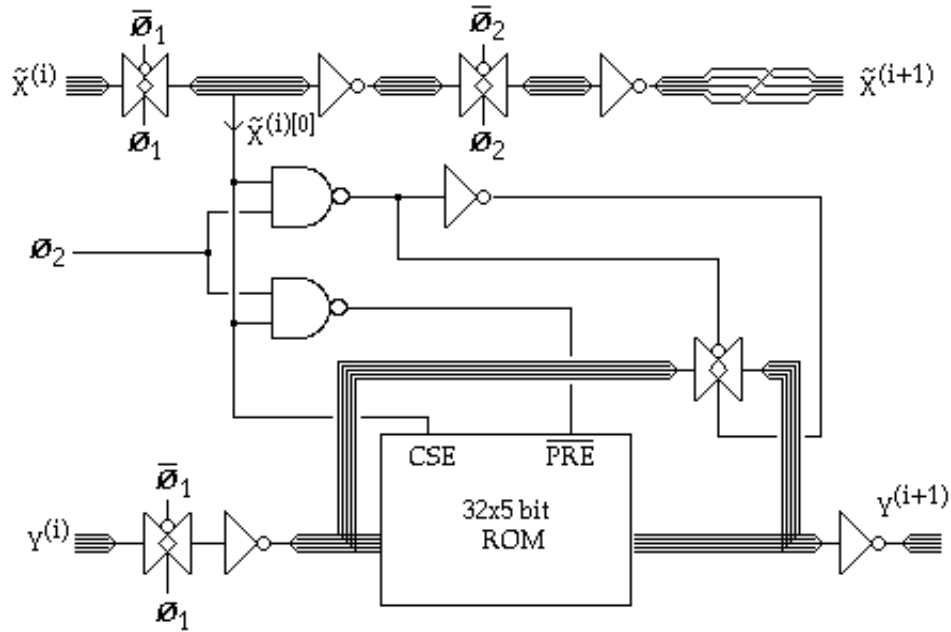


Figure 2 Block Diagram of the finite ring cell implementation

There are two main sections in this design; the latches and the ROM. Both are made up of a mixture of dynamic and static logic, as with the binary cell. The selection of logic implementation is based on minimizing the (Area.Period) product. All latches are controlled by two phase, non-overlapping clocks. By placing the latches controlled by ϕ_1 before the ROM, some duplication of devices is eliminated by using the inverters on the y input lines for both the latch function and decoder complement function.

The ROM is decomposed into four sections: the row decoder, column decoder, storage array and the pull up drivers. The ROM is configured as 8 rows by 4 columns and allows ring moduli of up to 32 to be implemented.

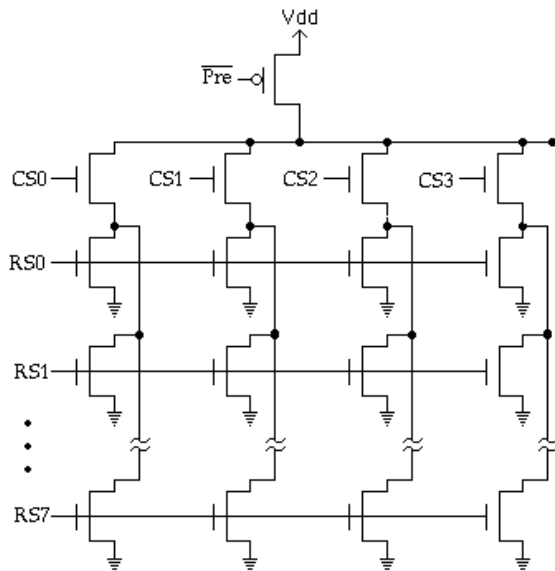
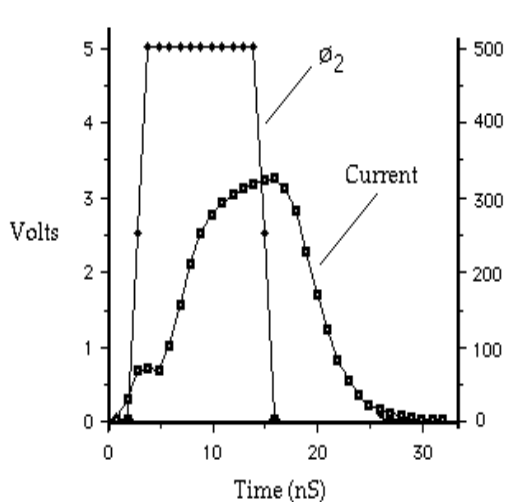


Figure 3 ROM storage details

Figure 3 shows the ROM storage circuitry. The ROM is dynamic, in that the parasitic capacitance associated with the column select transistors is pre-charged, and the ROM output is subsequently evaluated by the row select circuitry. During pre-charge, this transistor charges the array of nMOS transistors. If the column and row select lines choose a location where the row transistor is missing the charge is held in the parasitic capacitance of this bit line, if the transistor is present, however, it will pull the bit line down to ground. The output, $y^{(i+1)}$, is buffered by an inverter so a one is programmed by including the transistor, and a zero by excluding the transistor.

In order to increase the throughput rate of the cell, the time period during which the storage circuitry pre-charges and the row and column decoders evaluate is overlapped.



The current spike, shown in Figure 4, is measured at 330 μA peak, and is not significant when considering the total cell current requirement of over 3mA peak current. The current spike associated with charging the bit lines that are not evaluated is just over 60 μA . This represents the current spike that would occur if the evaluation and pre-charge intervals were non-overlapping. Since there are 5 bit lines, the total line capacitance charge current is 300 μA versus a combined capacitance charge and select line current spike of 570 μA . This represents an excellent trade-off between increased speed and increased current.

Figure 4 Current spike during overlap

It should be pointed out that steps can be taken in a chip-level design to stagger the time periods at which blocks of cells produce this current spike, in order to reduce the current spike at the chip level. Implied here is a controlled form of clock-skew. The finite ring cell allows race-free skew to be applied against the direction of data flow. i.e. along the linear connection of cells. All bit-level systolic cells will exhibit periods of relatively high current flow, and so the 'smearing' of the aggregate of these current spikes is good practice for all bit-level designs.

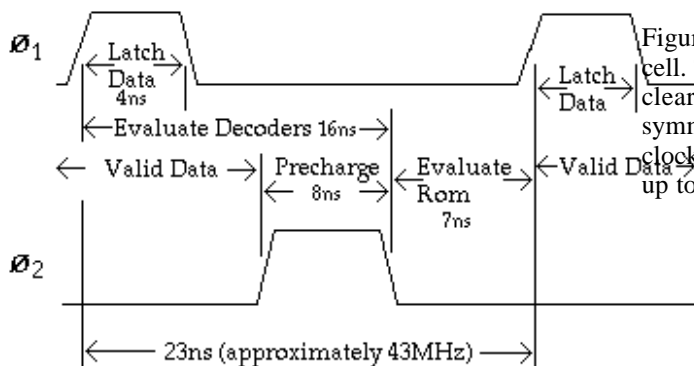


Figure 5 shows the complete pipeline cycle for the cell. The overlap between evaluate and pre-charge is clearly seen. Note that the overlap also allows a symmetrical sequencing of the two non-overlapping clock pulses. The simulated cycle allows pipelining up to 43MHz.

Figure 5 Timing diagram for a complete finite ring cell period

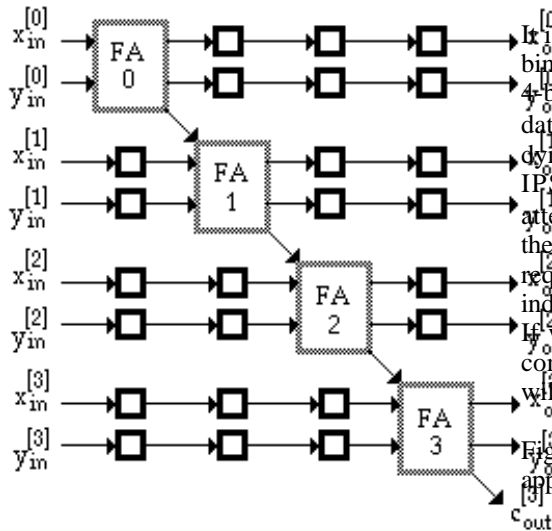
Comparison with binary cell

In a recent study [Jul'88] we compared the use of this cell versus the use of an efficient binary cell, in a FIR filter design with fixed coefficient taps. The fabrication technology is a 3 μ double metal CMOS process that is available to Canadian universities through the Canadian Microelectronics Corporation (CMC). The two cells are shown in Figure 6 at the same scale.

The binary cell is essentially a multiplexed logic gated full adder with pipeline latches. The finite ring cell is 5-bits with no fault detection. Both cells include a 'sliding' latch for implementing FIR filter structures. The area difference is about 2:1 in favour of the binary cell; the critical path of the finite ring cell is, surprisingly enough, slightly faster than that of the binary cell even though there are 355 transistors in the finite ring cell and only 86 in the binary cell.

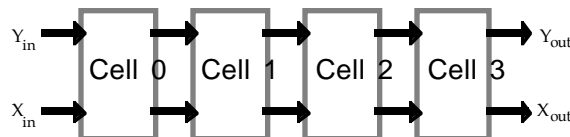
Fig 6 Binary cell versus finite ring cell

A direct comparison was made in applying both cells to a FIR filter problem. We took the case of a fixed coefficient FIR filter, with binary input/output. Coefficient quantization was taken as 8-bits, with a data quantization also of 8-bits. The length of the filter was 64 taps with a computational range of 20 bits, and output scaled to 10 bits. For the binary cell, the area required is 2966.4mm^2 , (30 chips using the 3μ target technology). The area required for the finite ring approach is 857mm^2 (9 chips). The throughput rate is slightly higher for the finite ring array without taking clock skew into account.



It is important to recognize the structural differences between the binary and finite ring arrays. This is emphasized in figure 7 for a 4-bit binary adder and a 4-bit IPSP_m. The binary adder requires data skewing and de-skewing and the cells are connected in the dynamic range dimension. To perform the same function as the IPSP_m we would require an array of 4x4 full adders with attendant 2-D clock skew problems. Without even considering the structural differences (there is no data skewing and deskewing required in the finite ring approach and the architecture reverts to independent linear systolic arrays) there is a 3.5:1 area advantage. If we look at larger filter lengths, and consider the filtering of complex sequences (with QRNS mapping) the area advantage will grow, to perhaps an order of magnitude.

Figure 7 Structural comparison between binary and finite ring approaches



Homogeneous VLSI Arrays (an example)

We do not yet have a general theory for producing homogeneous arrays from DSP algorithms. We do, however, have examples of the application of arrays of IPSP_m cells to certain standard structures. Clearly DSP algorithms such as FIR filters will produce very regular arrays. Instead, let us consider the example of an irregular structure, a 2-D radix 2 DFT computational element (CE) with twiddle factor multipliers. This is normally built with fixed site multiplications; we will

consider a DIT CE with pre-multipliers. In order to make maximum use out of the ability to compute over finite rings, the computational element will be based on small modulus quadratic residue rings. This is a departure from the current direction in quadratic residue computation in which a large single ring is used, with a Fermat Prime modulus and implemented with modified binary arithmetic[Tay'85]. There are many advantages available to multiple small ring systems that are missing in single large ring structures; generic systolic cell arrays are natural, and reducing bit-level interconnections has many advantages in terms of clockskew minimization, testing and fault detection.

The quadratic residue ring

We will introduce quadratic residue ring theory with an addition to the notation introduced earlier.

Quadratic Residue Ring:

$QR(m_k) = \{S: \oplus, \otimes\}; S = \{A^\circ, A^*\}$ with $A^\circ, A^* \in R(m_k)$ and $A^\circ = a+jb, A^* = a-jb; j = -1; a, b, j \in R(m_k), m_k =$

$p_i^{e_i}, p_i = 4k+1, a$ prime. A° will be referred to as the *normal* component of element \mathbf{A} and A^* as the *conjugate* component of element \mathbf{A} . The multiplication and addition operators both compute component-wise.

Direct Sum Quadratic Ring:

$QR(M) = \{ S : \overline{\oplus}, \overline{\otimes} \}; S = \{(A_1^\circ, A_1^*), \dots, (A_L^\circ, A_L^*)\}; M = \prod_{k=1}^L m_k.$

Note that the actual computations take place over the base fields associated with the normal and conjugate components of the quadratic residue elements.

2-D DFT computational element array

For the 2-D system we assume a true 2-D computational element rather than the often used 1-D row-column computation. In order to generate a suitable radix-2 computational array we start with the basic formulation of a 2x2 DFT, computed over a direct sum of quadratic residue rings, as shown in equation (2).

$$\mathbf{X}_{k,l} = \bigoplus_{n=0}^1 \bigoplus_{m=0}^1 \{ \mathbf{x}_{n,m} \otimes \mathbf{j}^{2(nk+ml)} \} \quad (2)$$

with $\mathbf{j} = \sqrt{-1} \in QR(M), k,l \in [0, 1)$, and $\mathbf{x}, \mathbf{X} \in QR(M)$. Using the QRNS decomposition we isolate the computations into 2L independent but structurally identical processors. The isolation of computations is important because we use QRNS in this approach for its structural, rather than arithmetic complexity reduction properties. The operations within the QRNS ring structure are base ring computations, even though the result will map to the extension ring. When embedded in an N point 2-D FFT structure, equation (2) is augmented with complex twiddle factor multipliers, $\{\alpha\}$. We will assume a pre-multiplication form for the butterfly. The operation of one of these processors (say the *normal* element) is shown in equation (3):

$$X_{k,l}^\circ = \bigoplus_{n=0}^1 \bigoplus_{m=0}^1 \{ x_{n,m}^\circ \otimes \mathbf{j}^{2(nk+ml)} \otimes \alpha_{n,m}^\circ \} \quad (3)$$

We take note of the fact that $\alpha_{0,0}^\circ = 1$, in any position of the NxN array. Equation (3) can be computed recursively (and a systolic structure obtained) using intermediate variables A, B as shown in equation (4).

$$X_{k,l}^\circ = A^{[2k+1]} \oplus (\alpha_{1,0}^\circ \otimes \mathbf{j}^{2k} \otimes B^{[2k+1]}) \quad (4)$$

where initial values are: $A^{[0]} = x_{0,0}^\circ \oplus x_{0,1}^\circ, B^{[0]} = x_{1,0}^\circ \oplus x_{1,1}^\circ$
with the recursive formulations given in equation (5):

$$A^{[i+1]} = A^{[i]} \oplus \{x_{0,1}^{\circ} \otimes \alpha_{0,1}^{\circ} \otimes (j^{2(i+1)} \oplus [-j^{2i}])\} \quad (5)$$

$$B^{[i+1]} = B^{[i]} \oplus \{x_{1,1}^{\circ} \otimes \alpha_{1,1}^{\circ} \otimes (\alpha_{1,0}^{\circ})^{-1} \otimes (j^{2(i+1)} \oplus [-j^{2i}])\}$$

to compute $A^{[i]}$ and $B^{[i]}$ for $i=1,2,3$.

Note that we have defined an inverse, $(\alpha_{1,0}^{\circ})^{-1}$. Although we cannot rely on the QRNS mapped inverse for $\alpha_{1,0}$, the inverse exists for each component of $\alpha_{1,0}$ providing we compute over a base field. For the QRNS, the allowable moduli, between 5 and 6 bits, with $4k+1$ prime factors, are 61, 57, 53, 41, 37, 29, 25, 17. With the exception of 25, they are all primes.

By observing, and removing duplication of IPSP functions, we can make some simplifications in the structure. The semi-systolic (some broadcasting required) realization of the 2-D CE is shown in Figure 8. The fact that multiplication does not involve extra hardware allows us to *spread out* the multipliers over the array. Contrast this to conventional structures that require fixed multiplication sites.

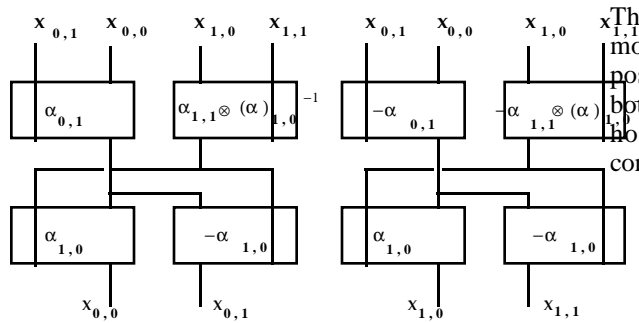


Figure 8 Minimum IPSP realization of the CE

The CE consists of two arrays, each with four IPSP modules. Note that there is some interchange of data position and some broadcasting between the top and bottom two pairs of arrays. The basic pipeline structure, however, maintains two linear arrays running concurrently.

VLSI array design details

The example system combines the butterfly structure for both the normal and conjugate elements for one modulus, i.e. twice the circuitry indicated in figure 8.

Figure 9 shows the VLSI block diagram layout for the array (I/O drivers and pads not shown). The array measures $3360\mu \times 4200\mu$ and consists of approximately 28,000 transistors. The total number of BIPSP_m cells is 80 with 8 inputs and 8 outputs at 5-bits each (4 pairs of quadratic elements). Since the realization, presented in this paper, is carried out using a double metal process, RC clock skew in the array is not a major problem; however, capacitive loading on the global clock lines due to the 80 cells can cause considerable skewing problems.

Figure 9 VLSI layout of for a single quadratic ring

For this reason, each cell contains a local clock buffer with a minimum input capacitive loading on the global clock lines. This is a critical issue for pipelined systolic array implementations.

Conclusions

This paper has discussed the concept of producing homogeneous arrays for high performance VLSI implementations of DSP algorithms. The homogeneity is obtained from very regular arrangements of a single, mask programmable, generic cell. The cell computes the inner product step processor over finite rings or fields, and allows linear systolic arrays to be used at the bit level rather than the 2-D arrays associated with binary implementations. A 5-bit cell is only twice the area of a full adder cell, but allows a complete fixed multiplier inner product step to be computed using 5 cells. This is the same number of cells that the binary implementation requires for a single addition. Results of a comparative study between the finite ring cell and a binary cell are presented and show that the finite ring cell is far superior in terms of area efficiency for typical DSP applications. The cell has a throughput rate equal to that of the binary cell, provides much better array structures and can include low-overhead fault detection circuitry. An example of a radix-2, 2-D computational element (butterfly) for a DFT is used to illustrate the homogeneity that can result from the application of this cell to DSP computations. This example is chosen to emphasize the difference between the typical irregular structure, using conventional implementation, and the homogeneous structure resulting from application the finite ring cell computing over quadratic residue rings. Examples of VLSI implementations in a 3μ CMOS process, made available through the Canadian Microelectronics Corporation, are used to illustrate the results.

References

- [Cap'81] Capello, P.R. and Steiglitz, K. (1981). "Digital Signal Processing Applications of Systolic Algorithms", in VLSI Systems and Computations, H.T. Kung, R. Sproull and G. Steele eds., Computer Science Press, pp.245-254.
- [Cor'83] Corry, A. and Patel, K. (1983) "Architecture of a CMOS Correlator", Proceedings of the Int. Conf. on Circuits and Systems, pp. 522-525.
- [Hat'86] Hatamian, M. and Cash, G.L. (1986) "High Speed Signal Processing, Pipelining, and Signal Processing", Proc. ICASSP, April, pp.1173-1176.
- [Iwa'85] Iwano, K. and Steiglitz, K. (1985) "Some Experiments in Leaf-cell Optimization", in VLSI Signal Processing, IEEE Press, pp. 387-395.
- [Jul'88] G.A.Jullien, P.D.Bird, J.T.Carr, M.Taheri, W.C.Miller, "An Efficient Bit-Level Systolic Cell Design for Finite Ring Digital Signal Processing Applications", Journal of VLSI Signal Processing (accepted), August 1988.
- [McC'82] McCanny, J.V. and McWhirter, J.G. (1982) "Implementation of Signal Processing Functions using 1-bit Systolic Arrays", Electronics Letters, Vol. 18, No. 6, March, pp.241.
- [Sza'67] Szabo, N.S. and Tanaka, R.I. (1967). " Residue Arithmetic and its Applications to Computer Technology", McGraw-Hill, New York.
- [Tah'87a] M. Taheri, G.A. Jullien, and W.C. Miller, " Systolic ROM Arrays For Implementing RNS FIR Filters", IEEE Intr. Conf. on Acoustics, Speech, and Signal Processing, ICASSP'87, April 1987,pp. 771-774.
- [Tah'87b] M. Taheri, G.A. Jullien, and W.C. Miller, (1987) " Fault Detection in RNS Systolic Arrays," IEE Electronics Letters, Vol. 23, No. 4, Feb. , pp. 165-166
- [Tay'85] F.J. Taylor, G. Papadourakis, A. Skavantzios and A. Stouraitis, " A Radix-4 FFT Using Complex RNS Arithmetic", IEEE Trans. Computers, Vol. C-34, June 1985,pp. 573-576.