CoreFIR Finite Impulse Response (FIR) Filter Generator

DirectCore

Product Summary

Intended Use

- Finite Impulse Response (FIR) Filter for Actel FPGAs

Key Features

- Core Generator
  - Executable File Outputs Run-Time Library (RTL) Code and Testbench Based on Input Parameters
  - Self-Checking – Executable Tests Generated Output against Algorithm
- Distributed Arithmetic (DA) Algorithm
  - Multiplier-Free Computation
  - Low Cost
  - Optimized for Actel FPGAs
- Folding Architecture to Minimize Design Size
  - Serialized Computation when System Clock Rate is Faster than the Data Sample Rate
- Efficient Structure Using Embedded RAMs
  - Lookup Tables Utilize Embedded RAMs
- On-Chip DA Lookup Table Generator for FPGA with Embedded RAMs
- Embedded RAMs Initialized as DA Lookup Table
- DA Lookup Table ROM Synthesis for FPGA without Embedded RAMs
- Multiple DA lookup Tables to Split Large Number of Taps
- Actel FPGA-Optimized RTL Code
- Supports 2 to 128 Taps
- 1- to 32-Bit Input Data and Coefficient Precision

Supported Families

- Fusion
- ProASIC3/E
- ProASIC$^{\underline{PLUS}}$ ®
- Axcelerator®
- RTAX-S
- SX-A
- RTSX-S

Core Deliverables

- Evaluation Version
  - RTL Code of a Sample Filter and Compiled RTL Simulation Model Fully Supported in the Actel Libero® Integrated Design Environment (IDE)
- RTL Version
  - A Microsoft Windows® Binary Executable of the CoreFIR Generator
  - VHDL FIR Module
  - VHDL Test Harness

Synthesis and Simulation Support

- Synthesis: Synplicity®, Synopsys® (Design Compiler®/FPGA Compiler™/FPGA Express™), Exemplar™
- Simulation: OVI-Compliant Verilog Simulators and Vital-Compliant VHDL Simulators

Contents

Device Utilization and Performance

The CoreFIR generates FIR filters with many configurations. Table 1 provides the typical utilization and performance data for the generated FIR filters implemented with the configurations listed in Table 2 on page 3. Refer to Table 2 on page 3 for the Configuration column in Table 1.

Table 1 • **CoreFIR Device Utilization and Performance**

Family	Configuration	Cells or Tiles			RAM Blocks	Utilization		Performance MHz
		Combinatorial	Sequential	Total		Device	Total	
Fusion	1	454	129	583	0	AFS060	38%	69
Fusion	2	1410	375	1784	0	AFS250	29%	56
Fusion	3	3080	679	3759	0	AFS250	61%	52
Fusion	4	5511	935	6446	8	AFS600	47%	45
Fusion	5	7089	1708	8797	0	AFS600	64%	40
Fusion	6	24356	3718	28074	45	AFS1500	73%	31
ProASIC3	1	454	129	583	0	A3P060	38%	69
ProASIC3	2	1410	375	1785	0	A3P125	58%	56
ProASIC3	3	3080	679	3759	0	A3P1000	15%	52
ProASIC3	4	5511	935	6446	8	A3P1000	26%	45
ProASIC3	5	7089	1708	8797	0	A3P1000	36%	40
ProASIC3	6	24356	3718	28074	45	A3P1500	73%	31
ProASIC^{PLUS}	1	558	116	674	0	APA075	22%	29
ProASIC^{PLUS}	2	2054	427	2481	0	APA150	40%	19
ProASIC^{PLUS}	3	3540	661	4201	0	APA1000	8%	19
ProASIC^{PLUS}	4	6391	872	7271	8	APA1000	13%	17
ProASIC^{PLUS}	5	8775	1606	10381	0	APA750	32%	13
Axcelerator	1	229	148	377	0	AX125	19%	174
Axcelerator	2	693	478	1171	0	AX250	28%	110
Axcelerator	3	1231	719	1950	0	AX250	46%	111
Axcelerator	4	2249	852	3101	4	AX500	38%	74
Axcelerator	5	3129	1704	4833	0	AX1000	27%	73
Axcelerator	6	9132	3355	12487	32	AX2000	39%	46
RTAX-S	1	229	148	377	0	RTAX1000S	2%	114
RTAX-S	2	693	478	1171	0	RTAX1000S	6%	76
RTAX-S	3	1231	719	1950	0	RTAX1000S	11%	66
RTAX-S	4	2249	852	3101	4	RTAX1000S	17%	41
RTAX-S	5	3129	1704	4833	0	RTAX1000S	27%	45
RTAX-S	6	9132	3355	12487	32	RTAX2000S	39%	29
SX-A	1	386	159	545	0	A54SX16A	38%	112
SX-A	2	1115	480	1595	0	A54SX72A	26%	64
SX-A	3	1831	727	2558	0	A54SX72A	42%	63
RTSX-S	1	381	159	540	0	RT54SX32S	19%	52
RTSX-S	2	1115	480	1595	0	RT54SX72S	26%	36
RTSX-S	3	1831	727	2558	0	RT54SX72S	42%	36

Notes:

1. The data above are obtained by typical synthesis and place-and-route methods. Other core parameters can result in different utilization and performance.
2. Cell (tile) count may vary depending on the actual coefficient values.

Table 2 • **Test Configurations**

Configuration	nbits_input	nbits_coef	ntaps	fpga_family	coef_fixed
1	8	16	8	All	1
2	16	16	16	All	1
3	12	15	32	All	1
4	12	15	32	AX, RTAX-S, APA	0
5	16	15	64	All	1
6	16	16	128	AX, RTAX-S, APA	0

FIR Filter Using Distributed Arithmetic Algorithm

Distributed Arithmetic Algorithm Overview

FIR filters are used in applications that require exact linear phase response. Typical applications for a FIR filter include: image processing, digital audio, digital communication, and biomedical signal processing. A FIR filter is defined in EQ 1:

$$y[n] = \sum_{0}^{ntaps-1} c[n] \times x[n]$$

EQ 1

where:

$c[n] = h[ntaps - n - 1]$

and h is the impulse response. The term *ntaps* is short for *number of taps*.

In summary, the direct computation for one point of FIR requires:

ntaps multiplications + (*ntaps*-1) additions.

Distributed Arithmetic (DA) is a well-known method for eliminating resources in multiply-and-accumulate structures (MACs) implementing digital signal processing (DSP) functions. DA trades memory for combinatory elements, resulting in an efficient implementation in FPGAs. Another feature of DA is its easy serialization of the input, which further reduces the cost of operation when FIR data rate is low compared to the system clock, a common scenario in FIR applications.

The input of a FIR can be expressed in the composition of its bits, as shown in EQ 2:

$$x[n] = \sum_{0}^{nbits\_in-1} x[n][b] \times 2^b$$

EQ 2

where *x*[*n*][*b*] is the b^{th} bit of *x*[*n*] and *nbits_in* is the number of bits of input. The resulting output of the FIR filter is shown in EQ 3:

$$y[n] = \sum_{0}^{ntaps-1} c[n]x[n] = \sum_{0}^{ntaps-1} c[n] \sum_{0}^{nbits\_in-1} x[n][b]2^b$$

EQ 3

Changing the summation order gives the results shown in EQ 4:

$$y[n] = \sum_{0}^{nbits\_in-1} 2^b \sum_{0}^{ntaps-1} c[n]\ x[n][b] = \sum_{0}^{nbits\_in-1} 2^b T(X[b])$$

EQ 4

where: $T(X[b]) = \sum_{0}^{ntaps-1} c[n]\ x[n][b]$ and $X[b]$ is a collection of the b^{th} bits of *ntaps* different taps.

Note that the $x[n][b]$ can only be 0 or 1. There are 2^{ntaps} different values of *T*. If *T* is pre-calculated and stored inside a RAM or ROM, the FIR computation becomes *nbits_in* table lookup operations using $x[b]$ and *nbits_in–1* additions. Multiplication operations are eliminated.

In summary, the FIR computation using DA for one point of FIR requires:

nbits_in table lookups + *(nbits_in-1)* additions.

The cost to eliminate multiplication is a memory block to store 2^{ntaps} pre-computed values.

The serialization of table lookup and addition is possible because table *T* is the same for each *b*. If one table lookup and one addition can be finished in one cycle, the total computation will finish in *b* cycles. The serialization of the FIR introduces further opportunity to reduce the size of the design, which is the key to an efficient FPGA design.

Example Design of a FIR Filter Using DA

An example of a FIR with four taps (*ntaps* = 4) and four bits for inputs (*nbits_in* = 4) is shown in Figure 1.

The expression $x[n][b]$ represents the b^{th} bit of input $x[n]$. In the example, in the first cycle, all 0^{th} bits of input $x[n]$ to $x[n-3]$ are fed into the lookup table as an input address; in the second cycle, all 1st bits of inputs input $x[n]$ to $x[n-3]$ are fed into the lookup table; in the third cycle, all 2nd bits of inputs input $x[n]$ to $x[n-3]$ are fed into the lookup table; and in the fourth cycle, all 3rd bits of inputs input $x[n]$ to $x[n-3]$ are fed into the lookup table. The shifter shifts the outputs of the lookup table for the inputs of the adder, which accumulates for the final result.

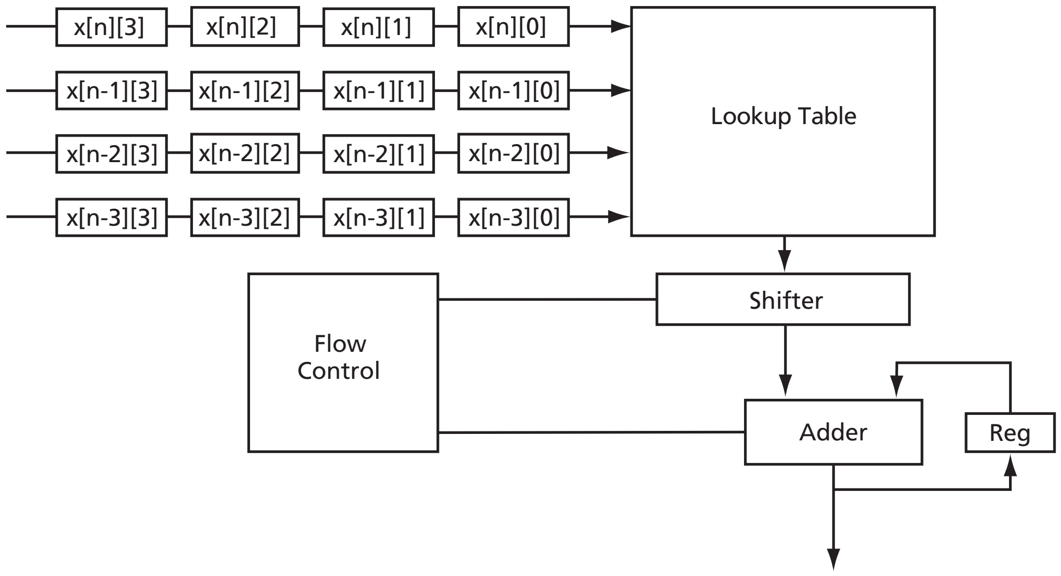


Figure 1 • **Example Implementation of a Bit-Serialized FIR Using DA**

The serialized DA implementation in Figure 1 uses a table lookup with 16 words, and takes four clock cycles to finish one FIR point.

Storage and Large Number of Taps

As seen in the previous section, the size of the lookup table is 2^{ntaps}, which is exponentially increased with more *ntaps*. A design with a large number of taps needs to have several lookup tables. Let *ntaps* = *p* × *q*. If we split taps into *p* groups, each group has *q* taps. Then the FIR becomes as shown in EQ 5:

$$y[n] = \sum_{0}^{nbits\_in-1} 2^b \sum_{0}^{n=ntaps-1} c[n]\ x[n][b] = \sum_{0}^{nbits\_in-1} 2^b \sum_{0}^{n=pq-1} c[n]\ x[n][b]$$

EQ 5

By splitting *ntaps* into two level summations, we have the result shown in EQ 6:

$$y[n] = \sum_{0}^{nbits\_in-1} 2^b \sum_{0}^{i=p\text{-}1} \sum_{0}^{j=q\text{-}1} c[iq+j]\ x[iq+j][b]$$

EQ 6

Refer to "FIR Filter with Large Number of Taps" on page 8 for further information.

General Description

The CoreFIR is an Actel FPGA-optimized RTL generator that produces a finite impulse response filter. It implements the DA algorithm to eliminate multiplication for faster and smaller designs. The CoreFIR is a generator which utilizes Actel FPGA's embedded RAM blocks as DA lookup tables (when available) to further reduce the size of the design. The generator also reads the user system clock rate and data sample rate to explore using a folding or serial architecture to further reduce size, especially when the system clock rate is much greater than the data sampling rate. The generator automatically switches to the use of multiple DA lookup tables when the requested FIR filter has a large number of taps. Figure 2 shows the functional block diagram of a generated FIR filter design. More complex designs may contain multiple lookup tables, accumulators, or control sections.

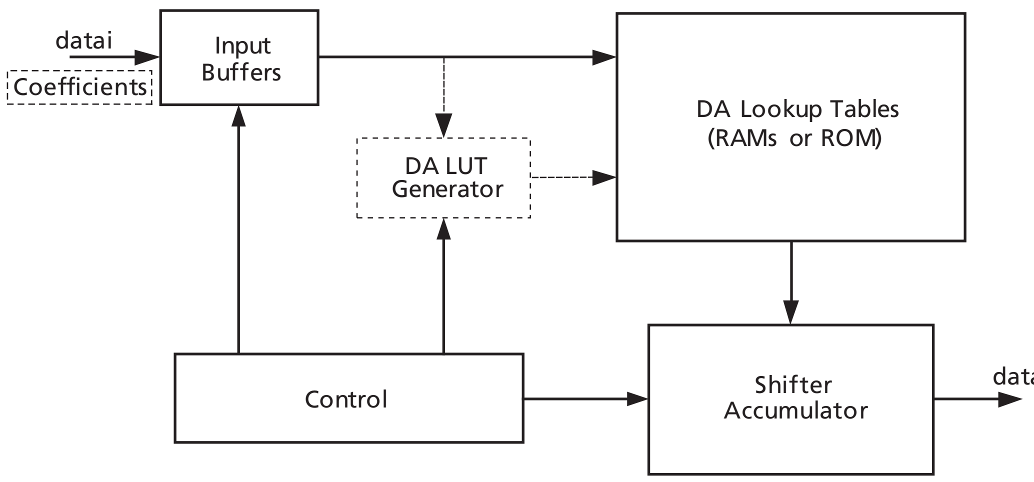


Figure 2 • **Functional Block Diagram**

Functional Block Description

The functional blocks shown in Figure 2 illustrate the architecture of the generated FIR filter using the DA algorithm.

Input Buffers

The Input Buffers block stores the input data which contains *ntaps* data points, where *ntaps* is the number of taps of the FIR filter. The Input Buffers block also circulates the data bits to address the DA Lookup Tables (LUTs) required by the DA algorithm. An optional function of the input buffers block is to share its storage with the DA LUT generator. The coefficients used in computing the LUT content can be stored in the input buffers when a design uses the embedded RAM blocks for the LUTs.

DA Lookup Tables (LUTs)

The DA LUTs store the LUT contents for the distributed algorithm. The generator implements the DA LUTs in two ways: (1) synthesized ROM using FPGA cells; and (2) embedded RAM blocks supported by the on-chip DA LUT Generator. The first method is for an FPGA without embedded RAM blocks, intended primarily for a small FIR filter. The latter is for an FPGA with embedded RAM blocks. FIR filters with a large number of taps may require multiple LUTs.

Shifter and Accumulator

The Shifter and Accumulator perform additions with LUT outputs and the alignments of LUT outputs required by the DA algorithm. Multiple accumulators and shifters may be needed to implement a FIR filter with a large number of taps.

DA LUT Generator

The DA LUT Generator computes the LUT contents required by the distributed arithmetic algorithm. It reads the coefficients from the Input Buffers block and writes the LUT words into the embedded RAM blocks. These blocks are available only for designs that use embedded RAM blocks as LUTs. The DA LUT Generator produces LUT contents for multiple LUTs when implementing a FIR filter with variable coefficients. Refer to "DA LUT Generation" on page 10 and "I/O Timing Diagram of LUT Initialization" on page 11 for detailed information on initialization of the DA LUT.

Control

The state machine inside the Control block controls the operations of all other blocks. It controls the input buffers to ensure they operate based on the specified system clock rate and sample rate, monitors input enable and coefficient input enable, and circulates input data bits to address the DA LUTs. It also controls the shifters and accumulators to ensure they operate based on the requested FIR configuration and DA algorithm. The Control block coordinates the initialization of the LUTs by the DA LUT generator when using embedded RAMs. The Control logic is designed to support folding or serialization of computation when the system clock rate is substantially higher than the data sampling rate.

I/O Signal Description

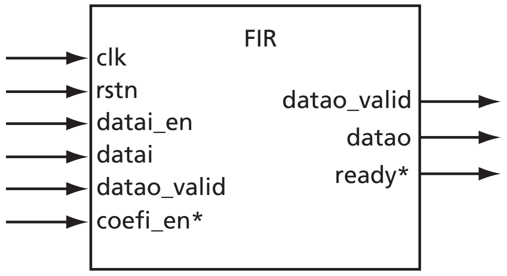
The FIR filter generated by the CoreFIR Generator consists of the I/O signals defined in Table 3 (see Figure 3 on page 7).

Table 3 • **I/O Signal Description**

I/O Signal	Direction	Width	Polarity	Description
clk	Input	1	N/A	Master clock, positive edge
rstn	Input	1	Active low	Master reset, asynchronous
datai_en	Input	1	Active high	Input data enable
datai[1]	Input	nbits_in[1]	N/A	Input data or coefficients[1]
datao_valid	Output	1	Active high	Output data valid
datao	Output	nbits_out[3]	N/A	Output data
coefi_en[2]	Input	1	Active high	Coefficient input enable
ready[2]	Output	1	Active high	Ready to input datai

Notes:

1. Input *datai* is also the input for coefficients for design using embedded RAMs as DA LUTs. In this case the width can be the maximum of *nbits_in* and *nbits_coef*.
2. Ports *coefi_en* and *ready* are only available when *coef_fixed* = 0.
3. Refer to "Number of Bits of Output (nbits_out)" on page 8 for details.
4. Refer to Table 4 for *nbits_in* and *nbits_coef*.



Note: ***coefi_en** and **ready** are available when coef_fixed = 0.

Figure 3 • **I/O Signals**

CoreFIR Generator Parameters

The CoreFIR generates the RTL code for FIR filters with a variety of parameters. These parameters include generic FIR parameters such as number of taps, number of input's bits, number of coefficients' bits, and data type, as well as implementation parameters such as FPGA family, use embedded RAMs, system clock rate, and data sampling rate. The CoreFIR supports the variations specified in Table 4.

Table 4 • **CoreFIR Generator Configuration Parameters**

		Recommended Selection		
Parameter Name	**Description**	**AX**	**APA**	**SX-A**
module_name	Name of generated module	–	–	–
nbits_input	Number of input's bits of data	2 – 24	2 – 24	2 – 24
nbits_coef	Number of coefficients' bits of data	2 – 24	2 – 24	2 – 24
ntaps	Number of taps	2 – 128	2 – 64	2 – 32
tap	Array of coefficients	–	–	–
data_signed	Data type: 0 = unsigned, 1 = signed	0,1	0,1	0,1
fpga_family	FPGA family	ax	apa	sxa
coef_fixed	0 = filter with configurable coefficients 1= filter with fixed coefficients	–	–	–
sys_clk_frq	Input clock frequency	–	–	–
sample_ratio	Sampling rate = sys_clk_frq/sample_ratio	≥ nbits_in	≥ nbits_in	≥ nbits_in
Module_lang	Reserved. VHDL only.	VHDL	VHDL	VHDL

Refer to "Appendix I: Sample Configuration File" on page 12 for a sample usage of the parameters shown in Table 4 in a configuration file for the CoreFIR Generator. Detailed discussions about these parameters are in the sections of this datasheet that follow.

Number of Taps (*ntaps*)

The FIR generator supports the number of taps specified by the device families in Table 4. The variable *ntaps* in the configuration file specifies the setting of this parameter. Refer to "Appendix I: Sample Configuration File" on page 12 for details.

Number of Bits of Inputs (*nbits_in*) and Coefficients (*nbits_coef*)

The FIR Generator supports the number of bits of inputs and coefficients for the device families specified in Table 4. These parameters are set with the variables *nbits_in* and *nbits_coef* in the configuration file. Refer to "Appendix I: Sample Configuration File" on page 12 for details.

Number of Bits of Output (*nbits_out*)

The FIR Generator supports only full precision computation. Thus, the number of bits of output is determined by the number of input's and coefficients' bits for the device family as specified in Table 4 on page 7. The number of bits of output are specified by EQ 7:

$$nbits\_out = nbits\_in + nbits\_coef + ceil(log_2(ntaps)) - 1 \qquad EQ\ 7$$

where *ceil* is the ceiling function of a floating point data.

Asymmetric FIR and Symmetric FIR

The FIR generator supports an asymmetric FIR filter only. Symmetric FIR filters will be supported in future releases.

Embedded RAM as LUTs (*coef_fixed*)

The FIR Generator utilizes a switch that determines whether to implement DA LUTs by embedded RAM blocks or by synthesized ROM using FPGA cells. The LUTs are implemented by synthesized ROM using FPGA cells when *coef_fixed* is equal to 1. The LUTs are implemented by embedded RAM blocks available for Axcelerator, ProASIC$^{\underline{PLUS}}$, and ProASIC3 devices when *coef_fixed* is equal to 0. This setting may be set to 1 for a filter design with fixed coefficients for an FPGA device with embedded RAM such as AX, RTAX-S, APA, and PA3, since the overhead of the DA LUT Generator overrides the benefits of using an embedded RAM block as a LUT. The *coef_fixed* configuration parameter is valid only when the configuration parameter *fpga_family* is set to *ax*, *apa*, or *pa3*. Refer to Table 4 on page 7 and "Appendix I: Sample Configuration File" on page 12 for details.

Signed/Unsigned Inputs and Coefficients (*data_signed*)

The FIR Generator supports signed or unsigned operations. The generator supports two cases: both input and coefficient are unsigned, or both input and coefficient are signed. It supports an unsigned implementation when the configuration parameter *data_signed* is equal to 0, and a signed implementation when the configuration parameter *data_signed* is equal to 1. Refer to Table 4 on page 7 and "Appendix I: Sample Configuration File" on page 12 for details.

System Clock Frequency (*sys_clk_frq*)

The FIR Generator reads in the system clock frequency via configuration parameter *sys_clk_frq*. The generated testbench assigns this frequency to its clock generation. The generated design runs at this frequency inside the test bench. The configuration parameter should be specified with the unit of MHz. Refer to Table 4 on page 7 and "Appendix I: Sample Configuration File" on page 12 for details.

Sample Ratio (*sample_ratio*)

The FIR Generator supports a configuration parameter, *sample_ratio*, which specifies the sampling rate against the system clock frequency. It defines that the data sampling rate is equal to *sys_clk_frq/sample_ratio*. This parameter provides guidance to implement a folding architecture to reduce the size of the design. The configuration parameter *sample_ratio* can only be a positive integer greater than 1. Refer to Table 4 on page 7 and "Appendix I: Sample Configuration File" on page 12 for details.

Module Name (*module_name*)

The FIR Generator supports a configuration parameter, *module_name*, that specifies the name of the generated module. The generated testbench has the name <module_name>_tb. Refer to Table 4 on page 7 and "Appendix I: Sample Configuration File" on page 12 for details.

FPGA Family (*fpga_family*)

The FIR Generator supports a configuration parameter, *fpga_family*, that specifies the targeted Actel FPGA device family. The options are *ax*, *apa*, *pa3*, and *sxa*. The option for RTAX-S is *ax*. The option for RTSX-S is *sxa*. Refer to Table 4 on page 7 and "Appendix I: Sample Configuration File" on page 12 for details.

Architecture Variations

The DA algorithm for FIR provides an excellent solution, but also introduces many variations on the design architecture due to limitations of the FPGA resource.

FIR Filter with Large Number of Taps

As illustrated in section "FIR Filter Using Distributed Arithmetic Algorithm" on page 3, the number of words of the DA LUT is 2^{ntaps}, which is exponentially increased with ntaps. A LUT splitting method, as defined in "Storage and Large Number of Taps" on page 5, effectively reduces the memory usage. The CoreFIR Generator utilizes this method to reduce the memory usage. It usually splits the coefficients into eight or nine taps for each LUT when embedded RAM blocks are available.

An example of the split lookup table implementation of a FIR with eight taps (*ntaps* = 8) and four bits for inputs (*nbits_in* = 4) is shown in Figure 4. In the example, eight taps have been split into two groups. Each has four taps, and each group addresses separate lookup tables. This differs from the case in Figure 1 on page 4, which only has one LUT.

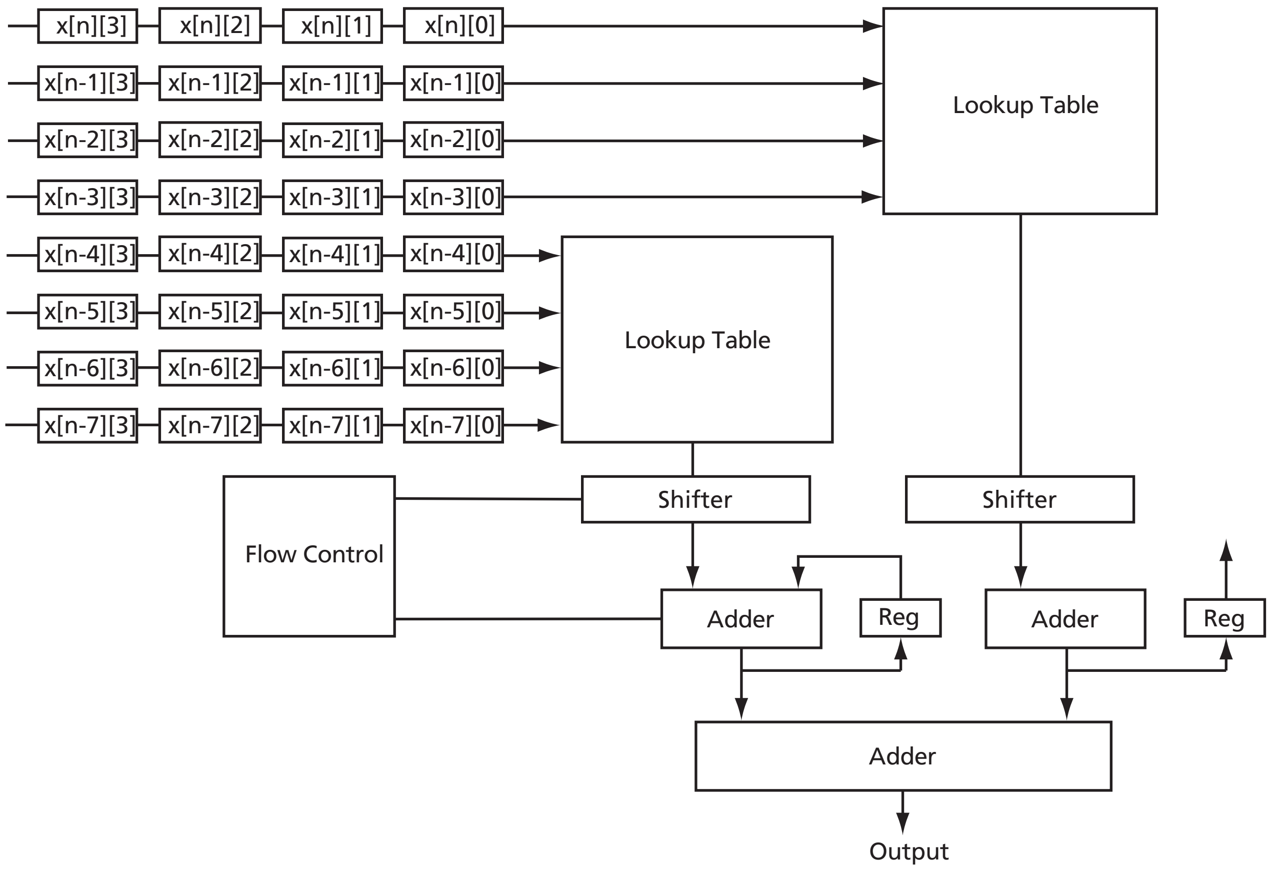


Figure 4 • **Example of Split Lookup Table Implementation**

Folding

The system clock rate of many FIR filter systems is a multiple of the data rate (or data sampling rate). For typical FPGA implementation, the size of the design is key for efficient implementation. Thus, exploitation of the ratio between the system clock rate and data rate is an effective approach to reduce the size of the design. In other words, folding or serialization of the computation can reduce the size of the design. The DA algorithm for FIR introduces bit-serialization of the operations. This property of the DA can be very efficient for exploring the ratio between system clock rate and data rate. If the number of bits of input is *nbits_in*, it takes *nbits_in* table lookup and additions to finish one output point of the FIR. If the system clock rate is *nbits_in* times faster than data rate, the serialization of table lookup and additions is done with the optimized timing. The parameter *sample_ratio* defines the ratio between the system clock rate (*sys_clk_frq*) and the data sampling rate (*data_rate*), as shown in EQ 8:

$$sample\_ratio = sys\_clk\_frq/data\_rate$$

EQ 8

CoreFIR supports folding when *sample_ratio* is greater than or equal to *nbits_in*. The serialized operations of table lookup and addition are done in *nbits_in* clock cycles of the system clock, and the design is idle during the rest of *sample_ratio* and *nbits_in* cycles. The generator only requires that the *sample_ratio* be an integer; the system clock rate is an exact multiple of the data rate. Future releases may support a *sample_ratio* less than *nbits_in*.

DA LUTs Using FPGA Cells

Some Actel FPGA families such as SX-A and RTSX-S do not have an embedded RAM implementation. In this case, the CoreFIR Generator requires that the lookup table be hard-coded as ROM using FPGA cells. This configuration does not need the DA LUT Generator shown in Figure 2 on page 5. The generator selects this configuration when the configurable parameter *coef_fixed* is set to 1 or the configuration parameter fpga_family is not one of *ax*, *apa*, or *pa3*.

DA LUTs Using Embedded RAM Blocks

Many Actel FPGA families have embedded RAM blocks. The FIR generator takes advantage of these embedded RAM blocks, and the DA LUTs are implemented using these embedded RAM blocks. This configuration requires additional overhead in that the embedded RAM blocks must be initialized by a DA LUT Generator as shown in Figure 2 on page 5. The generator selects this configuration when the configurable parameter *coef_fixed* is set to 0 and the configuration parameter *fpga_family* is set to *ax*, *apa*, or *pa3*.

DA LUT Generation

The DA LUT Generator computes the LUT contents of the distributed arithmetic algorithm. It reads the coefficients from the Input Buffers block and writes the LUT words into the embedded RAM blocks. This block is only available when using embedded RAM blocks as LUTs – when the configuration parameter *coef_fixed* is set to 0. After the reset is complete, the DA LUT Generator will wait for the Input Buffers block to signal that the coefficients are loaded into the input buffers. Then the DA LUT generator will compute the LUT words and write them into the embedded RAM blocks. The DA LUT Generator produces LUT contents for multiple LUTs when implementing a FIR filter with a large number of taps. The generator has only one computation engine, and initializes multiple LUTs sequentially. After the initialization of the RAM blocks, the output *ready* will go high to let the system know that the FIR filter is ready to accept data.

Input Buffering Scheme

The Input Buffers block always performs the functions defined in "Input Coefficients" on page 10, but only performs functions defined in "Input Data" on page 10 when the embedded RAM blocks are used as the DA LUTs (when *coef_fixed* is set to 1).

Input Data

The input dataflow is designed to use the scheme shown in Figure 5 to reduce the size of registers. The horizontal movement of the input ensures the bits of the inputs feed into the lookup table, and that it happens at every cycle. Vertical movement of input data only occurs as the most significant bit (MSB) is fed into lookup table, when it switches to the next FIR data point.

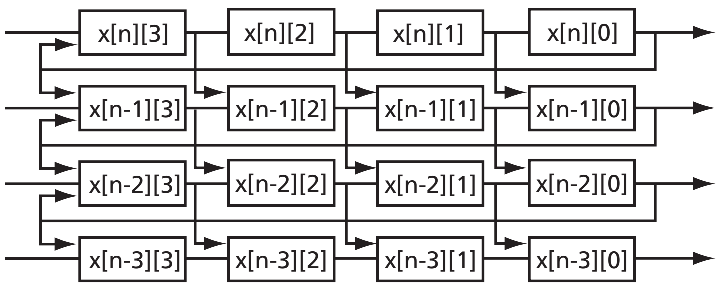


Figure 5 • **Example of Input Buffering Scheme**

Input Coefficients

The CoreFIR generator shares the input buffers for coefficients input when embedded RAM blocks are used as the DA LUTs. The configuration parameter *coef_fixed* is set to 0. In this configuration, the width of the input datai is the maximum of *nbits_in* and *nbits_coef*. The input *datai* reads in coefficients when input *coefi_en* is high. After enough coefficients are fed into the buffer, *coefi_en* is ignored and the coefficients stay inside the input buffers until the DA LUT Generator finishes the initialization of the embedded RAM blocks.

User Interface

The generator executable reads one command line parameter, which is the name of the configuration file. It generates RTL code for the module and testbench based on the parameters in the configuration file. Refer to Table 4 on page 7 and "Appendix I: Sample Configuration File" on page 12 for details of the configuration file.

Clock and Reset

Clock

The CoreFIR generates a FIR filter design that uses only positive-edge-triggered registers. The entire design is fully synchronized using the positive edge of the input clock *clk*, including the embedded RAM blocks (when available).

Reset

The CoreFIR generates a design that uses only one active low asynchronous reset. The entire design is asynchronously reset by the input *rstn*.

Input and Output Timing

I/O Timing Diagram of Normal FIR Operation

The I/O timing under normal FIR operation is illustrated in Figure 6. The labels s0 and s2 refer to the data sampling point for input data, while s1 is the sampling point for output data. Due to variations of the configuration, you should refer to comments in the generated module for t0 and t1. These parameters are given in the number of the clock cycles of the input clock, *clk*.

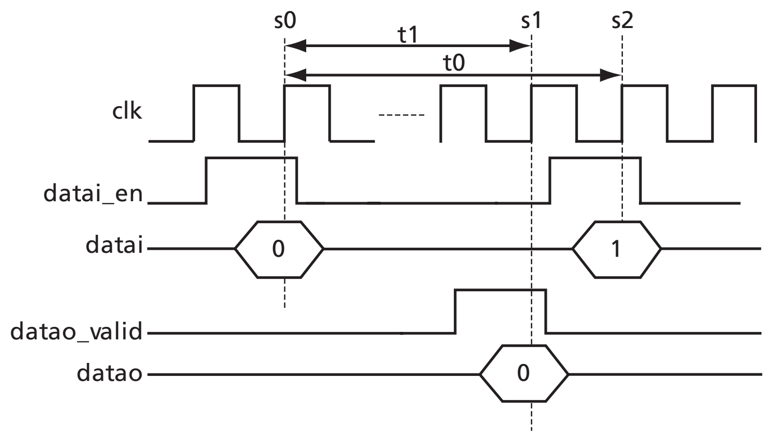


Figure 6 • **I/O Timing Diagram of Normal FIR Operation**

I/O Timing Diagram of LUT Initialization

The I/O timing for LUT initialization is illustrated in Figure 7. In this figure, s0 and s1 are the starting and ending points for feeding coefficients, while s2 is the sampling point for output ready. Due to the variation of the configuration, refer to the comments inside the generated module for t2, which are given in the number of the clock cycles of the input clock *clk*.

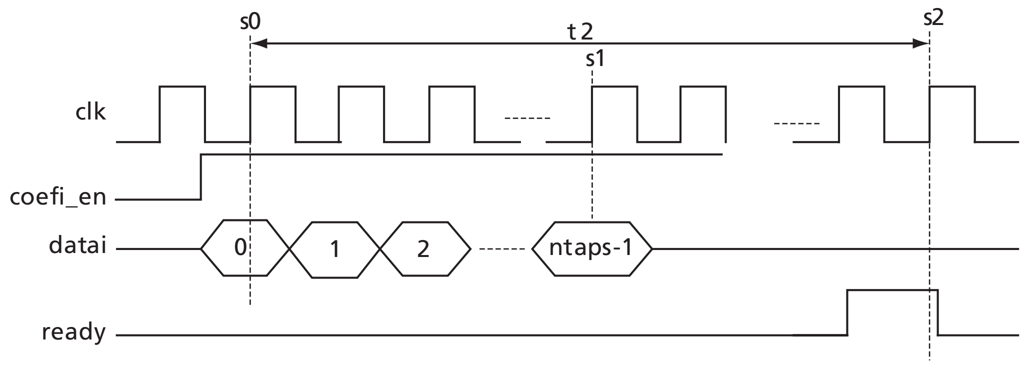


Figure 7 • **I/O Timing Diagram for LUT Initialization**

Appendix I: Sample Configuration File

The following shows a sample configuration file.

```
module_name     firtest
nbits_input     8
nbits_coef      5
ntaps           13
tap             8 14 21 27 31 31 27 21 14 8 4 2 1
data_signed     0
fpga_family     ax
coef_fixed      1
sys_clk_frq     25
sample_ratio    16
module_lang     vhdl
```

Ordering Information

Order CoreFIR through your local Actel sales representative. Use the following numbering convention when ordering: CoreFIR-XX, where XX is listed in Table 5.

Table 5 • **Ordering Codes**

XX	Description
EV	Evaluation Version
AR	RTL for unlimited use on Actel devices
UR	RTL for unlimited use and not restricted to Actel devices

List of Changes

The following table lists critical changes that were made in the current version of the document.

Previous Version	Changes in Current Version (v3.0)	Page
v2.1	The "Supported Families" section was updated to include Fusion.	1
	Table 1 was updated to include Fusion data.	2
v2.0	The "Supported Families" section was updated.	1
	Table 1 was updated.	2

Datasheet Categories

In order to provide the latest information to designers, some datasheets are published before data has been fully characterized. Datasheets are designated as "Product Brief," "Advanced," and "Production." The definitions of these categories are as follows:

Product Brief

The product brief is a summarized version of an advanced or production datasheet containing general product information. This brief summarizes specific device and family information for unreleased products.

Advanced

This datasheet version contains initial estimated information based on simulation, other products, devices, or speed grades. This information can be used as estimates, but not for production.

Unmarked (production)

This datasheet version contains information that is considered to be final.

www.actel.com

Actel Corporation	Actel Europe Ltd.	Actel Japan	Actel Hong Kong
		www.jp.actel.com	www.actel.com.cn
2061 Stierlin Court Mountain View, CA 94043-4655 USA	Dunlop House, Riverside Way Camberley, Surrey GU15 3YL United Kingdom	EXOS Ebisu Bldg. 4F 1-24-14 Ebisu Shibuya-ku Tokyo 150 Japan	Suite 2114, Two Pacific Place 88 Queensway, Admiralty Hong Kong
Phone 650.318.4200 **Fax** 650.318.4600	**Phone** +44 (0) 1276 401 450 **Fax** +44 (0) 1276 401 490	**Phone** +81.03.3445.7671 **Fax** +81.03.3445.7668	**Phone** +852 2185 6460 **Fax** +852 2185 6488

51700056-2/12.05