

TOWARDS AN FIR FILTER TISSUE

Peter R. Cappello[†]

Computer Science Department
University of California
Santa Barbara, CA 93106

INTRODUCTION

This paper presents a VLSI chip architecture for finite impulse response (FIR) filtering. The chip is intended for applications where high throughput is more important than short latency. It is well known that FIR filters, in principle, can be implemented at arbitrarily high sample rates [1,2] since multiple output samples can be computed in parallel. Attention is paid in this paper to doing so with VLSI technology. An important property of any system intended for VLSI implementation is structural regularity. Cellular automata [3] have an iterative form of structural regularity. They are especially well-suited to VLSI technology; iterative regularity simplifies design and nearest-neighbor communication simplifies operation. The emergence of VLSI technology thus is rekindling interest in cellular automata. Systolic algorithms / architectures, a kind of cellular automata, have received much recent attention. Systolic array implementations of FIR filters are presented in [4,5,6,7,8].

Taking these benefits one step further, we may use bit-level cells -- cells that operate on a bit (or a few bits) rather than word-level cells. This approach may be used to advantage as a way of implementing word-level cells as is done, for example, with the Iterative Array Multipliers described in [9]. Bit-level cells may also be used to organize systems as is suggested by the classic Sequential Iterative Systems [10]. Bit-level systolic algorithms have been reported for Convolution [11,12,13,14,15], FIR Filtering [11,14,16], HR Filtering [14], Inner Product [17], Linear Transformation (including Discrete Fourier and Walsh Transformations) [11,18,19], Matrix Product [20], and String pattern-matching [21].

All of these bit-level cellular algorithms have very similar underlying architectures - arrays of one bit full adders (string pattern-matching excepted) with control circuitry that differs somewhat in each algorithm.

An arithmetic tissue for FIR filtering

"tissue: ... is an aggregate of cells usually of a particular kind or kinds together with their intercellular substance that form one of the structural materials out of which the body of a plant or an animal is built up." [22]

Bit-level cellular algorithms suggest the development of "computational tissues." A tissue for FIR filtering is proposed in this paper; FIR filters are widely used in practice, and the tissue proposed can be used to implement most of the bit-level systolic algorithms mentioned above. Thus, although each cell has simple logic and switching capabilities, the collection of cells - the tissue - is a versatile computational material. This tissue may be fashioned into FIR filter "organs" for use in specialized, high-throughput digital signal processing systems. With this architectural and performance context in mind, we proceed to identify our tissue's specific anatomic requirements.

THE TISSUE'S ANATOMY

The bit-level systolic algorithms referenced in the Introduction all use arrays of cells that can act as 1-bit full adders. Each algorithm, however, requires somewhat more of its cells. These additional capabilities are (perhaps surprisingly) modest. The proposed tissue's anatomic features derive from the requirements of a highly parallel bit-level systolic algorithm for FIR filtering. We thus focus on this algorithm's implementation as a means of coherently introducing the FIR filter tissue's anatomy. Our goal is to incorporate all the requirements of this algorithm into a homogeneous graph of bit-level cells that are functionally identical.

The word-level algorithm presented produces K outputs in parallel, where K is the order of the FIR filter. The structure of the algorithm resembles very closely that of a systolic linear transform. The algorithm is illustrated for a 4-tap FIR filter. It executes on a 4×4 orthogonal mesh of processors, depicted in Fig. 1. Input and output are in blocks of 4, and are skewed in time. Inputs move southeastward at the rate of one cell/cycle. Outputs move eastward at the same rate. On each cycle, the cells do an inner product step: $y' \leftarrow y + a \cdot x$. Note, however, that the inputs are delayed an extra cycle, as indicated by the horizontally aligned delay elements. Fig. 1 presents the filter computation at cycle 3 (assuming that the computation starts on cycle 0). Following that figure, one can see that y_3 will be available on cycle 4, y_7 and y_2 on cycle 5, y_{11} , y_6 , y_1 on cycle 6, y_{15} , y_{10} , y_5 , y_0 on cycle 7, and so on. This design is derived in the full report.

We now turn our attention to a bit-level algorithm that is compatible with these word-level cells. Fig. 2 depicts a pipelined array, adapted from [18], for performing multiplication of x by a constant, a . The values of x , a , and their product, s , are represented in two's complement. Each cell does what may be considered the bit-level analogue of an inner-product step:

$$\begin{aligned} s' &\leftarrow (a \cdot x) \oplus s \oplus c \\ c' &\leftarrow (a \cdot x \cdot s) + (a \cdot x \cdot c) + (s \cdot c) \end{aligned}$$

[†]This work was supported partly by the National Science Foundation under Grant ECS-8307955 and the Office of Naval Research under contract N00014-81-K-0664.

Note, however, in the array of Fig. 2 that:

1. the black cells on the upper left hand boundary use the complement, \bar{a} , rather than a in their full adders, and
2. the cell that multiplies \bar{a}_0 by x_n has a carry-in of x_n , not 0.

The initial sum bits of such an array are set to zero. A particular integer product is formed as it flows from top to bottom through the array. See [18] for a derivation of this bit-level cellular multiplier.

The bit-level array is coupled to the word-level array by making the following observation. If y is used as the sum-bit (s_i) initial values, then $y + a \cdot x$ are the sum-bit final values (s_o). Finally, Fig. 3 depicts the bit-level FIR filter algorithm. That figure illustrates a 4-tap FIR filter, where each scalar is represented with 3 bits; output scalars are 8 bits: $3 \times 2 + \lceil \log_2 4 \rceil$. Note that there is one extra delay element along each input bit path, as required by the word-level algorithm.

This cellular algorithm forms the basis of the anatomic requirements for the FIR filter tissue; we proceed to enumerate these implied requirements.

Topological requirements

Both the word-level and the bit-level algorithms require the orthogonal mesh of cells. It thus is chosen as our tissue's intercellular structure.

Cell requirements

Cell requirements are presented in two steps: The inner cell requirements and the outer cell requirements. The reader is cautioned that this inner - outer dichotomy is an artifact of presentation; the actual cell is implemented as a (PLA-based) finite state machine. Those data associated with the inner cell have a subscript of 'i' for "inner;" those with the outer cell, an 'o'.

The inner cell is simply that depicted in Fig. 2. It has four inputs and four outputs. It generates $a_i \cdot x_i$ and uses this bit product as one of the inputs in the 1-bit full adder. Each output is defined as a boolean function of the inputs:

$$\begin{aligned} a_i &\leftarrow a_i. \\ s_i &\leftarrow (a_i \cdot x_i) \oplus s_i \oplus c_i. \\ x_i &\leftarrow x_i. \\ c_i &\leftarrow (a_i \cdot x_i \cdot s_i) + (a_i \cdot x_i \cdot c_i) + (s_i \cdot c_i). \end{aligned}$$

The outer cell contains (what are in affect) the switching and memory capabilities that are used in the bit-level algorithms of interest. Fig. 4 gives its schematic. For two's complement integer product, the carry-in bit of the inner cell may take on the value of 1) c_o ; 2) zero; 3) x_o .

- $c_i \leftarrow \text{select}\{c_o, 0, x_o\}$.

When two's complement is used, certain cells must generate $\bar{a}_i \cdot x_i$ rather than $a_i \cdot x_i$. Since this can be determined when the cell's state is established, the cell can be programmed with \bar{a} rather than a . No switch is needed in the cell for this.

In the word-level algorithm for FIR filtering, each input data stream is delayed one extra cycle. This optional delay is used for that purpose.

- $x_o \leftarrow \text{select}\{x_i, x_i \cdot z^{-1}\}$.

Treating these switches independently, we have 24 distinct states: 6 different switch settings and 2 bits of memory: 5 state bits suffice[†]. The *SET* bit, a control input, indicates if the inputs are to be used to set the state (*SET*), or are to be used as data values (*SET*).

THE TISSUE'S FUNCTION

Once a cell's state has been set, its behavior is determined. Underlying the programming of cells is the assumption that they will need to be programmed (i.e., configured for a particular FIR filter) infrequently. The procedure for doing so is slow; it avoids broadcasting.

Upon receiving a *set* signal, a cell will go into *set* mode. In set mode, the cell sets its state according to its other input's values. In order to keep this procedure simple and port (pin) efficient, two decisions have been made:

1. The tissue state is not programmed incrementally, every cell in the tissue is programmed (though not identically) when the tissue state is programmed.
2. Cell states are not changed incrementally, the entire cell state is set (reset) every time that set mode is entered.

This approach obviates the need for cell and state bit address information.

Although the tissue is designed specifically to support FIR filtering, some other cell and tissue capabilities are noted.

1. The cells, which are essentially arithmetic, can be made to perform a variety of boolean functions, depending on their state, such as *and* (\wedge), *or* (\vee), *not* (\neg), *exclusive-or* (\oplus), *implication* (\rightarrow) and *exclusive-nor*, also known as *if and only if* (\Leftrightarrow).
2. Although each cell is identical, the actual function a cell performs depends on its state. The tissue's cells do not all have the same state while executing any particular algorithm. In this sense, the tissue actually has the architecture of a homogeneously structured MIMD machine.
3. In addition to two's complement arithmetic, the tissue supports unsigned, sign-magnitude, and one's complement arithmetic.
4. Tissue can be shared: Virtual partitions can be created by appropriately setting cell states.
5. Because the tissue is extensible, it need never be the bottleneck in a high-throughput computer system.

PROJECT STATUS AND PLANS

A prototype cell, based on a 1-metal layer NMOS process that is supported by both MOSIS [23] and ALLENDE [24], has been successfully simulated using the Berkeley tools: Crystal and Esim.

There is a working tissue simulator. It has been used to test bit-level cellular algorithms.

Once established, a tissue "program," (i.e., the matrix of desired cell states) must be communicated to the individual cells. The tissue does not function as a simple SIMD machine. The loading sequence, as described in the previous section, has been worked out by hand on a case by case basis, using only low-level software aids. A tissue program loader will be written in order to automate this tedious, error-prone procedure.

In fact the minimal state space is smaller: not all switch settings are meaningful.

Program compilation is the process of establishing those cell states that are needed to perform a certain algorithm. To date, small examples have been worked out by hand. A high-level language and compiler will be developed for programming the tissue. As a preliminary step, we intend to write algorithm-based expert-compilers. For example, we will write an FIR filter compiler. It is envisioned that, with this software, an FIR filter applications engineer will be prompted for the FIR filter parameters. For a given bit-parallel, word-parallel FIR filter those parameters include:

1. word-size: input, output, and internal,
2. number of filter taps,
3. tap values, and
4. arithmetic: two's complement, one's complement, etc.

These parameters determine each cell's state.

CONCLUSIONS

Given a tissue of suitable size, one may program the tissue's cells so that it implements any FIR filter. The parameters of such a tissue program are: the number of taps, the tap values, the word sizes (input, output, and internal), and the type of arithmetic (two's complement, one's complement, or sign-magnitude). The FIR tissue thus is versatile.

With respect to performance, the tissue supports high-throughput applications; the filter implementation is blocked and pipelined, with a cycle time that depends on neither the word size nor the number of taps.

The FIR filter tissue has many desirable anatomic features. It is homogeneous, that is, the tissue is an array of structurally identical cells. Each cell is simple - roughly on the order of a latched one-bit full adder. The intercellular structure - the orthogonal mesh - is also simple. It is extensible, that is, the extent of the tissue can be increased (e.g., by adding a new row of cells) without redesigning any of the parts. Homogeneous simple cells, cellular communication, and extensibility all contribute to making this tissue especially well-suited to VLSI implementation.

REFERENCES

- 1 Stockham, T. G., "High speed convolution and correlation," *AFIPS Conf. Proc.*, vol. 28. Washington, DC: Spartan, 1966.
- 2 Gold, B. and C. M. Radar, *Digital Signal Processing of Signals*. New York: McGraw-Hill, 1969.
- 3 Burks, A. W., Editor, *Essays on Cellular Automata*. Urbana, IL: Univ. of Illinois Press, 1970.
- 4 Kung, H. T., "Why Systolic Architectures." *Computer*, 15 (1982), 1, pp. 37-46.
- 5 Kung, H. T. and C. E. Leiserson, "Algorithms for VLSI Processor Arrays," in Mead and Conway, *Intro. to VLSI Systems*, Wesley Publishing Co., Reading, MA, Oct. 1980.
- 6 Ahmed, H. M., J. Delosme, and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *IEEE Computer* 15(1) Jan. 1982.
- 7 Rao, S. K. and T. Kailath, "VLSI and the digital filtering problem," Information Systems Lab., Stanford Univ. 1984.
- 8 Lu, H-H, Lee, E. A., and D. G. Messerschmitt, "Fast recursive filtering with slow processing elements," Dept. of Electrical Engineering & Computer Sciences, Univ. of California, Berkeley, 1984.
- 9 Hwang, K. "Computer Arithmetic." New York: John Wiley & Sons, Inc., 1979.
- 10 Hennie, F. C. "Finite-State Models for Logical Machines." New York: John Wiley & Sons, Inc., 1968.
- 11 Cappello, P. R. and K. Steiglitz, "A Note on Free Accumulation in VLSI Filter Architectures." to appear (April or May) in *IEEE Trans. on Circuits and Systems*, .R (1985).
- 12 Cappello, P. R. and K. Steiglitz, "Completely Pipelined Architectures for Digital Signal Processing." *IEEE Trans. on Acoustic, Speech, and Signal Processing*. 31 (1983), 4, pp. 1016-1023.
- 13 Cappello, P. R. and K. Steiglitz, "Bit-Level Fixed-Flow Architectures for Signal Processing." In Proceedings of the *IEEE Int. Conf. on Circuits and Computers*, Sept. 1982, New York.
- 14 Cappello, P. R. and K. Steiglitz, "Digital Signal Processing Applications of Systolic Algorithms." In H. T. Kung, Bob Sproul, and Guy Steele, (eds.), *VLSI Systems & Computations*, Rockville, Maryland: Computer Science Press, 1981.
- 15 Evans, R.A., D. Wood, J.V. McCanny, J.B. McWhirter, and A.P.H. McCabe, "A CMOS Implementation of a Systolic Multi-bit Convolver Chip." In Proceedings of *VLSI '83*, Aug. 1983, Trondheim, Norway.
- 16 Denyer, P. B. and D.J. Myers, "Carry-Save Arrays for VLSI Signal Processing." In Proceedings of *VLSI '81*, Aug. 1981, Edinburgh.
- 17 Urquhart, R. B., "VLSI Architectures for the Linear Discriminant Function Classifier." In Proceedings of the *IEEE Workshop on CAPAIDM*, Pasadena, CA, 1982.
- 18 McCanny, J.V., J.G. McWhirter, J.B.G. Roberts, D.J. Day, T.L. Thorp, "Bit Level Systolic Arrays." In Proceedings of the *15th Asilomar Conf. on Circuits, Systems, & Computers*, Monterey, CA, Nov. 1981.
- 19 McCanny, J.V., and J.G. McWhirter, "Bit-level Systolic Array Circuit for Matrix-Vector Multiplication." *IEE Proceedings*, 130 (1983), Pt. G, 4, pp. 125-130.
- 20 McCanny, J.V., K.W. Wood, J.G. McWhirter, and C.J. Oliver, "The relationship between word and bit level systolic arrays as applied to matrix X matrix multiplication." In Proceedings of the *SPIE Tech. Symp., Real Time Signal Processing VI*, San Diego, Aug. 1983.
- 21 Foster, M. J. and H. T. Kung, "Toward a Theory of Systolic Algorithms for VLSI." (Abstract), In Proceedings of the conference on *Advanced Research in Integrated Circuits*, Cambridge, MA, Jan. 1980.
- 22 *Webster's Third New Int. Dictionary*, Chicago, Ill: Encyclopedia Britannica, 1971.
- 23 "The MOSIS System." Information Sciences Institute, ISI/TM-84-128.
- 24 Mata, J. M., "The ALLENDE Layout System User's Manual." Princeton University, Dept. of Electrical Engineering and Computer Science, VLSI Memo #9, June 1984.

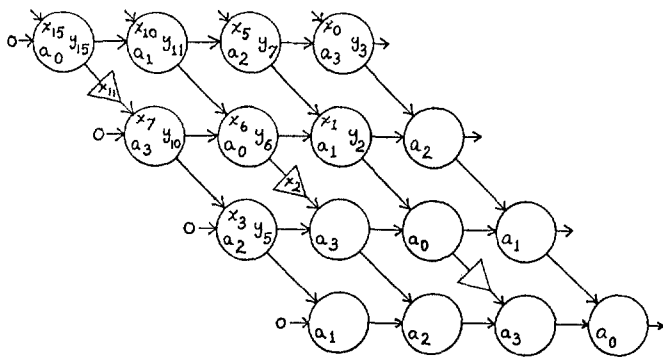


Fig. 1 The highly parallel FIR filter algorithm executes on an orthogonal mesh of processing elements.

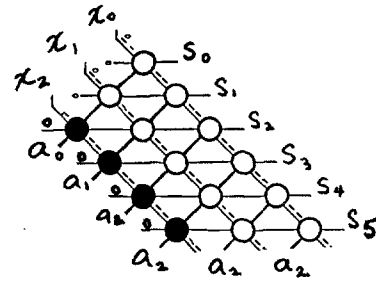


Fig. 2 A pipelined two's complement array multiplier, adapted from [18]. The operands, x and a , are represented in 3-bit two's complement; the output, s , is represented in 6-bit two's complement.

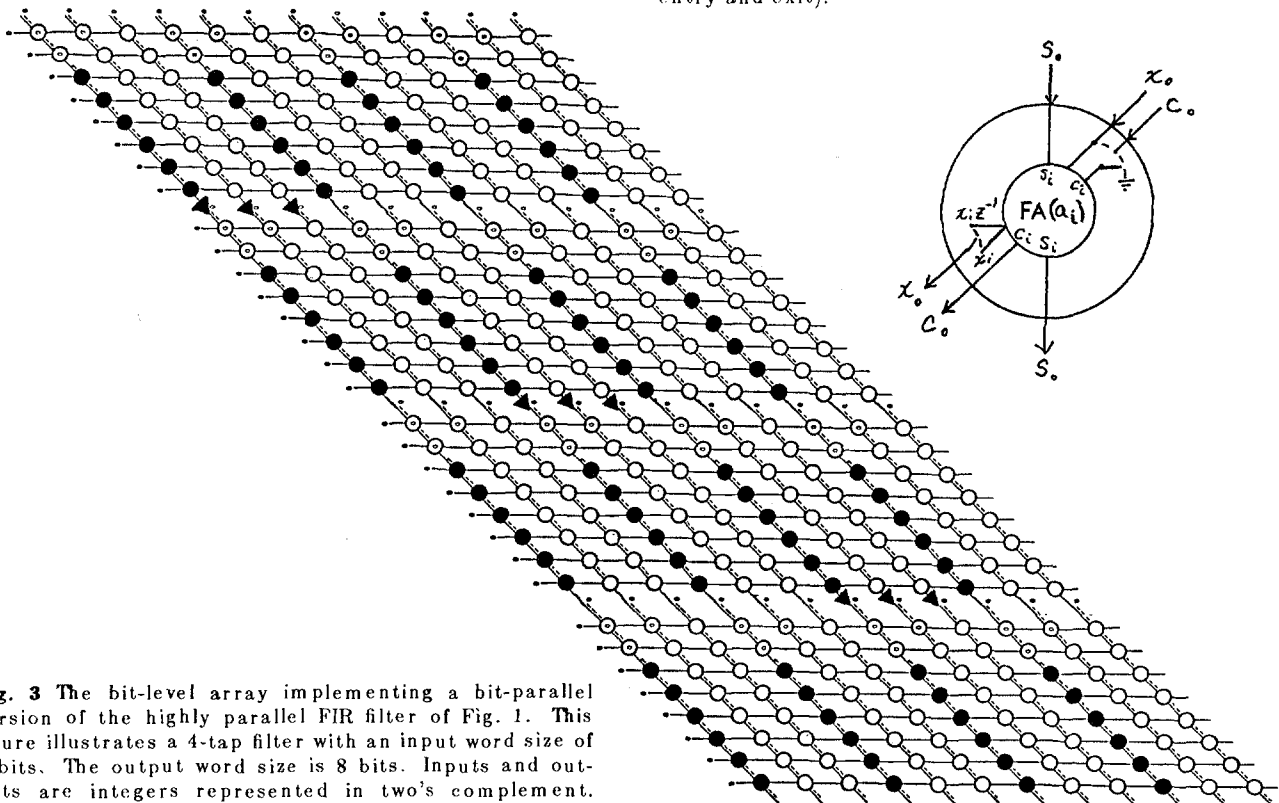


Fig. 3 The bit-level array implementing a bit-parallel version of the highly parallel FIR filter of Fig. 1. This figure illustrates a 4-tap filter with an input word size of 3 bits. The output word size is 8 bits. Inputs and outputs are integers represented in two's complement. Each white circle represents a latched full adder as in Figure 2. The black circles represent cells that use the complement of the a -bit. The black triangles represent delay elements, and are used to delay the input data stream, x , as required by the word-level algorithm. The circles that have a '0' in them represent an $a=0$ encoded into the cell. These cells merely keep the pipeline synchronized.

Fig. 4 A schematic of the outer cell. The data in the inner cell have a subscript of i (on both entry and exit); the data in the outer cell have a subscript of o (on both entry and exit).