A Coarse-Grained FPGA Architecture for High-Performance FIR Filtering

James R. Anderson
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052
(408)765-0097
james_r_anderson@ccm.sc.intel.com

Siddharth Sheth
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052
(408)765-6968
ssheth@td2cad.intel.com

Kaushik Roy
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907
(765)494-2361
kaushik@ecn.purdue.edu

1. ABSTRACT

This paper introduces a coarse-grained FPGA architecture that is specialized for high-performance Finite Impulse Response (FIR) filtering. The proposed architecture provides the flexibility of a DSP processor with performance and area efficiency similar to that of a custom ASIC design, while allowing all of the basic FIR design parameters, including coefficient precision, to be configured. Previous research has already shown that FPGAs can provide a high-performance alternative to DSP processors. Experimental comparisons in this paper show that the performance and area efficiency of the proposed architecture is similar to that of custom approaches across a wide range of filter sizes and configurations.

1.1 Keywords

Field Programmable Gate Array (FPGA), architecture, Finite Impulse Response (FIR) filtering, digital signal processing (DSP)

2. INTRODUCTION

Digital signal processing systems are generally computationally intensive and require tremendous I/O bandwidth. In order to meet the performance demands of DSP systems, many custom hardware systems rely on ASICs. However, ASICs are inherently inflexible, and development of such systems is costly and time-consuming. DSP processors provide a flexible, low-cost alternative, but are performance-limited.

FPGAs provide a third alternative that maintains the flexibility of the DSP processor via re-programming while providing performance levels significantly beyond that of DSP processors. Because of the flexibility and performance characteristics, FPGAs have already been considered for implementing DSP systems [3]. One recent study found that an order of magnitude performance increase over that of a DSP processor is possible for an FPGA-based DSP system [6]. Other special purpose systems, such as the reconfigurable Splash [2] and PAM [4] systems, have also been implemented using FPGAs for custom applications. DSP systems based on programmable architectures designed for low power dissipation have also been studied [1], [7].

In this paper we target the application of FIR filtering, which has widespread use in many DSP systems. We present a specialized cell which can be arranged in a variety of topologies to provide an FPGA architecture targeted for FIR filtering. The objective of our cell-based approach is to provide an architecture that allows all of the basic FIR filter design parameters to be configured, yet provides the performance and area efficiency of custom designed ASIC approaches.

The next section gives an overview of FIR filtering and discusses two custom design approaches that provide upper and lower guidelines on performance and implementation area. Section 4 introduces the cell-based approach by first describing a basic cell for 8-bit precision, and then describing an enhanced cell which can be used for 8, 16, or 24-bit precision. Section 5 compares the performance and layout area of several sizes of cell-based implementations with the two custom design approaches discussed in Section 3. Section 6 concludes the paper.

3. CUSTOM APPROACHES

3.1 Overview

In mathematical terms, a sampled data FIR filter can be represented by the convolution sum,

$$y(n) = h_0 x(n) + h_1 x(n-1) + \ldots + h_N x(n-N) \quad (1)$$

where *y(n)* is the filtered output at sample *n*, *x(n)* is the input at sample *n*, h_i are the impulse response coefficients, and *N* is the order of the filter [5]. A straight-forward method of implementing an FIR filter is to directly evaluate the sum of products in the convolution sum. This method requires storing the most recent *N* samples of the input, multiplying each sample by the corresponding filter coefficient, and summing the products. Figure 1 shows a block diagram of a 4-tap FIR filter, where each filter tap consists of a delay element, an adder, and a multiplier.

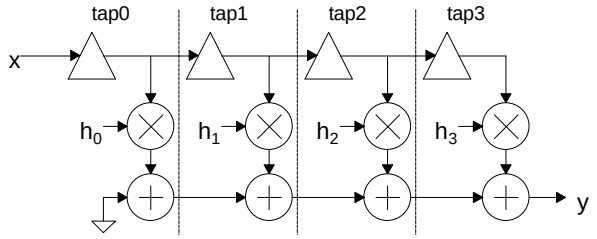

Figure 1. 4-tap FIR filter.

FIR filter characteristics, such as passband ripple, stopband attenuation, and transition bandwidth, depend on the value of the coefficients, the precision of the coefficients and input data, and the length of the filter (number of taps). The cutoff frequency depends on the sample rate, or the rate at which input samples enter the filter. Thus, other than the actual values used for the filter coefficients, there are only three basic design parameters: precision, filter length, and sample rate. As will be discussed later, our method allows each of these design parameters to be reconfigured.

The FIR implementation shown in Figure 1 is referred to as a *tapped delay line* structure. However, many implementation topologies exist for FIR filtering. In an alternative implementation, the delay elements could be placed between the adders in the taps. Also, if the coefficients are known to be symmetric, $h_{-n}=h_n$, which is usually the case in a linear phase filter, only one multiplier is required for every two taps, reducing the number of multipliers by a factor of two [8]. It is also often the case that every other coefficient is zero, allowing the multiplier and adder in every other tap to be eliminated.

For the purposes of this paper we will not assume any a priori knowledge of the filter coefficients, other than their precision. Thus, all the following approaches to FIR filtering use the tapped delay line method shown in Figure 1.

For *custom designed* FIR filters, there are two approaches which provide guidelines for comparisons of performance and hardware requirements among different FIR implementations: parallel and sequential. The parallel approach seeks maximum performance with no constraint on hardware. In contrast, the sequential approach minimizes the hardware requirements while yielding the poorest performance. Both approaches will be discussed in the following sections.

3.2 Parallel Approach

In order to maximize the sample rate of an FIR filter, we must reduce the length of the critical path. Referring to Figure 1, if the delay elements are implemented as registers, then clearly the critical path runs from the output of the first register to the sample output *y*. In order to minimize the critical path length, the amount of computation performed in parallel should be maximized. This can be accomplished by performing all coefficient multiplications in parallel and organizing the adders in a binary tree structure. Using registers for the delay elements and latches to store the coefficients, Figure 2 shows the parallel implementation of a 4-tap FIR filter. Note that *shift_register* contains four registers connected in series, and *latch_file* contains four latches.

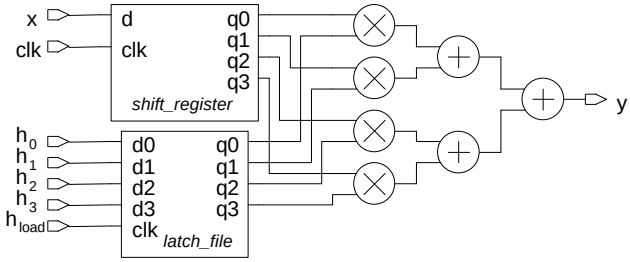

Figure 2. Parallel implementation.

Assuming coefficient precision is fixed, the sample rate and hardware area requirements of a parallel implementation for an *N*-tap FIR filter are,

$$SampleRate = \left(DelayMult + DelayAdd \cdot \lceil \log_2 N \rceil \right)^{-1} \quad (2)$$

$$Area = N\left(AreaReg + AreaLatch + AreaMult \right) + (N-1) \cdot AreaAdd \quad (3)$$

Of course, we have neglected register setup and hold times in (2) and interconnect area in (3). However, these equations are adequate for our purposes, since all design comparisons in following sections will be based on experimental data. The sample rate decreases inverse logarithmically and the hardware area increases linearly as the number of taps grows. Thus, in the comparisons of following sections, the parallel approach will provide an upper guideline for both performance and implementation area.

3.3 Sequential Approach

With the goal of minimizing hardware area requirements, the sequential approach seeks to reuse as much hardware as possible. Since multipliers and adders are area-costly,

only one of each is used. Each product term in (1) is computed sequentially and added to the accumulated sum in an adder/register combination. However, in order to sequentially select the input sample and coefficient to be multiplied, tri-state buffers and simple control logic are required. Figure 3 shows the sequential implementation of a 4-tap FIR filter. The controller simply counts from 0 to (*N*-1) and enables the next tri-state buffer, which allows the next *q*-output of the *shift_register* and *latch_file* to be multiplied at each count. On each 0th count, the *dshift* is activated to shift the data samples and *zero* is activated to set the register of the accumulator to zero.

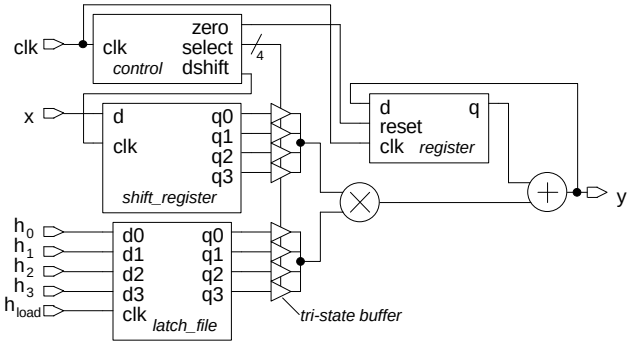

Figure 3. Sequential implementation.

Although the critical path for this approach is shorter than that of the parallel approach, the circuit must be cycled *N* times to produce a single output sample. Thus, the sample rate and hardware area of an *N*-tap sequential implementation are,

$$SampleRate = \left[N\left(DelayBuffer + DelayMult + DelayAdd \right) \right]^{-1} \quad (4)$$

$$\begin{aligned} Area = {} & N\left(AreaReg + AreaLatch + 2 \cdot AreaBuffer \right) \\ & + \left\lceil \log_2 N \right\rceil \cdot AreaControl + AreaMult \\ & + AreaAdd + AreaReg \end{aligned} \quad (5)$$

Once again, register setup and hold times as well as interconnect area have been neglected. The value of *AreaControl* is the area for the control circuitry for a 2-tap FIR. This area grows by a factor of $\log_2 N$ as the number of taps grows. Of course, this is a rough approximation, but it is adequate for our purposes.

The sample rate now decreases inverse linearly as the number of filter taps is increased. Thus, the performance of the sequential approach degrades more quickly than that of the parallel approach. The area of the sequential approach increases linearly with *N*. However, since the multiplier and adder area do not change with *N* and the buffer area is not significant, the area of the parallel approach grows much quicker than that of the sequential approach as *N* increases. Thus, the sequential approach provides a lower guideline for both performance and implementation area.

4. CELL-BASED APPROACH

The objective of the cell-based approach is to provide an FPGA architecture for FIR filtering applications that has performance and implementation area similar to that of custom-designed circuits. Specifically, the architecture is intended to provide the high performance of a custom parallel approach or the area efficiency (in terms of filter taps per unit area) of a custom sequential approach. In addition, the architecture is designed to allow the circuit to be easily configured for design points at or between these two extremes. The following section discusses the basic reconfigurable cell which meets these objectives and serves as an introduction to the enhanced reconfigurable cell, which is discussed in the next section.

4.1 Basic Reconfigurable Cell

The basic cell is a simple extension of the sequential implementation of a 4-tap FIR shown in Figure 3. The extension allows cells to be connected in a topology that enables the number of filter taps and data sample rate to be user-configurable. Figure 4 shows the basic cell. We assume that the filter coefficients and input samples are 8-bit single precision.

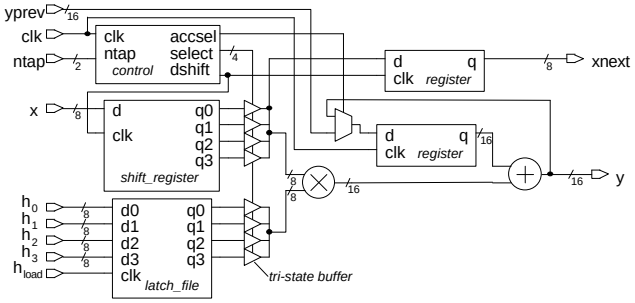

Figure 4. Basic cell.

The cell of Figure 4 is designed to implement between one and four filter taps, depending on the value of the *ntap* input on the control block. In order to extend the sequential implementation, a mux and a register have been added and the control block has been changed to include some additional functionality. The addition of the mux allows an initial value to be loaded into the accumulator register before the summation of the products begins. This mux and the addition of the register connected to *xnext* output allows cells to be placed in series to create chains of cells implementing FIR filters with a large number of taps. This will be discussed in more detail shortly. Let us first look at the operation of a single cell.

Table 1 shows the operation of a single cell, if the cell is used to implement four taps (*ntap*=3). Assume each line of the table is a clock period and a new data sample, labeled d0, d1, etc., arrives at input *x* once every four clocks. Also, assume that the *yprev* input has a value of zero and that the filter coefficients *h*0 through *h*3 have already been loaded into the *latch_file*. Table 1 lists the values held by the *shift_register*, the values on the *select* output of the

control block, and output *y*. The *acc_reg* column denotes the value held by the 16-bit accumulator register.

Table 1. Single cell operation (refer to Fig. 4).

Smp#	*sel*	*shift_reg*	*acc_reg*	*y*
0	0001	d0,0,0,0	0	0+h0·d0
0	0010	d0,0,0,0	h0·d0	h0·d0+h1·0
0	0100	d0,0,0,0	h0·d0	h0·d0+h2·0
0	1000	d0,0,0,0	h0·d0	h0·d0+h3·0
1	0001	d1,d0,0,0	0	0+h0·d1
1	0010	d1,d0,0,0	h0·d1	h0·d1+h1·d0
1	0100	d1,d0,0,0	h0·d1+h1·d0	h0·d1+h1·d0+h2·0
1	1000	d1,d0,0,0	h0·d1+h1·d0	h0·d1+h1·d0+h3·0
2	0001	d2,d1,d0,0	0	0+h0·d2
2	0010	d2,d1,d0,0	h0·d2	h0·d2+h1·d1
2	0100	d2,d1,d0,0	h0·d2+h1·d1	h0·d2+h1·d1+h2·d0
2	1000	d2,d1,d0,0	h0·d2+h1·d1 +h2·d0	h0·d2+h1·d1+h2·d0 +h3·0
3	0001	d3,d2,d1,d0	0	0+h0·d3
3	0010	d3,d2,d1,d0	h0·d3	h0·d3+h1·d2
3	0100	d3,d2,d1,d0	h0·d3+h1·d2	h0·d3+h1·d2+h2·d1
3	1000	d3,d2,d1,d0	h0·d3+h1·d2 +h2·d1	h0·d3+h1·d2+h2·d1 +h3·d0
4	0001	d4,d3,d2,d1	0	0+h0·d4

As shown by the table, the sum of the four products becomes available after the fourth clock edge. This value can be passed as output to the user or as input to *yprev* of the next cell in the chain. The *ntap* input can be adjusted to zero, in which case the cell would only implement a single tap and a new data sample would arrive on each clock. In this case, only the first register and latch in the *shift_register* and *latch_file*, respectively, are used (*select* is always 0001), and a new data output is available on each clock.

The basic cells can be arranged in a topology that allows for a variety of configurations. Figure 5 shows a topology that could be used for eight cells. For the sake of clarity, the filter coefficient inputs to the basic cells have been omitted. Depending on the mux settings and the number of taps implemented by each cell, some of the possible configurations for the circuit are,

1. Four independent filters (2 cells each) with 2 to 8 taps each (*sel1*=1, *sel2*=1, *sel3*=1);
2. Two independent filters (4 cells each) with 4 to 16 taps each (*sel1*=0, *sel2*=1, *sel3*=0); or
3. One filter (8 cells) with 8 to 32 taps (*sel1*=0, *sel2*=0, *sel3*=0).

For the first configuration, all *x* inputs and *y* outputs are used. However, for the second configuration, *x0* and *x2* are the inputs and *y1* and *y3* are the outputs, respectively. For the third configuration, *x0* is the filter input and *y3* is the filter output. The variety of configuration possibilities allows the same hardware to be used in many different design situations.

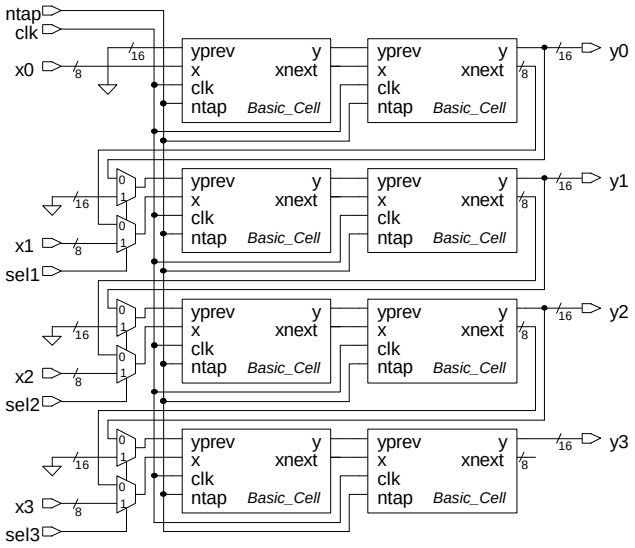

Figure 5. Example of a basic cell topology.

Table 2. Operation of two cells in series (Fig. 5).

Smp#	*y0*	*y1*
0	h0·d0	0
1	h0·d1+h1·d0	h0·d0
2	h0·d2+h1·d1+h2·d0	h0·d1+h1·d0
3	h0·d3+h1·d2+h2·d1+h3·d0	h0·d2+h1·d1+h2·d0
4	h0·d4+h1·d3+h2·d2+h3·d1	h0·d3+h1·d2+h2·d1+h3·d0
5	h0·d5+h1·d4+h2·d3+h3·d2	h0·d4+h1·d3+h2·d2+h3·d1 +h4·d0
6	h0·d6+h1·d5+h2·d4+h3·d3	h0·d5+h1·d4+h2·d3+h3·d2 +h4·d1+h5·d0
7	h0·d7+h1·d6+h2·d5+h3·d4	h0·d6+h1·d5+h2·d4+h3·d3 +h4·d2+h5·d1+h6·d0
8	h0·d8+h1·d7+h2·d6+h3·d5	h0·d7+h1·d6+h2·d5+h3·d4 +h4·d3+h5·d2+h6·d1+h7·d0
9	h0·d9+h1·d8+h2·d7+h3·d6	h0·d8+h1·d7+h2·d6+h3·d5 +h4·d4+h5·d3+h6·d2+h7·d1

The operation of a filter implemented using two cells in series, as with the top two cells of Figure 5, is illustrated in Table 2. The columns *y0* and *y1* represent the value on the *y* output for the first and second cell, respectively. Thus, *y0* is connected to *yprev* of the second cell. Assume that both cells are configured to implement four taps, so the two-cell filter has a total of eight taps. Unlike Table 1, each line of Table 2 represents four clock cycles, or a single sample period. The values shown are those at the end of the sam-

ple period (every fourth clock). Assume that filter coefficients *h*0 through *h*3 and *h*4 through *h*7 have already been loaded into the *latch_file* of the first and second cell, respectively. Once again, d0, d1, etc., represent the input data samples.

Although we have a chain of two cells implementing eight taps, the sample period is the same as that of the single cell implementing four taps, which was illustrated in Table 1. Although the number of taps has doubled, the sample period has remained at four clock cycles. Indeed, the performance of the cell-based approach does not depend on the number of cells in a chain. This characteristic is one of the primary strengths of using the cell-based approach, and allows designs to be easily scaled for larger filters without sample rate penalties.

Note in Table 2 that when a new input sample enters the filter, the filtered output is not available at the end of the sample period. Rather, the filtered output becomes available at the end of the following sample period. Thus, this approach has introduced an extra cycle of latency. Using the parallel or sequential approaches discussed earlier, the filtered output becomes available at the end of the same sample period during which the input arrived -- a latency of one sample period. Using the cell-based approach, an extra cycle of latency is added for each additional cell which is placed in the chain.

The additional latency is the primary drawback of the cell-based approach. For a very long filter that requires many cells in series, the latency may be too long for certain latency-critical, real-time applications. However, most applications can tolerate an increase in latency in exchange for a higher sample rate and greater data throughput. Of course, this drawback could be eliminated by implementing a binary tree of adders to globally sum the results from each cell at the end of each sample period. However, this type of approach would introduce a tremendous amount of hardware complexity and eliminate the ease with which the cell-based approach can be scaled.

For the cell shown in Figure 4, the critical path is from the controller through the tri-state buffers to the multiplier, adder, mux, and finally the accumulator register. Thus, the clock period of the cell is,

$$CellPeriod = DelayBuffer + DelayMult + DelayAdd + DelayMux \quad (6)$$

Again, setup and hold times have been neglected.

Each cell must cycle once for each of the filter taps it implements and each cell in a chain adds an additional sample period of latency. For a particular chain of cells, the sample rate and latency are given by,

$$SampleRate = \left(\lceil Taps/Cells \rceil \cdot CellPeriod \right)^{-1} \quad (7)$$

$$Latency = Cells\left(\lceil Taps/Cells \rceil \cdot CellPeriod \right) \quad (8)$$

where $Cells \le Taps \le 4 \cdot Cells$ for both (7) and (8).

If each cell is used to implement only one tap, the cell-based approach will have a sample rate similar to that of the parallel approach. As each cell is used to implement more taps, the sample rate will approach that of the sequential approach. Thus, not only does the ability to implement different numbers of taps using the same cell provide great configuration flexibility, it also provides performance flexibility.

When interconnection area is neglected, the area of the cell-based approach is approximately the area of a cell multiplied by the number of cells. Since a particular number of cells or given area can be configured to implement different sizes of filters, the efficiency with which the area is used can be configured. Thus, area efficiency can be traded for higher performance.

Recall that two of the three basic design parameters for an FIR filter are the number of taps and the sample rate. Both parameters can be reconfigured for different applications using the same hardware if a cell-based approach is used. The third parameter, coefficient precision, which we have assumed to be 8-bit single precision, cannot be adjusted using the cell-based approach discussed in this section. To make this parameter configurable, the basic cell must be enhanced for double or triple precision coefficients.

4.2 Enhanced Reconfigurable Cell

The enhanced cell is an extension of the basic cell and allows two or three cells to cooperate to form an FIR filter with double or triple precision coefficients. In other words, the same cells can be configured for 8, 16, or 24-bit filter coefficients and sample data. In order to understand the hardware additions which make this possible, let us first briefly describe a double precision multiplication.

We have two signed 16-bit values, d and h, which can be split into upper and lower bytes d1:d0 and h1:h0. The 16-bit values can be multiplied together using two 8-bit multipliers, *mult0* and *mult1*, and two 16-bit adders, *add0* and *add1*, in the manner shown by Figure 6. Note that *sx* stands for sign-extend in the figure. Bytes d0 and h0 are unsigned and bytes d1 and h1 are signed. Thus, the multipliers must be capable of handling unsigned, signed, and mixed-mode operands. Triple precision multiplication is a relatively straight-forward extension of this approach, which involves nine single precision multiplications.

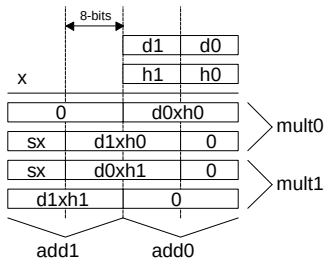

Figure 6. Double precision multiply.

Let us implement a double precision, single-tap FIR calculation using two simplified cells connected in the manner shown in Figure 7. For clarity, some components such as registers and control logic have been omitted. The 16-bit data sample, d1:d0, and filter coefficient, h1:h0, are split between the two 8-bit cells as shown. The box labeled *comb_logic* in each cell implements combinational multiplexing, sign extension, and zero extension operations. The control logic for the mux and *comb_logic* is not shown.

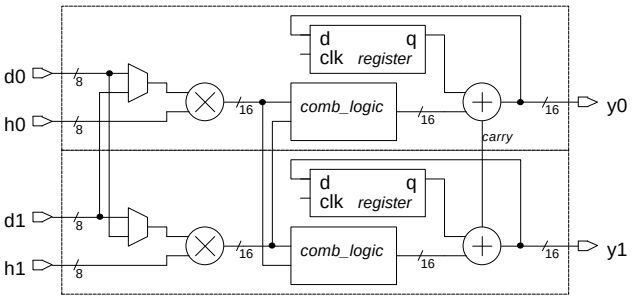

Figure 7. Double precision cell logic.

If we assume that the accumulator registers are initialized to zero, Table 3 shows the values at the output of the multipliers and the outputs *y1* and *y0* after each clock cycle. From the table, we can see that the double precision result (d1:d0)·(h1:h0)=(y1:y0) can be generated in three clock cycles. Thus, for double precision, three clock cycles are required for each tap implemented by the filter. For example, a cell implementing four taps would require twelve clock cycles.

Table 3. Double precision cell operation.
Note: sx() is sign-extend, ub() is upper-byte, lb() is lower-byte, and x is don't care.

clk#	*mult1*	*y1*	*mult0*	*y0*
0	d1·h1	0+d1·h1	d0·h0	0+d0·h0
1	x	sx(d1·h0):ub(d1·h0) +d1·h1	d1·h0	lb(d1·h0):0 +d0·h0
2	d0·h1	sx(d0·h1):ub(d0·h1) +sx(d1·h0):ub(d1·h0) +d1·h1	x	lb(d0·h1):0 +lb(d1·h0):0 +d0·h0

Using three 8-bit cells in an arrangement similar to that of Figure 7, we can generate a triple precision result in five clock cycles. Due to space limitations, we will for go the detailed description of triple precision operation. However, the final enhanced cell is shown in Figure 8. Note that the multiplier performs signed, unsigned, or mixed-mode calculations.

Depending on the *nbyte* and *nclk* settings, the cell can be configured for any of the three possible precision configurations. The *ubm* and *lbm* outputs are the upper and lower bytes from the multiplier, respectively. Similarly, the *ubpm* and *lbnm* inputs are the upper and lower bytes of the multipliers of the previous and next cells, in terms of byte order, respectively. Inputs *pxs* and *nxs* are connected to the *xs* outputs of the previous and next cells, in terms of byte order, respectively. Inputs *phs* and *nhs* are similar. The *ci* and *co* lines are required for carry propagation in the adder when double or triple precision is used.

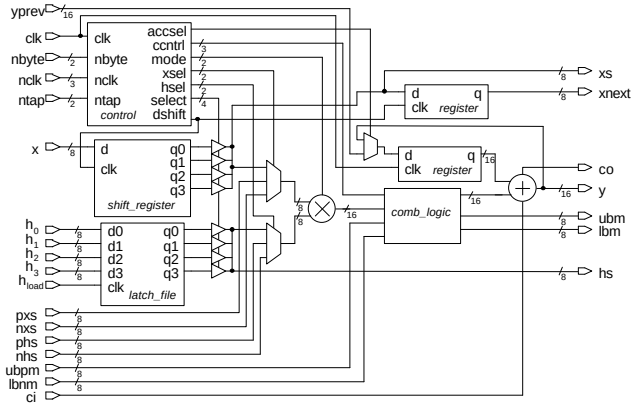

Figure 8. Enhanced cell.

Cells can be arranged in a variety of topologies which allow the same hardware to be configured for different filter applications. Figure 9 shows a small example of one such topology. Many of the details have been omitted in Figure 9 for the sake of clarity. Some of the possible configurations are described in Table 4.

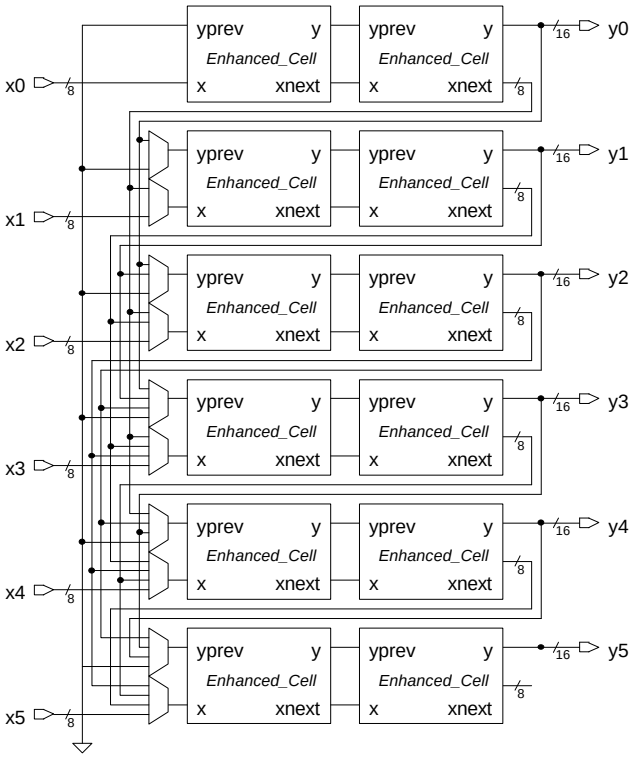

Figure 9. Example of enhanced cell topology.

Table 4. Some possible configurations of Figure 9.

Width	Filters	Taps	Inputs/Outputs of Filters
8-bits	1	12 to 48	x0/y5
8-bits	2	6 to 24	x0/y2, x3/y5
8-bits	3	4 to 16	x0/y1, x2/y3, x4/y5
8-bits	6	2 to 8	x0/y0, x1/y1, x2/y2, x3/y3, x4/y4, x5/y5
16-bits	1	6 to 24	x1:x0/y5:y4
16-bits	3	2 to 8	x1:x0/y1:y0,x3:x2/y3:y2, x5:x4/y5:y4
24-bits	1	4 to 16	x2:x1:x0/y5:y4:y3
24-bits	2	2 to 8	x2:x1:x0/y2:y1:y0, x5:x4:x3/y5:y4:y3

For a single precision configuration, the clock period for the enhanced cell is roughly the same as that of the basic cell from the previous section with additional delay for the additional combinational logic. However, for double or triple precision operation, the cell period must include the time to propagate the carry signal across the adders of one or two other cells, respectively. Thus, the cell has a different period for each of the three different precision configurations. In addition, double and triple precision require three and five clock cycles to generate the results for each tap. The same latency issues that were present for the basic cell still apply to the enhanced cell. Thus, the sample rate and latency for a particular chain of enhanced cells implementing a filter are given by,

$$SampleRate = \left(\lceil Taps/Cells \rceil \cdot Cycles(p) \cdot CellPeriod(p) \right)^{-1} \quad (9)$$

$$Latency = Cells\left(\lceil Taps/Cells \rceil \cdot Cycles(p) \cdot CellPeriod(p) \right) \quad (10)$$

where, $Cells \le Taps \le 4 \cdot Cells$, p is precision (8, 16 or 24), $Cycles(p)$ is 1 if p=8, 3 if p=16, and 5 if p=24, and $CellPeriod(p)$ is the cell period for precision p. A wide variety of possible sample rate and latency characteristics can be obtained from the same chain of cells.

The area of the enhanced cell has grown significantly from the basic cell due to the added complexity. However, all three of the basic FIR filter parameters can now be configured, allowing many possible FIR filters to be implemented with the same hardware.

5. DESIGN COMPARISONS

As mentioned in the introduction, the objective of the cell-based approach is to provide an FPGA architecture for FIR filtering, which allows basic filter design parameters to be reconfigured while performance and area efficiency fall between the two custom approaches: parallel and sequential. In the previous sections, four different approaches to implementing FIR filters have been discussed. In this section we examine the relative performance and area characteristics of the approaches across a range of filter lengths and determine if the objective has been met.

The data for each of the filter implementations in the following sections was generated using Mentor Graphics tools. After describing the hardware in VHDL, the circuit description was either synthesized or hand-designed, and a floorplan and layout were generated using automatic place and route and the Mentor Graphics 1.2um standard CMOS library. Although the 1.2um library is out of date by today's standards, it is adequate for our purposes, since we are only interested in *relative*, rather than absolute, performance and area characteristics.

The different approaches were implemented in the manner shown in the figures from previous sections. The parallel and sequential implementations were scaled-up from the circuits shown in the figures in order to implement filters with more than four taps. For all designs, products are generated using mixed-mode parallel array multipliers and sums were generated with carry lookahead adders. All other combinational logic was synthesized using Mentor Graphics tools. For each implementation, the area of the layout and the critical path delay including layout parasitics were measured. From the critical path measurement, the sample rate of the design was calculated. To simplify the comparisons, pin issues such as IO buffering were ignored.

5.1 Area Comparisons

Figure 10 shows the layout area in square microns versus the number of filter taps for single precision (8-bit) data and filter coefficients. Plots for parallel and sequential implementations are labeled "P" and "S", respectively. The enhanced cell approach is plotted for six different implementation sizes: 1, 2, 4, 6, 8, and 12 cells. Each of the cell implementations is labeled with the number of cells. For each of the implementations, four of the different filter sizes that can be configured (Taps=Cells, 2·Cells, 3·Cells, and 4·Cells) are plotted. For clarity, the basic cell approach is not included on the graph. However, the area of a basic cell implementation is roughly 65% of the area of an enhanced cell implementation with the same number of cells.

Recall that we are using the parallel and sequential implementations as guides to judge the efficiency of the cell based approach. The parallel implementations represent the upper guideline or least efficient use of area, while the sequential implementations represent the lower guideline or most efficient use of area. As can be seen from Figure 10, the cell implementations fall across the plot for the parallel implementations. For instance, if we compare the 12-cell curve to the parallel curve, we see that when each of the cells is configured for one tap, for a total of twelve taps, the area requirement is roughly twice that of the parallel implementation of a 12-tap FIR. However, if each cell is configured for four taps, for a total of 48 taps, then the

area requirement is roughly half that of the parallel 48-tap filter. Thus, the area efficiency of the cell based approach changes depending on the number of taps each cell is configured to implement.

Since two of the points for each of the six different sizes fall outside of the parallel curve, the enhanced cell based approach is not area efficient for single precision data and coefficients. However, this is expected since a significant amount of the hardware in the enhanced cell is devoted to allowing double or triple precision operation. For single precision, this hardware is not used and acts as overhead. Of course the basic cell implementation (not shown) compares much more favorably with the parallel approach since it does not contain the extra double and triple precision hardware.

Area comparisons for double precision (16-bit) data and filter coefficients are shown in Figure 11. Four enhanced cell implementation sizes are shown: 2, 4, 8, and 12. As with the previous figure, the area of the custom parallel and sequential implementations is plotted and labeled "P" and "S", respectively.

For double precision, the enhanced cell implementations fall almost entirely within the parallel and sequential guidelines. Recall that for double precision, two cells cooperate to implement between one and four taps. Thus the maximum number of taps for a particular number of cells used for double precision is half that of the same cells used for single precision. For the 12-cell implementation of a 6-tap filter, the area requirement shown in Figure 11 is roughly the same as the parallel approach and 4.4 times that of the sequential approach. If the same twelve cells are used to implement a 24-tap filter, the area requirement is a quarter of the parallel approach and only about 2.4 times that of the sequential approach.

For double precision, the enhanced cell approach is relatively area efficient, as most of the points for the four different sizes fall between the parallel and sequential guidelines. Indeed, for the 2 and 4-cell implementations, the area requirements are very similar to that of the sequential implementations.

Figure 12 shows area comparisons for triple precision (24-bit) data and filter coefficients. Three implementation sizes are shown for the enhanced cell approach: 3, 6, and 12. As can be seen from the figure, for triple precision, the enhanced cell approach is quite area efficient. All of the points for the cell approach fall within the parallel and sequential guidelines, with most points closer to the sequential approach. For the 12-cell implementation, when a 16-tap filter is implemented, the area requirements are roughly 0.23 and 1.9 times that of the parallel and sequential approaches, respectively. In the case of the 3-cell implementation, the area requirements are less than that of the sequential approach since the 3-cell approach implements a 24-bit multiplication using three 8-bit multipliers and the sequential approach uses 12-bit multipliers.

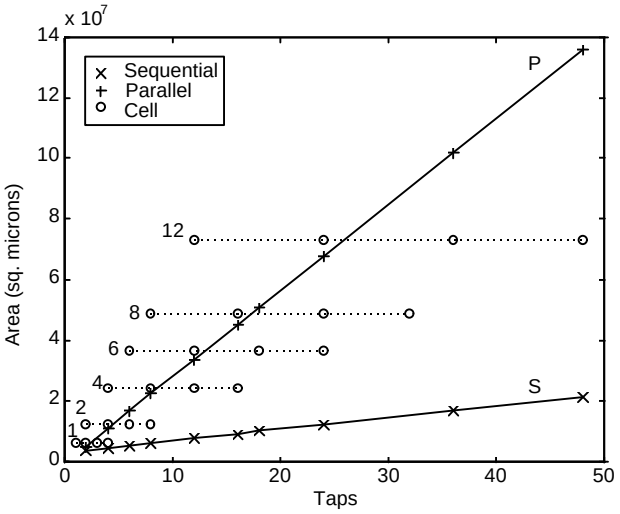

Figure 10. Area comparison for 8-bits.

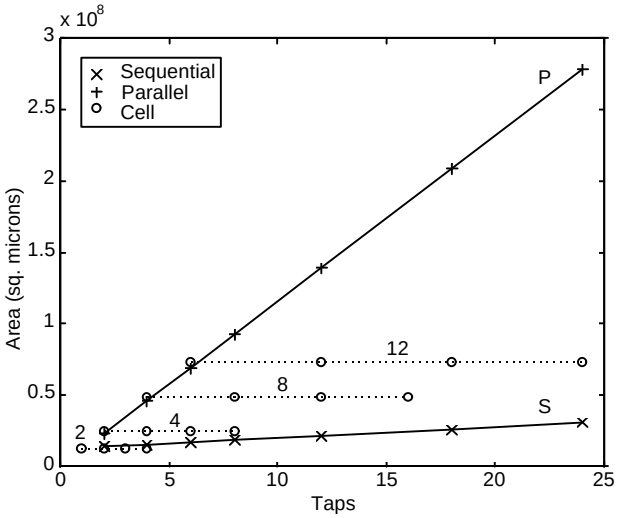

Figure 11. Area comparison for 16-bits.

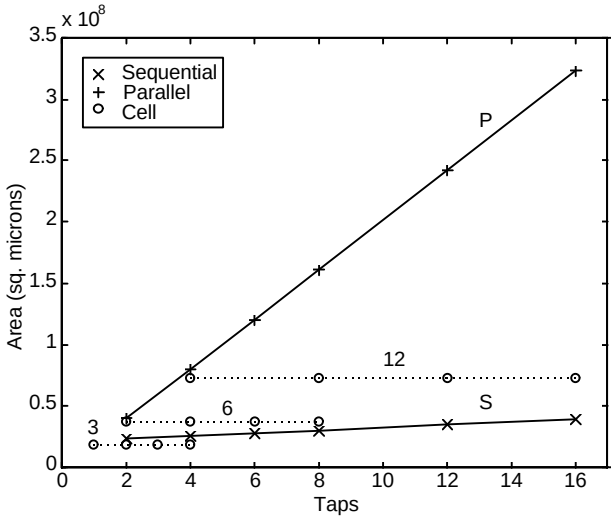

Figure 12. Area comparison for 24-bits.

Note that for the 12-cell implementation, for example, the same hardware was used for each of the three area comparisons. As the precision was increased from 8-bits to 24-bits, the hardware became more area efficient. Thus, depending on the precision and the number of taps, the same hardware can be configured for a variety of area efficiencies relative to a custom design. As we will see in the next section, this is also true for performance.

5.2 Performance comparisons

Similar to the previous section, the enhanced cell-based approach is compared to custom parallel and sequential implementations across a range of filter lengths. However, in this section we examine the relative performance of the enhanced cell. In particular, the sample rate, or the rate at which input samples may enter the filter, is considered. In this case, the parallel implementations represent the upper guideline on performance, while the sequential implementations represent the lower guideline on performance.

Figure 13 shows the sample rate in MHz versus the number of filter taps for single precision input data and filter coefficients. The same implementations that were shown in the area comparison of Figure 8 are also shown in Figure 13. From Figure 13, we see that the cell implementations fall across the plot for the parallel implementations, with many cell points above or near the parallel points. For example, for the 12-cell curve, when a 12-tap filter is implemented, the sample rate is 1.6 times that of the parallel approach. When a 48-tap filter is implemented using the same hardware, the sample rate is only about half that of the parallel approach.

Although for small numbers of cells (1 or 2) the performance is close to that of the sequential approach, as the number of cells is increased the performance becomes closer to that of the parallel approach. Thus, for single precision operation the performance of the enhanced cell is relatively close to that of the parallel approach. For the basic cell approach, which is not shown in Figure 13 for clarity, the relative performance is even better, since the basic cell has a sample rate of 1.8 times that of the enhanced cell.

Performance comparisons for double precision are shown in Figure 14. For double precision, the performance has moved closer to that of the sequential approach, with all of the cell points falling below the parallel points. For the 12-cell implementation of a 6-tap filter, the sample rate is 0.92 times that of the parallel approach. For a 24-tap filter, the sample rate is 0.32 times that of the parallel approach. Depending on the number of taps in the filter, the performance of the cell implementation can be configured to be close to either the parallel or sequential implementation.

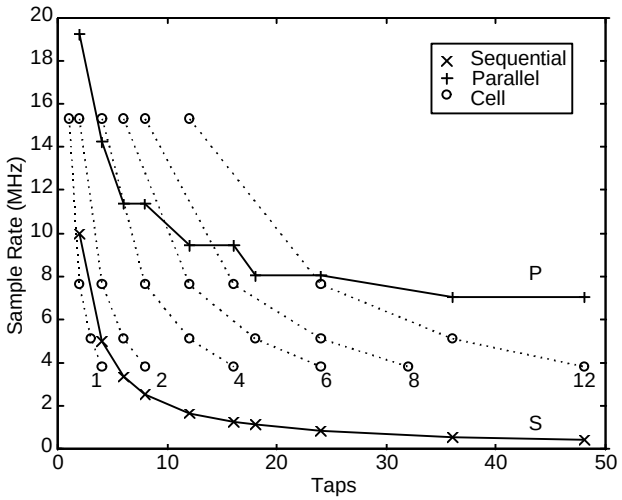

Figure 13. Performance comparison for 8-bits.

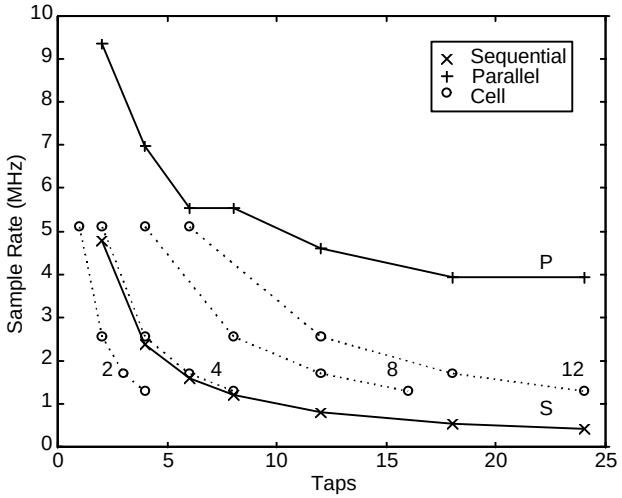

Figure 14. Performance comparison for 16-bits.

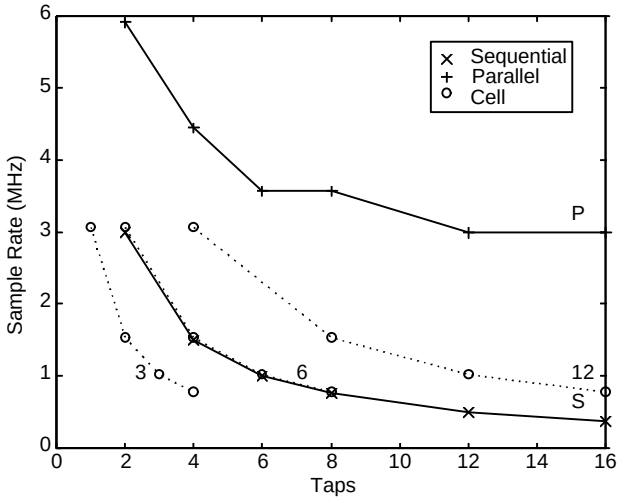

Figure 15. Performance comparison for 24-bits.

Triple precision operation results in the worst performance for cell implementations, as shown by Figure 15. The cell points have moved much closer to the sequential points,

with some sizes (3 and 6) being at or below the performance of the sequential implementation. As previously discussed, a cell operating in triple precision requires five clock cycles to generate the product and sum for one tap. In addition, carry signals must propagate across three parallel cells requiring a longer clock period. These two characteristics combine to produce the relatively low performance in the triple precision case.

As the precision of the input data and coefficients is decreased or the number of cells is increased, the performance of the cell approach moves toward that of the parallel approach. Thus, the cell approach has the best performance characteristics when the cells are operating in single precision mode. However, this is not surprising since in this case the cell approach also has the worst area efficiency. The cell approach allows one to easily trade area efficiency for performance by configuring the same hardware differently.

An excellent property of the cell approach is that the relative performance improves as the number of cells is increased. However, the trade-off for the performance improvement is an increase in latency. Although the latencies of the parallel and sequential approaches remain unity with an increase in the number of filter taps, each additional cell in a chain of cells adds additional latency. For very large filters used in latency-critical applications, the cell-based approach may be inappropriate. However, most applications do not require FIR filters of more than about 25 taps, for which the cell approach has a reasonable latency.

6. Conclusion

The enhanced cell approach can be arranged in a variety of topologies to provide an FPGA architecture which is specialized for FIR filtering. For different cell topologies, the filter coefficients, data and coefficient precision, filter length, sample rate, and area efficiency can each be reconfigured for a particular application using the same hardware. Depending on the particular configuration, performance and area efficiency can be similar to that of a custom parallel or sequential implementation. Thus, we have a coarse-grained FPGA architecture for FIR filtering which has the flexibility of a DSP processor with performance and area efficiency similar to that of an ASIC.

In future research we will extend the capabilities of the enhanced cell to include other common DSP functions, such as Infinite Impulse Response (IIR) filtering. In addition, we plan to develop heterogeneous architectures that contain cells which can implement different functions or numbers of taps. We also plan to consider grafting a number of cells onto a more traditional FPGA architecture to develop a more general purpose DSP FPGA.

7. ACKNOWLEDGMENTS

This research is sponsored by DARPA under grant number DAAH04-96-1-0222 and the Rockwell and Lucent Foundations.

8. REFERENCES

[1] Abnous, A. and Rabaey, J. Ultra-low-power domain-specific multimedia processors. in VLSI Signal Processing IX, IEEE Press, November 1996, 459-468.

[2] Arnold, J. M., Buell, D. A., and Davis, E. G. Splash 2. in Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1992, 316-324.

[3] Bergmann, N. W. and Mudge, J. C. Comparing the performance of FPGA-based custom computers with general-purpose computers for DSP applications. in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1994, 164-171.

[4] Bertin, P., Roncin, D., and Vuillemin, J. Programmable active memories: a performance assessment. in Research on Integrated Systems: Proceedings of the 1993 Symposium, 1993, 88-102.

[5] Oppenheim, A. V. and Schafer, R. W. Digital signal processing. Prentice-Hall, Englewood Cliffs, N.J., 1975.

[6] Petersen, R. and Hutchings, B. L. An assessment of the suitability of FPGA-based systems for use in digital signal processing. in 5th International Workshop on Field Programmable Logic and Applications, Oxford, England, August 1995, 293-302.

[7] Rabaey, J. Reconfigurable computing: the solution to low power programmable DSP. in Proceedings of the 1997 ICASSP Conference, Munich, April 1997.

[8] Weste, N. and Eshraghian, K. Principles of CMOS VLSI design, Addison-Wesley, 1993.