A new structure for the implementation of bit/serial adaptive IIR filter is presented. The bit level system consists of gated full adders for the arithmetic unit and data latches for the data path. This approach allows recursive operation of the IIR filter to be implemented without any global interconnections, minimal delay time, chip area and I/O pins. The coefficients of the filter can be updated serially in real time for time invariant and adaptive filtering. A fourth order bit/serial IIR filter is implemented on a 2 micron CMOS technology clocked at 55 MHz.
VLSI Implementation of Adaptive BIT/Serial IIR Filters
by
Rajeev Badyal

A THESIS
submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of
Master of Science

Completed January 29, 1992
Commencement April 1992
APPROVED:

Redacted for Privacy

Assistant Professor of Electrical and Computer Engineering, in charge of major

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented January 29, 1992
Typed by Rajeev Badyal for Rajeev Badyal
# TABLE OF CONTENTS

1. INTRODUCTION  
   1.1 GOALS AND OUTLINE  

2. SYSTOLIC IMPLEMENTATION OF IIR FILTER  
   2.1 IIR FILTER STRUCTURE  
   2.2 OVERVIEW OF THE MULTIPLIER CELL DESIGN  

3. IMPLEMENTATION OF IIR FILTER CHIP  
   3.1 FILTER CELL  
   3.2 FOURTH ORDER IIR FILTER  
   3.3 LAYOUT  
   3.4 TIMING  

4. CONCLUSION  

BIBLIOGRAPHY
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IIR FILTER BLOCK</td>
</tr>
<tr>
<td>2</td>
<td>FILTER CELL</td>
</tr>
<tr>
<td>3</td>
<td>INTERACTION OF THE BITS IN WORD ALK AND BLK TO FORM C11</td>
</tr>
<tr>
<td>4</td>
<td>BIT LEVEL CELL FOR MULTIPLICATION</td>
</tr>
<tr>
<td>5</td>
<td>BASIC ARRAY REQUIRED FOR PIPELINED MULTIPLICATION</td>
</tr>
<tr>
<td>6</td>
<td>FORMATION OF INNER PRODUCT</td>
</tr>
<tr>
<td>7</td>
<td>MAIN ARRAY CELL SHOWING LOGIC FUNCTION FOR 2'S COMPLIMENT ARITHMETIC</td>
</tr>
<tr>
<td>8</td>
<td>ACCUMULATOR CELL SHOWING LOGIC FUNCTION OPERATION</td>
</tr>
<tr>
<td>9</td>
<td>SERIAL/PARALLEL MULTIPLIER</td>
</tr>
<tr>
<td>10</td>
<td>CIRCUIT FOR LOADING COEFFICIENTS</td>
</tr>
<tr>
<td>11</td>
<td>FULL ADDER</td>
</tr>
<tr>
<td>12</td>
<td>LAYOUT OF A 1-BIT STATIC FULL ADDER</td>
</tr>
<tr>
<td>13</td>
<td>LAYOUT OF A 1-BIT MULTIPLIER CELL</td>
</tr>
<tr>
<td>14</td>
<td>D FLIP FLOP</td>
</tr>
<tr>
<td>15</td>
<td>SPICE SIMULATION OF THE CRITICAL PATH</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

The demand for high speed real-time signal processing has led to many new and innovative array architectures for digital filtering. A digital filter is a computational process in which the sequence of input signals is converted into a sequence of output signals representing the alteration of the data in some prescribed manner. A common example is the process of filtering out a certain range of frequencies in a signal while rejecting all other frequencies.

Systolic arrays have played a significant role and have been applied in many algorithms for digital signal processing. A systolic system consists of a set of interconnected cells, each capable of performing simple operations. Uniform, regular communication and control structures of these arrays offer a substantial advantage for the VLSI design and implementation of many algorithms. Information in a systolic system flows between cells in a local pipelined fashion. The communication with the outside world occurs only at the "boundary" cells where only those cells on the array boundaries may be I/O ports of the system. Advantages of these arrays include modular
expansionability, regular data and control flows, uniform cells, elimination of global broadcasting, limited fan-in and fast response time.

The bit level systolic techniques have been applied to the design of many non-recursive components such as multipliers/adders, FIR filters, correlators, discrete Fourier transform and other DSP algorithms [1-2]. This approach, however, has had a limited application to the implementation of recursive algorithms such as IIR filters [3-4].

1.1 GOALS AND OUTLINE

This thesis introduces a bit serial implementation of a high speed adaptive serial/parallel IIR filter using the architectures presented in [5] and discusses some of the practical issues involved in the VLSI implementation of these architectures. The filter is adaptive where the coefficients $a_1, a_2, \ldots a_n$ and $b_1, b_2, \ldots b_n$ can be altered on line to generate high pass, low pass or bandpass filtering. The serial/parallel implementation reduces the number of I/O pads, due of serial data input, and reduces the chip area.

Figure 1 shows an architecture of a fourth order IIR filter and the internal structure of each cell is depicted in figure 2. It is necessary that the cell area is minimized and at the same time high throughput is obtained. Each cell consists of two multipliers which occupy the largest area. The goal is to implement area efficient multipliers.

Highly parallel algorithms for multiplication have been presented in [6-8]. The detail of the different mutipliers will be discussed in the
next chapters. These parallel array-multipliers provide high throughput but require large area as well as a large number of I/O pins. Parallel array multipliers are expensive and non-practical for large order filter implementation. Alternative bit-serial methods are investigated in this thesis.

Chapter two discusses the systolic architectures for IIR filters presented by [5]. Different approaches for the internal arithmetic architectures and their advantages and disadvantages are discussed in this chapter. Chapter three describes the complete design and implementation of a fourth order IIR filter chip using filter cell approach discussed in Chapter 1 and 2. Chapter three discusses the general structure of the 8-bit filter cell followed by the implementation and operation of the fourth order filter. The layout, pinout and size of the chip and the timing requirement of the chip is also discussed in chapter three, followed by a conclusion.
Figure 1: IIR filter block
Figure 2: Filter cell
Chapter 2

SYSTOLIC IMPLEMENTATION OF IIR FILTER

Traditionally IIR filters have been implemented using microprocessors programmed to perform the necessary arithmetic operation on the digital data to obtain the desired filter type. This sequential approach limits the throughput of the system and the parallelism of the system is not exploited.

One of the most important applications of systolic arrays has been in filter designs. The systolic array approach exploits the parallelism of the IIR filters.

2.1 IIR FILTER STRUCTURE

An IIR filter can be expressed in the recursive form as

\[ y(k) = a_1 y(k-1) + a_2 y(k-2) + \ldots + a_n y(k-n) + b_0 u(k) + b_1 u(k-1) + b_2 u(k-2) + \ldots + b_n u(k-n) \]  

Taking z transform of (1)

\[ Y(z) = a_1 Y(z)z^{-1} + a_2 Y(z)z^{-2} + \ldots + a_n Y(z)z^{-n} + b_0 U(z)z^{-1} + b_1 U(z)z^{-2} + \ldots + b_n U(z)z^{-n} \]
\[ H(z) = \frac{N(z)}{D(z)} = \frac{[b_0 + b_1 z^{-1} + \ldots + b_n z^{-n}]}{[1 - a_1 z^{-1} + \ldots + a_n z^{-n}]} \]

The first part \( \sum_{i=1}^{n} a_i y(k-i) \) or \( 1/D(z) \), is known as the

Auto-Regressive (AR) and the second part \( \sum_{i=0}^{n} b_i u(k-i) \), or \( N(z) \) is the moving average (MA) part of the filter. An architecture for the direct implementation of IIR filter is shown in figure 1. This \( N^{th} \) order filter architecture is a cascading of basic filter cells where the number of cells used for the filter is equal to the order of the filter. Figure 2 shows the internal design of the filter cell. Each cell consists of two multipliers, two adders and two \( n \)-bit registers containing the coefficients \( a_1, a_2, \ldots, a_n \) and \( b_1, b_2, \ldots, b_n \). Efficient VLSI implementation of this filter requires minimal area and high throughput of the inner product (multiplier and adder) block for each cell. Highly parallel algorithms for multiplication have been presented [6-8]. In general these algorithms require \( n^2 \) basic cells (full adders) where \( n \) is the number of bits of the input data word. For the IIR filter cell, \( 2n^2 \) basic cells are required and hence \( 2Nn^2 \) multiplying basic cells are required for an \( N^{th} \) order filter. These multipliers provide high throughput but require large area and I/O pins which makes it prohibitive for higher order filter design.
2.2 OVERVIEW OF THE MULTIPLIER CELL DESIGN

In this section various approaches for the design of multipliers presented in [6-8] are discussed.

McCanny et al., [6] : They have presented a bit level multiplier as illustrated in figure 3. Each square represents a gated full adder unit into which the input bits are latched. The data bits are expanded horizontally. The organization of bits is such that each word enters the array in a bit-serial manner, the words $a_{lk}$ from the right and words $b_{kl}$ from the left with the least significant bit in each word $(a_{lk0}, b_{kl0})$ entering ahead of the next significant bit $(a_{lk1}, b_{kl1})$ and so on. On each pulse of a system clock bits in the words $a_{lk}$ move one cell to the left whilst bits in the words $b_{kl}$ move one cell to the right and as these interact any partial products which are formed are passed vertically downwards. Any carry bits which are generated in the process remain on fixed sites and these are latched from the output of a cell to its input on each clock cycle. Each row of cells within a diamond shaped region contains all the partial product sums required to form a bit of given significance in the result. In order to complete the full sum of products operations the partial product bits which are formed within each row must be accumulated as the diamond emerges from the array. This final accumulation process can be carried out by adding a pipelined tree of adder cells to the bottom of the array. The results from this can be
clocked out in a bit serial manner, least significant bit first and a complete result is formed every 2m-1 clock cycles. This architecture requires 2n^2-n cells, each cell looking like figure 4, and has a 50\% efficiency. For the IIR filter cell, 2(2n^2-n) cells are required and hence 2N(2n^2-n) cells are required for an N^{th} order filter.
Figure 3: Interaction of the bits in words alk and bkl to form c11
McCanny et al., [7]: They have presented a completely iterative, pipelined multiplier array as shown in figure 5. It comprises of a diamond-shaped array of $m^2$ latched, gated full adder cells each of which is connected only to its nearest neighbors. All the inputs and outputs of each cell are latched. The operation of each cell is illustrated in figure 5. It performs the 1-bit logic function:

$$p = a \times b; \quad s = s' \text{ XOR } c' \text{ XOR } p;$$

$$c = (s' \times c') + (c' \times p) + (s' \times p)$$

where $a$ and $b$ represent individual bits of the two numbers to be multiplied, $s'$ is one bit of the accumulating sum of partial products.
and \( c' \) is a bit carried in from the previous stage of the calculation. The resulting value of \( s \), the corresponding carry bit \( c \) and the input bits \( a \) and \( b \) are all stored in latches and then passed on to neighboring cells. The value of \( s' \) is also latched on input to each cell.

The \( n^{th} \) pair of numbers \( a(n) \) and \( b(n) \) to be multiplied are input to the circuit along the upper edges of the array with their constituent bits \( a_k(n) \) and \( b_k(n) \) staggered in time, as indicated by means of the external latches in Fig. 5. The numbers are arranged such that the most significant bit of \( a(n) \) (i.e. \( a_{m-1}(n) \)) and the least significant bit of \( b(n) \) (i.e. \( b_0(n) \)) enter the circuit on one clock cycle, the second most significant bit of \( a(n) \) and the second least significant bit of \( b(n) \) on the next clock cycle, and so on. This arrangement ensures that as each bit of \( a(n) \) moves across the array, it meets every bit of \( b(n) \) - one at each of the cells which it crosses. The \( k^{th} \) bit of each partial product \( a_{k-i}(n)b_i(n) \) is formed on one of the cells within the \( k^{th} \) vertical column, and the \( k^{th} \) bit of the product

\[
s_k(n) = \sum_{i=0}^{k} a_{k-i}(n) b_i(n)
\]

is formed by letting these components accumulated as \( s_k(n) \) passes down that column. The sum bits \( s'_k(n) \) and carry bits \( c'_k(n) \) which enter the array boundaries are set equal to zero, and the carries which are generated at each stage within the array are simply passed to the left
(the next most significant column) on the next clock cycle.

This circuit constitutes a pipelined carry-save multiplier and, since the carries do not have to ripple through at each stage of the calculation, the clock speed is limited only by the propagation delay within a single cell. However, as with any carry-save device, the residual carry bits which leave the basic multiplier circuit across the lower left-hand boundary in figure 5 must be added into the sum of partial products in order to complete the multiplication. This architecture requires \( n^2 \) cells and hence \( 2Nn^2 \) cells for an \( N^{th} \) order filter.
\[ b_2(n) \quad b_2(n-1) \quad b_2(n-2) \quad b_1(n) \quad b_1(n-1) \quad b_1(n-2) \]

\[ b_0 \quad w_0 \quad 0 \quad 0 \quad N \quad 1 \quad \ldots, \]

\[ S' \quad a \quad c' \quad S < S' \text{ xor } (a \cdot b) \text{ xor } c' \]

\[ c < (a \cdot b) \cdot S' + (a \cdot b) \cdot c' + S' \cdot c' \]

Figure 5: Basic array required for pipelined multiplication
R.B. Urquhart et al., [8] : They have proposed a 100% efficient architecture for multiplication. Consider the formation of a product $G$ of a data word $X$ and coefficient word $W$, where $X$ and $W$ are of $B$ and $C$ bits, respectively, i.e.

$$X = (x_{B-1}, x_{B-2}, \ldots, x_1, x_0)$$

$$W = (w_{C-1}, w_{C-2}, \ldots, w_1, w_0)$$

$$G = (g_{B+C-1}, \ldots, g_1, g_0)$$

$G$ can be formed by summing the partial products $g_{ij} = x^{(i)}w^{(j)}$ providing the significance of partial products is taken into account. If we store the bits of $W$ across a linear chain of $C$ cells, a parallelogram of partial products can be formed by passing the bits of the $X$ word across the chain. If the least significant bits of each word interact first, partial products of equal significance will emerge from the chain simultaneously. Alternatively, if the bits of the coefficient word $W$ are reversed, partial products of equal significance appear at a slant within the parallelogram. In either case, if a sequence of $X$ words are multiplied by $W$, their parallelograms of partial products will butt, resulting in a 100% utilization of processing elements.

Both of these schemes are easily extended to the formation of the inner product $G^*$ of vector $X=(X_1, X_2, \ldots, X_N)$ and $W=(W_1, W_2, \ldots, W_N)$, i.e.
\[ G^* = \sum_{k=1}^{\infty} x_k w_k \]

Input data is skewed and enters the array least significant bit first (Fig. 6). Each main array cell comprises a gated full adder and four latches for positive valued data or a gated full adder and five latches for two's complement data (Fig. 7). The parallelogram of partial products will move down through the array accumulating contributions from each term in the inner product. Since succeeding partial products are of greater significance, carries are recirculated. Wordlength growth is allowed for by adding a guard band of zeros (\(\log_2 N\) bits wide) at the end of each input vector. The inner product word \(G^*\) must be obtained by accumulating the parallelogram of partial products. Each accumulator cell (Fig. 8) consists of a full adder and full latches. This architecture requires \(n^2\) processing elements and hence \(2Nn^2\) cells for an \(N^{th}\) order filter.
Figure 6: Formation of an inner product
Figure 7: Main array cell showing logic function for 2's complement arithmetic

\[
S' \leftarrow S \oplus (xw \oplus CTRL) \oplus C
\]

\[
C' \leftarrow SC + S(xw \oplus CTRL) + C(xw \oplus CTRL)
\]

Figure 8: Accumulator cell showing logic function operation

\[
S' \leftarrow S \oplus g \oplus C
\]

\[
C' \leftarrow SC + Sg + gC
\]
R.F. Lyon [9]: An alternative approach to these multipliers is the bit serial/parallel multiplier presented by R.F. Lyon [9]. This implementation requires \( n \) processing elements for an \( n \)-bit data word and takes \( 2n \) clock cycles to multiply two \( n \)-bit numbers as shown in figure 9. Each \( n \)-bit data word is followed by \( n \)-bits of zero's inorder to flush the multiplier pipeline. For large \( n \) the serial/parallel architecture occupies smaller area and requires lesser I/O pins. The disadvantage of this multiplier over the array multipliers is that the bit serial/parallel multiplier has a lower throughput (\( 2n \) clock cycles as compared to every clock cycle for an array multiplier).
Table 1 shows the comparison of the serial/parallel architecture and the array multiplier implementations.
<table>
<thead>
<tr>
<th></th>
<th>ARRAY/PARALLEL MULTIPLIER</th>
<th>SERIAL/PARALLEL MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Of one bit multiplying elements per cell</td>
<td>$2n^2$</td>
<td>$2n$</td>
</tr>
<tr>
<td>Area per cell</td>
<td>$o(2n^2)$</td>
<td>$o(2n)$</td>
</tr>
<tr>
<td>Throughput</td>
<td>every clock cycle (after initial pipeline is full)</td>
<td>every $2n$ clock cycles</td>
</tr>
<tr>
<td>#Of one bit multiplying elements per Nth order filter</td>
<td>$2^{N2n}$</td>
<td>$N2n$</td>
</tr>
</tbody>
</table>

Table 1: Comparison of serial/parallel and array multipliers
CHAPTER 3

IMPLEMENTATION OF THE IIR FILTER CHIP

This chapter describes the design and implementation of an 8-bit input - 16 bit output fourth order serial/parallel IIR filter chip using the approach discussed earlier in Chapter 1 and 2. Section 3.1 discusses the general structure of the 8-bit filter cell. Section 3.2 discusses the implementation and operation of the fourth order filter using the filter cell. Section 3.3 discusses the layout, pinout and size of the chip and section 3.4 describes the timing requirement of the system.

3.1: FILTER CELL:

The general structure of the cell is shown in figure 2. An 8-bit serial/parallel implementation of this filter cell consists of two 8-bit serial/parallel multipliers two 8-bit shift registers and two 1-bit carry ripple full adders, as shown in figure 9.

The two serial/parallel multipliers receive bit serial inputs from the previous stage. The output of each stage is delayed by the 16-bit serial shift register to the input of the next cell. The Auto Regressive (AR) part of the cell receives u(k)'s from the boundary cell and the output y(k) is fed back to perform the Moving Average (MA) part.

The filter coefficients $a$ and $b$ are loaded serially from an external pin (section 3.3). The coefficients loading is performed by a chain of eight serially connected D-flip flops as shown in figure 10. The clock
signal to these flip flops is ANDed with a load coefficient signal provided externally which is held high for eight clock cycles to load the coefficients serially.
Figure 10: Circuit for loading coefficients
3.2: FOURTH ORDER IIR FILTER:

The fourth order system consists of four filter cells with two multiply and add units in each cell. The last cell feeds back the output signal $y(k)$ as shown in figure 1. The feedback consists of a 16-bit serial shift register which provides the necessary delay to the $u(k)$ inputs. Further this register also performs the truncation necessary to generate zero's to flush the multiplier results. The data among cells is transfered through a 16-bit shift register which provides the delay. The system input requirement is an 8-bit bit serial input, LSB in first, followed by an 8-bit string of zero's. These zero's are added to flush the higher 8-bits of the product $a_n \times u(k)$.

A complete snap shot of the filter operation is shown below:
After the system is initialized, at each consecutive clock cycle an output data bit is obtained. The length of the output word is 16 bits. The intermediate truncations are provided by the ANDing of the input
to the 16-bit shift register with the externally provided AND control signal.

3.3: LAYOUT:

The fourth order IIR filter was implemented using the Mentor Graphics CAD tools in CMOS 2 micron technology. The size of the chip without the pads is 3600 microns * 3400 microns. The system was designed using static CMOS design approach. The static design approach allowed for a single phase simple clocking scheme. Figure 11 and 12 show the circuit and the layout of the static full adder. The size of the full adder was 228 microns * 183 microns and was used for the multiplier and the add unit. Layout of a 1-bit multiplier cell is shown in figure 13 which is 290 microns * 387 microns. It consists of a full adder, three D flip flops, a two input NAND gate and an inverter. The one bit multiplier was cascaded with seven similar cells in series to implement the 8-bit multiplier.
Figure 11: Full adder
Figure 12: Layout of a 1-bit static full adder
Figure 13: Layout of a 1-bit multiplier cell
The 16-bit shift register was implemented by serially cascading sixteen D flip flops. Figure 14 shows the circuit diagram of the D flip flop. To ease interconnection among D flip flops the input, output and the CLK signal ports have the same height in the layout.
The entire chip requires 16 pins and is shown below.
3.4: TIMING:

SPICE simulations for different blocks of the chip indicate that maximum delay occurs at the output of the full adder driving a D flip flop. Figure 15 shows the SPICE simulation for this block. A delay of 9 ns from high to low and from low to high transitions resulted in a 55
MHz maximum operating frequency.

The timing requirement for the chip are shown below.

The timing diagram for the chip indicates that the input is latched at the leading edge of the clock and a valid output is obtained at the trailing edge of the clock. The AND control signal is maintained low for the first 8 clock cycles and high for the next 8 cycles which is repeated.
Figure 15: SPICE simulation results of the critical path
In this thesis, the implementation of a fourth order bit/serial adaptive IIR filter was presented. This was achieved by designing and simulating a set of bit level cells such as multipliers and adders. The most important part of the filter structure is that it allows the recursive computation to be implemented directly and the system is modular for higher order filters.

Direct comparison between bit/parallel and bit/serial architectures was presented. The results indicated that the bit/serial method is more efficient in terms of chip area and I/O interface with minimum delay between cells. Based on the approach by R.F. Lyon [9] a fourth order 8-bit IIR filter was implemented. The size of the system was 3600 microns * 3400 microns with a speed of 55 MHz.
BIBLIOGRAPHY


