

Semi-Parallel FIR Filters

This chapter describes the implementation of semi-parallel or hardware-folded, full-precision FIR filters using the Virtex™-4 DSP48 slice. Because the Virtex-4 architecture is flexible, constructing FIR filters for specific application requirements is practical. Creating optimum filter structures of a semi-parallel nature saves resources and potential clock cycles. Therefore, optimum filter structures of a semi-parallel nature can be created without draining resources or clock cycles.

This chapter demonstrates two semi-parallel filter architectures: the four-multiplier FIR filter using distributed RAM and the three-multiplier FIR filter using block RAM. These filters illustrate how resources are saved by using available clock cycles and hardware-folding techniques. Reference design files are available for system generator in DSP, VHDL, and Verilog. The reference designs permit filter parameter changes including coefficients and the number of taps.

Overview

A large array of filtering techniques are available to signal processing engineers. A common filter implementation to exploit available clock cycles, while still achieving moderate to high sample rates, is the semi-parallel (also known as folded-hardware) FIR filter. In the past, this structure used the Virtex-II embedded multipliers and slice-based arithmetic logic. However, the Virtex-4 DSP48 slice introduces higher performance multiplication and arithmetic capabilities to enhance the use of semi-parallel FIR filters in FPGA-based DSP designs.

Semi-Parallel FIR Filter Structure

A wide variety of filter architectures are available to FPGA designers due to the *liquid hardware* nature of FPGAs. The type of architecture is typically determined by the amount of processing required in the number of available clock cycles. The two most important factors are:

- Sample Rate (F_s)
- Number of Coefficients (N)

As illustrated in Figure 6-1, as the sample rate increases and the number of coefficients increase, the architecture selected for a desired FIR filter becomes a more parallel structure involving more multiply-add elements. Chapter 4, “MAC FIR Filters” addresses the details of sequential processing FIR filters including the single and dual MAC FIR Filter. Chapter 5, “Parallel FIR Filters” investigates the polar extreme of the fully-parallel FIR filter required for the highest sample rate filters. This chapter examines the common scenario requiring multiple processing elements working

over numerous clock cycles to achieve the result. These techniques are often referred to as semi-parallel and are used to maximize efficiency of the filter (see Figure 6-1).

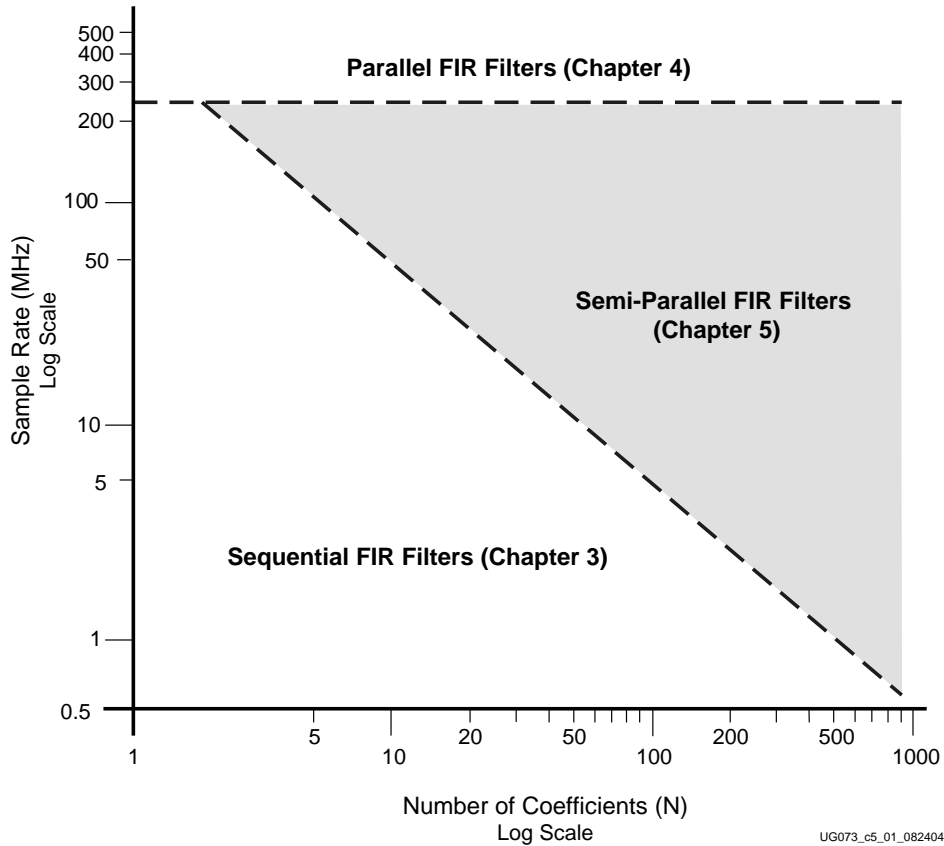


Figure 6-1: Selecting Filter Architectures

The semi-parallel FIR structure implements the general FIR filter equation of a summation of products defined as shown in Equation 6-1.

$$y_n = \sum_{i=0}^{N-1} x_{n-i} \cdot h_i \tag{Equation 6-1}$$

Here a set of N coefficients is multiplied by N respective time series data samples, and the results are summed together to form an individual result. The values of the coefficients determine the characteristics of the filter (for example, a low-pass filter).

Along with achievable clock speed and the number of coefficients (N), the number of multipliers (M) is also a factor in calculating semi-parallel FIR filter performance. The following equation demonstrates how the more multipliers used, the greater the achievable performance of the filter.

$$\text{Maximum Input Sample rate} = (\text{Clock speed} / \text{Number of Coefficients}) \times \text{Number of Multipliers}$$

The above equation is rearranged to determine how many multipliers to use for a particular semi-parallel architecture:

$$\text{Number of Multipliers} = (\text{Maximum Input Sample rate} \times \text{Number of Coefficients}) / \text{Clock speed}$$

The number of clock cycles between each result of the FIR filter is determined by the following equation:

$$\text{Number of Clock cycles per result} = \text{Number of Coefficients} / \text{Number of Multipliers}$$

The bit growth on the output of the filter is the same as for all FIR filters and is explained in “Bit Growth” in Chapter 4. The large 48-bit internal precision of the DSP48 slice means that little concern needs to be paid to the internal bit growth of the filter.

Four-Multiplier, Distributed-RAM-Based, Semi-Parallel FIR Filter

Once the required number of multipliers is determined, there is an extendable architecture using the DSP48 slice for use as the basis of the filter. This section assumes the specifications in Table 6-1 describe the filter implementation and its functions.

Table 6-1: Four-Multiplier, Semi-Parallel FIR Filter Specifications

Sampling Rate	112.5 MSPS
Number of Coefficients	16
Assumed Clock Speed	450 MHz
Input Data Width	18 Bits
Output Data Width	18 Bits
Number of Multipliers	4
Number of Clock Cycles Between Each Result	4

Figure 6-2 illustrates the main structure for the four-multiplier, semi-parallel FIR filter.

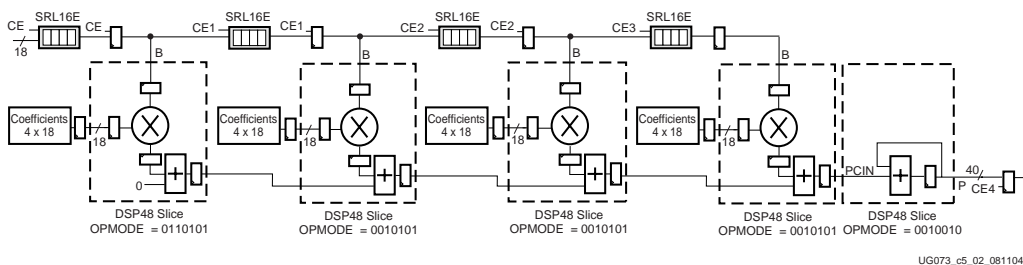
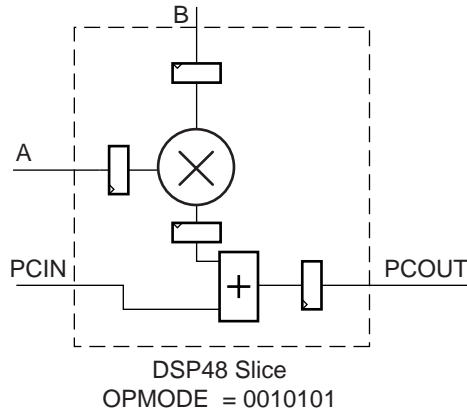


Figure 6-2: Four-Multiplier, Semi-Parallel FIR Filter in Accumulation Mode

The DSP48 slice arithmetic units are designed to be chained together easily and efficiently due to dedicated routing between slices. Figure 6-2 shows how the four DSP48 slice multiply-add elements are cascaded together to form the main part of the filter structure. Figure 6-3 provides a detailed view

of the main multiply-add elements. The two pipeline registers are used on the B input to compensate for the register on the output of the coefficient memory.



UG073_c5_03_081104

Figure 6-3: Detailed Diagram of a Single Multiply-Add Element

An extra DSP48 slice is required on the end to perform the accumulation of the partial results, thus creating the final result. A new result is created every four cycles. Every four cycles, the accumulation must be reset to the first partial value of the next result. As in the MAC FIR Filter, this reset (or load) is achieved by changing the OPMODE value of the DSP48 slice for a single cycle. OPMODE is changed from binary 0010010 to binary 0010000 (just a single bit change). At the same time, the capture register is also enabled, and the final result is stored on the output (see Figure 6-4).

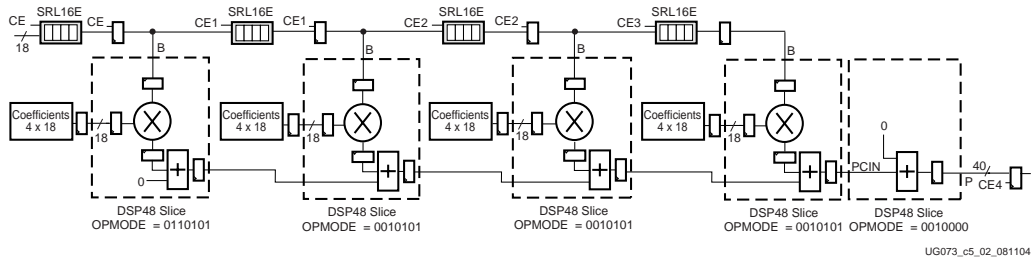


Figure 6-4: Four-Multiplier, Semi-Parallel FIR Filter at the Start of a New Result Cycle

Control logic is required to make this dynamic change occur. The specifics are detailed in “Control Logic and Address Sequencing,” page 90.

Data Memory Buffers

This example uses eight memories. Four SRL16Es are used as data buffers. Each SRL16E holds the four samples needed for the result. They are written to once every four cycles (the input data rate is 4x slower than the internal rate), and the shifting characteristic of the SRL16E is exploited to pass old

samples along the time series buffer. The extra register on the output of each data buffer is required to match up the data buffer pipeline with the extra delay caused by the adder chain. The extra register should not cost extra resources, because it is already present in the slice containing the SRL16E (see Figure 6-5).

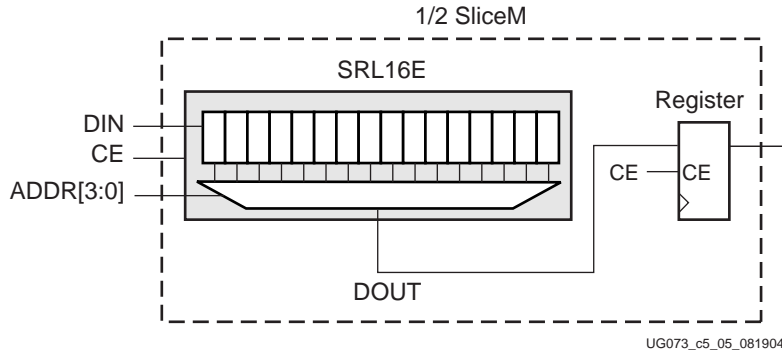


Figure 6-5: Single Bit of One Input Memory Buffer

As long as the depth does not exceed 16, the resources required for each of these input memory buffers is determined by the bit width of the input data (n). Therefore, $n/2$ SliceM is required for each memory buffer, leading to nine slices per buffer in this filter example. For depths up to 32, resources are a little more than doubled because two SRL16Es are needed, as well as an extra output multiplexer. For more information on SliceM, refer to the CLB section in the *Virtex-4 User Guide*.

Coefficient Memory

The coefficients are divided up into four groups of four. This arrangement is determined by dividing the total number of coefficients by the number of multipliers used in the implementation. In this example, if the total number of coefficients is 16, and the number of multipliers is four, four coefficients per memory are needed.

Note that filters with a total number of coefficients that are integer-divisible by the required number of multipliers are very desirable. System designers should take this into account when designing their filters to get the optimal filter specification for the implementation used. Otherwise, the coefficients will have to be padded with zeros to achieve a number of coefficients that are integer-divisible by the number of multipliers.

The coefficients are simply split into groups according to their order. The first four in the first memory, the second four in the second memory, and so on (see Figure 6-6).

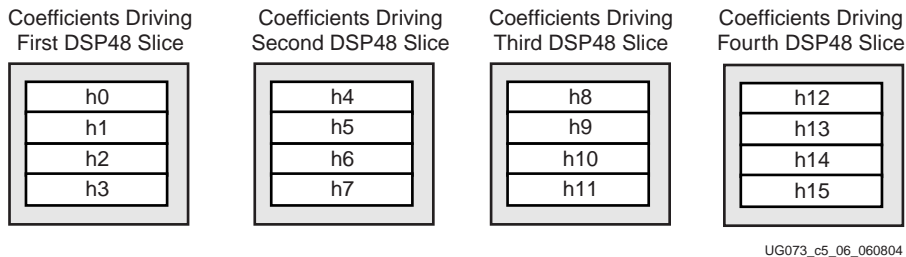


Figure 6-6: Coefficient Memory Arrangement

The adder chain architecture of the DSP48 slice means that each Multiply-Add cascade multiplication must be delayed by a single cycle so that the results synchronize appropriately when added together. This delay is achieved by addressing of the memories and is explained in “Control Logic and Address Sequencing”.

Distributed RAM (refer to Chapter 2, “XtremeDSP Design Considerations,” for detailed information on distributed RAMs) are used for the coefficient memories. The reason for their use is that it would be an extremely inefficient usage of the larger block RAMs, especially given their scarcity versus the smaller abundant distributed RAMs. The larger block RAM comes into play when the number of coefficients per memory starts to increase to the point where the cost in slice resources becomes significant (for example, greater than 64).

The total cost of the current example is 36 slices. The coefficient width is 18 bits, and distributed RAMs cost $n/2$ slices (that is, nine slices per memory and four memories). For larger distributed RAMs (larger than 16 elements), the size begins to increase as Write Enable (WE) control logic and an output multiplexer is needed. The distributed memory v7.0 in the CORE Generator system can be easily used to create these little distributed RAMs and get accurate size estimates.

Control Logic and Address Sequencing

The Control Logic and Address Sequencing is the most important and complicated aspect of semi-parallel FIR filters, and getting it right is crucial to the operation of the filter. The control logic is discussed in two separate sections:

- Memory Addressing
- Clock Enable Sequencing

Memory addressing must provide the necessary delay for each multiply-add element mentioned in “Coefficient Memory,” page 89, caused by the adder chain. This is not the case when using an adder tree; the DSP48 slice is most efficiently used in adder chains.

Figure 6-7 illustrates the control logic required to create the necessary memory addressing. The counter creates the fundamental zero through three count. This is then delayed by one cycle by the use of a register in the control path. Each successive delay is used to address both the coefficient memory and the data buffer of their respective multiply-add elements. A single delay for the second multiply-add element, two delays for the third multiply-add element, etc. Note that this is extensible control logic for M number of multipliers.

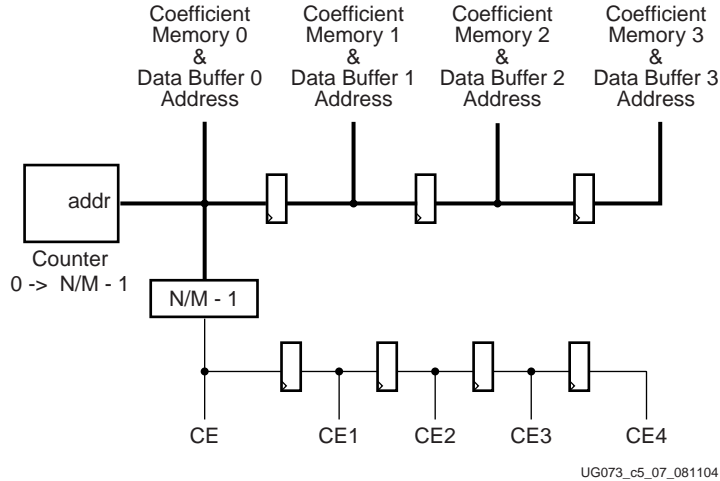


Figure 6-7: Control Logic for the Four-Multiplier, Semi-Parallel FIR Filter

Figure 6-7 also shows clock enable sequencing. A relational operator is required to determine when the count limited counter resets its count. This signal is High for one clock cycle every four cycles, to represent the input and output data rates. The Clock Enable signal is delayed by a single register just like the coefficient address, and each delayed version of the signal is tied to the respective section of the filter. Refer to Figure 6-2 to see the signal connections to the element. Figure 6-8 illustrates the control logic waveforms changing over time.

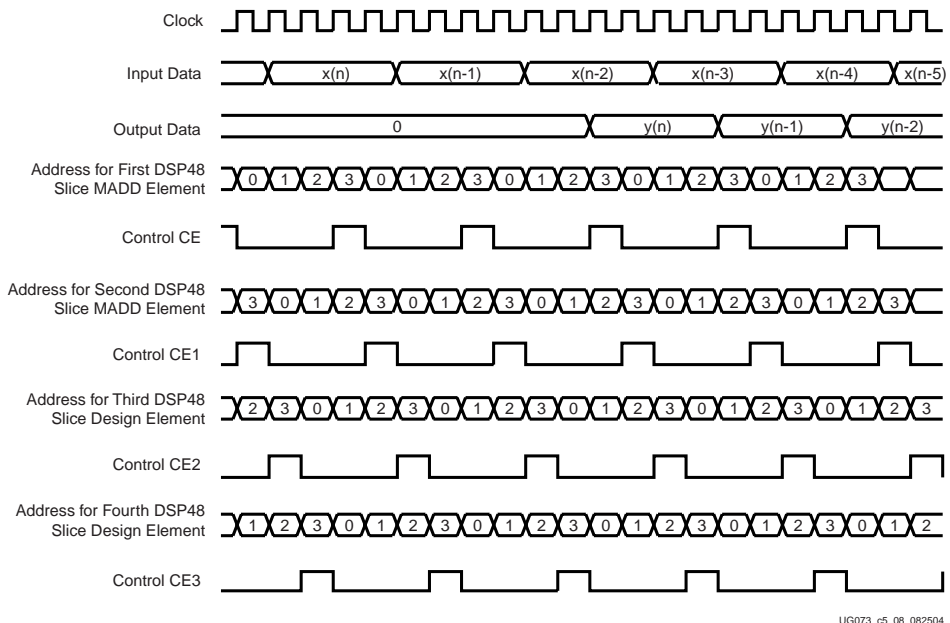


Figure 6-8: Control Waveforms for Semi-Parallel FIR Filters

Resource Utilization

Table 6-2 shows the resources used by a 16-tap, four-multiplier, distributed-RAM-based, semi-parallel FIR filter.

Table 6-2: Resource Utilization

Elements	Slices	DSP48 Slices
Multiply-Add		5
Input Data Buffers	36	
Coefficient Memories	36	
Capture Register	20	
Main Control Counter	2	
Relational Operator	1	
Multiply-Add Element Control	9 (3 per extra element)	
Total	104	5

Three-Multiplier, Block RAM-Based, Semi-Parallel FIR Filter

This section investigates a different filter structure, the three-multiplier, block RAM-based, semi-parallel FIR filter (see Figure 6-9).

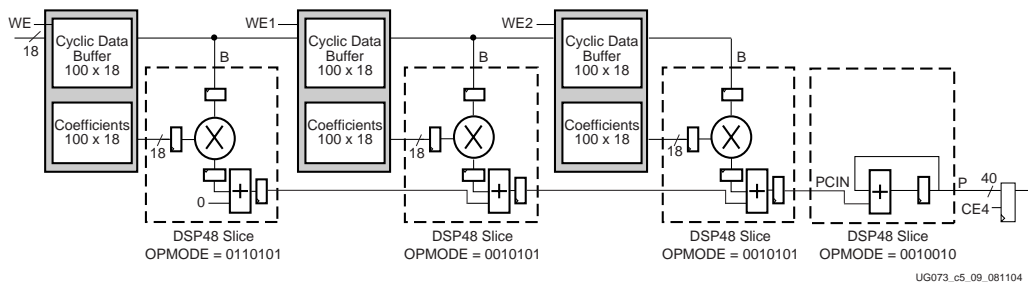


Figure 6-9: Three-Multiplier, Block-RAM-Based, Semi-Parallel FIR Filter

The decision to use this implementation is based on the filter specification. The filter specifications are described in Table 6-3.

Table 6-3: Three-Multiplier, Block RAM-Based, Semi-Parallel FIR Filter Specifications

Parameter	Value
Sampling Rate	4.5 MSPS
Number of Coefficients	300
Assumed Clock Speed	450 MHz
Input Data Width	18 Bits
Output Data Width	18 Bits

Table 6-3: Three-Multiplier, Block RAM-Based, Semi-Parallel FIR Filter Specifications

Parameter	Value
Number of Multipliers	3
Number of Clock Cycles Between Each Result	100

The structure is similar to the four-multiplier filter studied earlier. In this instance, the *lower sample rate* of the filter specification and the *larger number of taps* indicates that only three multipliers are required, each servicing 100 coefficients, leading to a new result yielded every 100 clock cycles.

Each memory buffer is required to hold 100 coefficients and also 100 input data history values. The dedicated Virtex-4 block RAM can be used in dual-port mode with a cyclic data buffer established in the first half of the memory to serve the shifting input data series.

Chapter 4, “MAC FIR Filters,” describes using these memories to store the input data series, the coefficients, and also the control logic required to make the cyclic RAM buffer operate. The rest of the control logic and data flow is identical to the first filter investigated except that only three multipliers are serviced, therefore, the control logic can be scaled back by one element. Also note that the WE signals are the inversion of their respective CE pair.

Table 6-4 shows the resource utilization for the 300-tap, three-multiplier, semi-parallel FIR filter.

Table 6-4: Resource Utilization

Elements	Slices	DSP48 Slices	Block RAMs
Multiply-Add		4	
Input Data Buffers and Coefficient Memories			3
Capture Register	20		
Main Control Counter	5		
Relational Operator	1		
Multiply-Add Element Control	12 (6 per extra element)		
Total	38	4	3

Other Semi-Parallel FIR Filter Structures

As with many DSP functions there are many different ways to implement a function. There is never one solution fitting all requirements for all specifications. For example, should distributed or block RAM be used for data storage? Should a systolic or a transposed implementation be used for a given filter? This chapter describes in detail the different techniques using single-rate FIR filters to get the maximum performance and low resource utilization using the Virtex-4 architecture.

This section introduces other possible semi-parallel FIR filter implementations and discusses the advantages and disadvantages of their structures.

Semi-Parallel, Transposed, Four-Multiplier FIR Filter

This structure is very different in nature to the main architecture discussed in this chapter (see Figure 6-10).

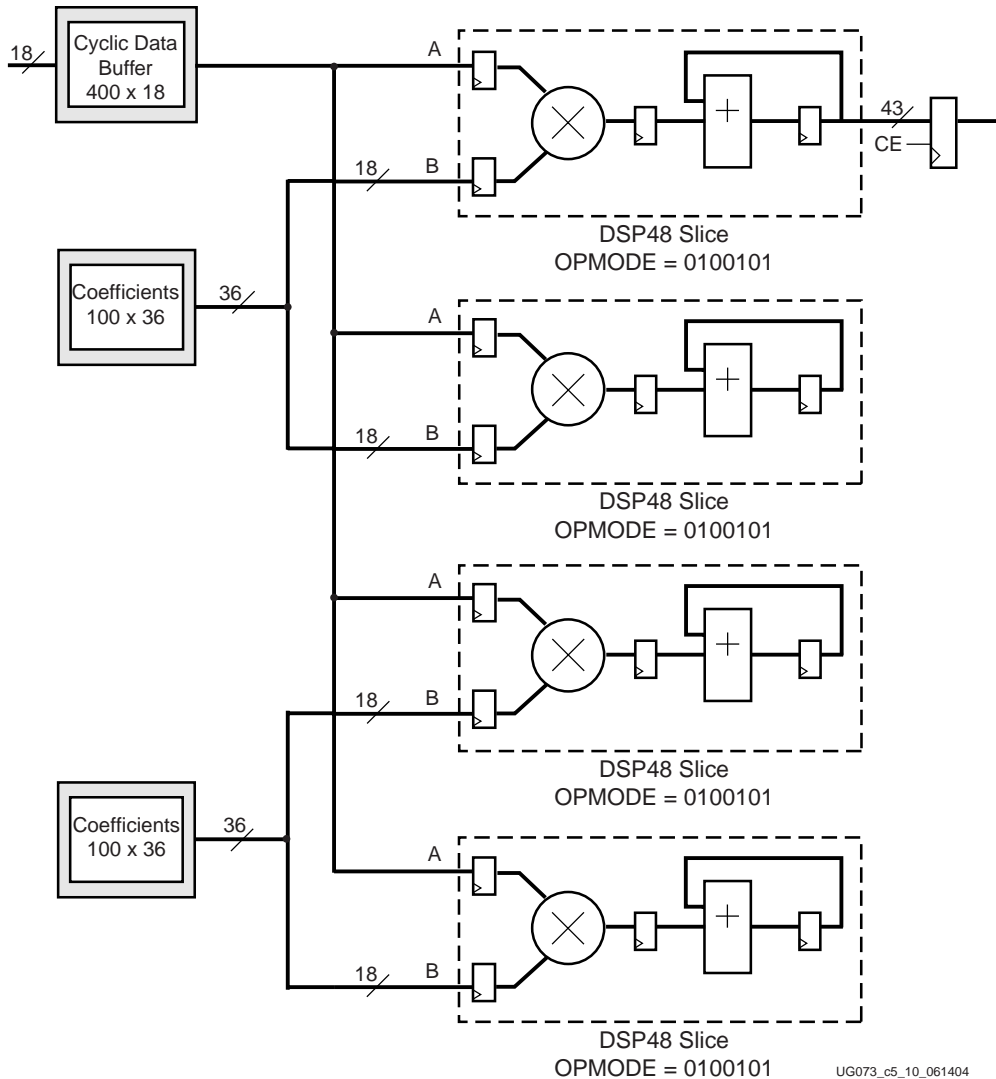


Figure 6-10: Semi-Parallel, Transposed FIR Filter

Only one data storage buffer is required, typically a block RAM. The data buffer output is also broadcast to all DSP48 slices. Each DSP48 slice works in accumulator mode until the last cycle of the calculation, when OPMODE changes to form an adder chain, and then passes the results to the next DSP48 slice. Actually, four results are being calculated at one time, and the completed result is output

from the last DSP48 slice. The previous elements are working on their respective parts of the next results.

Figure 6-11 shows the filter structure every time the DSP48 slice OPMODE is changed, which occurs once every result cycle.

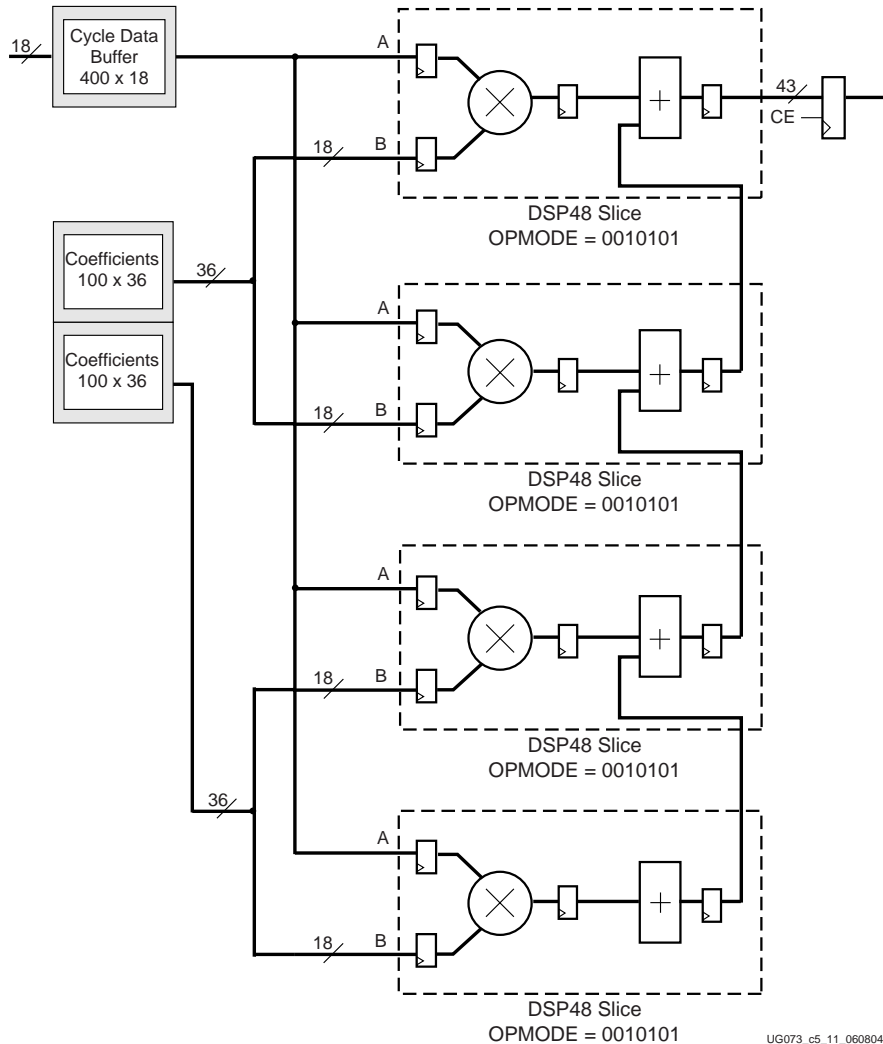


Figure 6-11: Semi-Parallel, Transposed FIR Filter (Combination of the Results)

Advantages and Disadvantages

The advantages to using the Semi-Parallel, Transposed FIR filter are:

- Lower resource utilization due to one less DSP48 slice required and a single input memory buffer.

- Low latency due to the transpose nature of the filter implementation is lower than the Systolic approach. The latency is equal to the size of one coefficient bank.
- The disadvantages to using the Semi-Parallel, Transposed FIR filter are:
- Lower performance due to the broadcast nature of the data buffer output can limit performance of the filter.
 - Control logic is more difficult to understand, but is still of a compact nature.

Rounding

The number of bits on the output of the filter is much larger than the input and must be reduced to a manageable width. The output can be truncated by simply selecting the MSBs required from the filter. However, truncation introduces an undesirable DC shift on the data set.

Due to the nature of two's complement numbers, negative numbers become more negative and positive numbers also become more negative. The DC shift can be improved with the use of symmetric rounding, where positive numbers are rounded up and negative numbers are rounded down. The rounding capability built into the DSP48 slice maintains performance and minimizes the use of FPGA fabric. This is ingrained in the DSP48 slice via the C input port and also the Carry-In port. Rounding is achieved in the following manner:

For positive numbers: Binary Data Value + 0.10000... and then truncate

For negative numbers: Binary Data Value + 0.01111... and then truncate

The actual implementation always adds 0.0111... to the data value using the C port input as in the negative case, and then adds the extra carry in required to adjust for positive numbers. Table 6-5 illustrates some examples of symmetric rounding.

Table 6-5: Symmetric Rounding Examples

Decimal Value	Binary Value	Add Round	Truncate: Finish	Rounded Value
2.4375	0010.0111	0010.1111	0010	2
2.5	0010.1000	0011.0000	0011	3
2.5625	0010.1001	0011.0001	0011	3
-2.4375	1101.1001	1110.0000	1110	-2
-2.5	1101.1000	1101.1111	1101	-3
-2.5625	1101.0111	1101.1110	1101	-3

In the instance of the semi-parallel FIR filter, an extra DSP48 slice is required to perform the rounding functionality. It cannot be ingrained into the final accumulator because the rounding cannot be done on the final result. If the C input is used and the accumulator is put into three-input add mode, then rounding is performed on the partial result. The more multipliers in the filter, the worse the rounding performance because even fewer inner products are included in the result. An extra DSP48 slice is required to perform the rounding.

Due to the finite nature of the DSP48 slices, it is recommended that the symmetric rounder be actually implemented in the fabric outside of the slices. The function is small and does not have to run at a high frequency because the results are running at the much slower input data rate.

Performance

It does not make sense to compare the performance of the semi-parallel FIR filter in a Virtex-4 device with Virtex-II Pro results because completely different techniques are used to build the filters. As a general statement though, Virtex-4 devices improve the speed of the design, shrink the area, and reduce power drawn by the filter. All designs assume 18-bit data and 18-bit coefficient widths.

Table 6-6 through Table 6-8 compare the specifications for three filters.

Table 6-6: 4-Multiplier, Memory-Based, Semi-Parallel FIR Filter Specifications (16-Tap Symmetric)

Parameter	Specification
Size	94 slices, 5 DSP48 slices
Performance	458 MHz clock speed, 114.5 MSPS
Power	TBD Watt

Table 6-7: 3-Multiplier, Block-RAM-Based, Semi-Parallel FIR Filter Specifications (300-Tap Symmetric)

Parameter	Specification
Size	38 slices, 4 DSP48 slices, 4 block RAMs
Performance	450 MHz clock speed, 4.5 MSPS
Power	TBD Watt

Table 6-8: 4-Multiplier, Block-RAM-Based, Semi-Parallel Transposed FIR Filter Specifications (400-Tap Symmetric)

Parameter	Specification
Size	46 slices, 4 DSP48 slices, 2 block RAMs
Performance	450 MHz clock speed, 4.5 MSPS
Power	TBD Watt

