Digital Signal Processing Reference Guide

Reference Guide

December 2014

Microsemi

Revision History

Date	Revision	Change
16 December 2014	Revision 2	Second Release
05 June 2014	Revision 1	First Release

Confidentiality Status

This is a non-confidential document.

Table of Contents

DSP Reference Guide

Introduction

SmartFusion®2 system-on-chip (SoC) field programmable gate array (FPGA)/IGLOO®2 FPGA devices have unique architectural features to address various digital signal processing (DSP) based applications for high performance systems. These features include the embedded mathblocks for high efficient arithmetic computations, large static RAM (LSRAM) for bulk data storage, and micro SRAM (uSRAM) for small data storage requirements such as coefficients. In addition to these features, flash based technology and low power options built in the Microsemi® SoC FPGAs offer unique advantages in low power DSP based systems.

The SmartFusion2/IGLOO2 mathblocks facilitate the following different signal processing applications:

- Finite Impulse Response (FIR) filters
- Infinite Impulse Response (IIR) filters
- Fast Fourier Transforms (FFT)
- Inverse Fast Fourier Transforms (IFFT)
- Discrete Cosine Transforms (DCT)

The SmartFusion2/IGLOO2 mathblocks have a built-in multipliers and adders that minimize the external logic required to implement multiplication, multiply-add, and multiply-accumulate (MACC) functions resulting in efficient resource usage and improved performance for DSP applications.

This reference guide describes the architectural features of the embedded mathblock, different DSP design methodologies that are supported, and a few widely used DSP applications with performance details.

Advantages of using the SmartFusion2/IGLOO2 Devices for DSP Applications

This section describes the advantages of using SmartFusion2/IGLOO2 devices for DSP applications. It has the following subsections:

- Fabric Performance
- Low Power FPGA
- Embedded Micro Controller

Fabric Performance

- SmartFusion2/IGLOO2 has double the fabric performance compared to the previous generation of Microsemi FPGAs.
- Built-in mathblocks that support DSP applications at or up to 350 MHz
- Built-in fabric clock conditioning circuits (FCCC) that generate clock frequencies of up to 400 MHz.
- Fabric memories, LSRAM and uSRAM operating at 400 MHz
- 16x 5 Gbps serializer/deserializer (SERDES), PCIe, and XAUI/XGXS + Native SERDES
- Densities up to 150 K look-up table (LUT), 5 Mbit SRAM, and 4 Mbit eNVM

Low Power FPGA

- 10 mW static power during operation

Embedded Micro Controller

- SmarFusion2 SoC FPGA has a 166 MHz ARM® Cortex®-M3 processor with on-chip embedded SRAM (eSRAM) and embedded nonvolatile memory (eNVM) for processor based DSP applications.
- The microcontroller subsystem (MSS) includes peripherals for - Control area network (CAN), Tri-Speed Ethernet, and universal serial bus (USB)

SmartFusion2/IGLOO2 Mathblock Architecture

The SmartFusion2/IGLOO2 mathblock architecture is optimized to implement various common DSP functions with maximum performance and minimum logic resource utilization. The dedicated routing region around the mathblock and the feedback paths provided in each mathblock maintain high performance by eliminating routing congestion.

Key Features of Mathblock

- High-performance, power optimized multiplications operations
- Supports 18 × 18 signed multiplication natively
- Supports 17 × 17 unsigned multiplications
- Supports dot-product: The multiplier computes (A[8:0] × B[17:9] + A[17:9] × B[8:0]) × 2^9.
- Built-in addition, subtraction, and accumulation units to combine multiplication results efficiently
- Adder support: (A × B) + C or (A × B) + D or (A × B) + C + D
- Independent registered third input C with data width of 44 bits
- Supports both registered and unregistered inputs and outputs (I/Os)
- Supports signed and unsigned operations
- Internal cascade signals (44-bit cascade input (CDIN) and cascade output (CDOUT)) enable cascading of the mathblocks to support larger accumulator/adder/subtractor without extra logic.
- Support loopback capability
- Supports up to 350 MHz operation
- Clock-gated input and output registers for power optimizations
- Capability to extend the width of adder/accumulator by implementing extra address in the FPGA fabric or using mathblocks

Mathblocks Resources

Table 1 lists the number of mathblocks available in the SmartFusion2/IGLOO2 devices.

Table 1 • **Mathblocks Resources**

SmartFusion2 Device	IGLOO2 Device	Number of Mathblock Rows	Number of Mathblocks Per Row	Total Number of Mathblocks
M2S005	M2GL005	1	11	11
M2S010	M2GL010	2	11	22
M2S025	M2GL025	2	17	34
M2S050	M2GL050	3	24	72
M2S090	M2GL090	3	28	84
M2S150	M2GL150	6	40	240

Figure 1 shows the functional diagram of the mathblock.

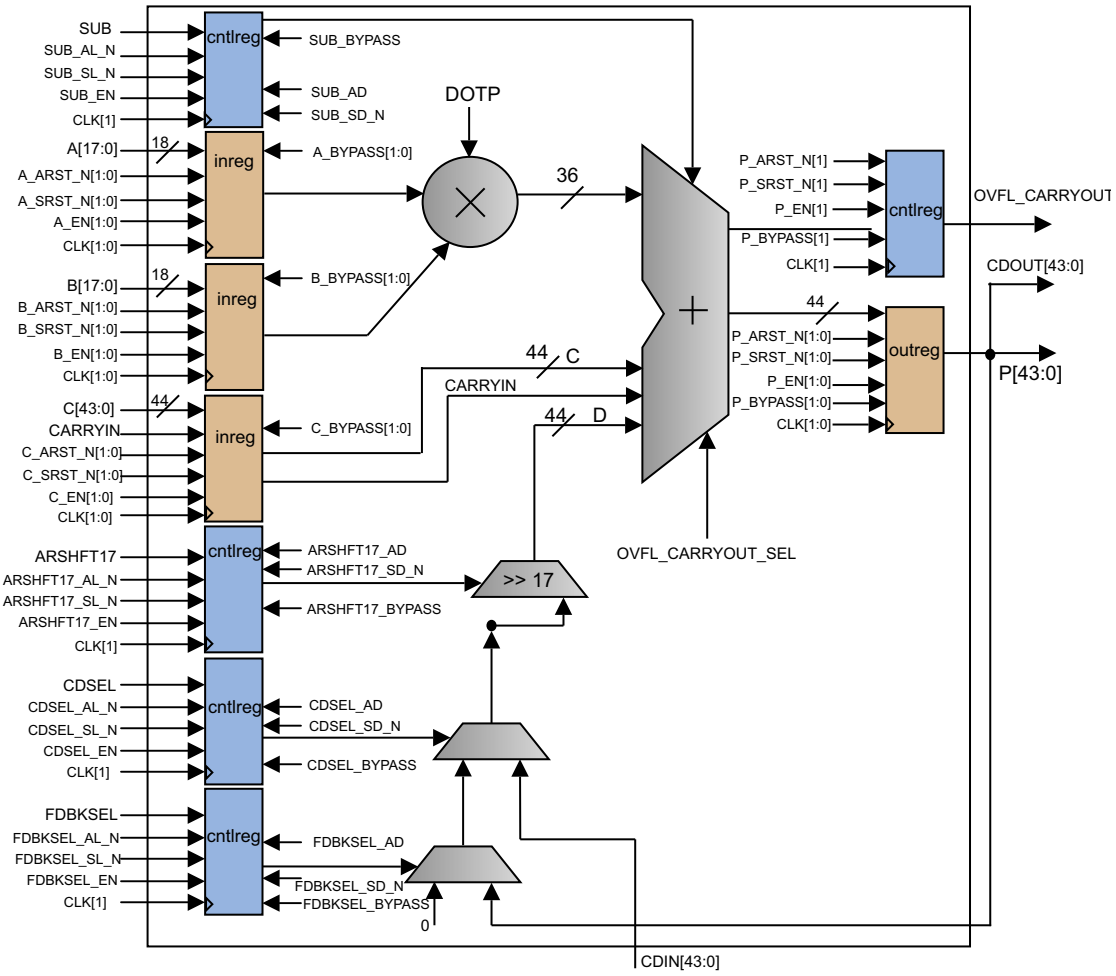

***Figure 1* • Functional Diagram of the SmartFusion2/IGLOO2 Mathblock**

Mathblocks can be accessed through the FPGA routing architecture and cascaded in a chain, starting from the left-most block to the right-most block.

Each mathblock consists of:

- Multiplier
- Adder/Subtractor
- I/O, Control Registers

Multiplier

The SmartFusion2/IGLOO2 mathblock can be used as a multiplier, which accepts two 18-bit inputs (A and B) and generates a 36-bit output. The mathblock multiplier can be configured in two different operating modes:

- Normal Mode
- Dot Product (DOTP) Mode

Normal Mode

In Normal mode, the mathblock implements a single 18 ×18 signed multiplier. The mathblock accepts A [17:0] and B [17:0]inputs and generates A*B with a 36-bit wide result. Figure 2 shows the functional block diagram of the mathblock in Normal mode.

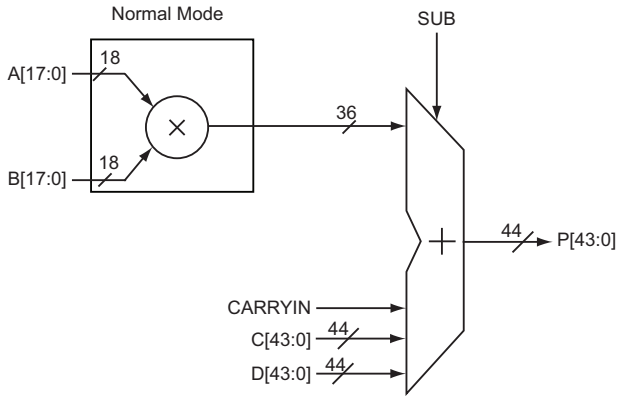

***Figure 2* • Functional Block Diagram of the Mathblock in Normal Mode**

Dot Product (DOTP) Mode

DOTP mode has two independent 9-bit × 9-bit multipliers with adder and the product sum is stored in upper 36 bits of 44-bit register. In Dot Product (DOTP) mode, the mathblock implements the following equation:

$$\text{DOTP result} = (A[8:0] \times B[17:9] + A[17:9] \times B[8:0]) \times 2^{9}$$

EQ 1

DOTP mode can be used to implement 9 × 9 complex multiplications.

Figure 3 shows the functional block diagram of the mathblock in DOTP mode.

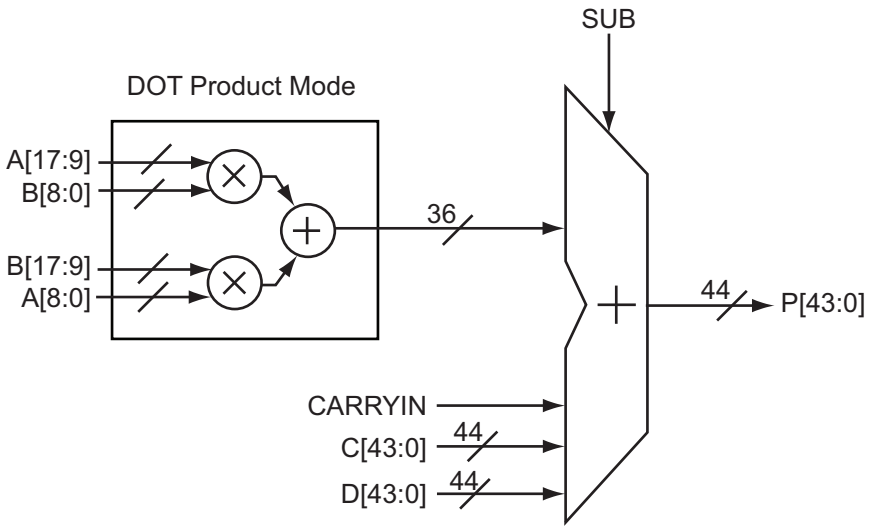

***Figure 3* • Functional Block Diagram of the Mathblock in DOTP Mode**

Math Functions with DOTP

When DOTP is enabled, several mathematical functions can be implemented using a single mathblock. Some of them are listed in Table 2

***Table 2* • Math Functions with DOTP**

S.No	Conditions	Implemented Equations
1	P = A[8:0] = B[17:9]; M = A[17:9]; N = B[8:0]	$Y = P^2 + M \times N$
2	P = A[8:0] = B[17:9]; Q = A[17:9] = B[8:0]	$Y = P^2 + Q^2$
3	A[8:0] = B[17:9] = 1; B = A[17:9]; Q = B[8:0]	$Y = 1 + Q^2$
4	A[8:0] = B[17:9] = 1; P = A[17:9]; Q = B[8:0]	$Y = 1 + P \times Q$
5	P = A[8:0] = A[17:9]; Q = B[17:9] = B[8:0]	$Y = P \times Q + P \times Q = 2 \times P \times Q$

The 9-bit×9-bit multipliers are extensively used in low precision video processing applications such as color space converters (YCbCr to RGB, YUV to RGB, etc), chroma resampler NTSC, PAL, etc,. In image processing, the operations involving 8-bit RGB such as 3×3, 5×5, 7×7 matrix multiplications, image enhancement techniques, scaling, resizing etc., 9-bit×9-bit multipliers are used. The SmartFusion2/IGLOO2 devices address these applications by using the mathblock in DOTP mode.

DOTP Mode Usage Recommendations

When designing with DOTP multiplier, the following recommendations must be used to achieve better performance:

- To perform Y = A×B + C×D equation, instantiate arithmetic IP from Libero catalog cores with DOTP enabled for 9×9 multiplications. This avoids inferring two 18×18 multipliers.
- Register the I/Os, when using arithmetic IP cores (mathblock).
- The registered I/Os must use the same clock.

Use the cascaded feature to connect the multiple mathblocks. This is achieved by connecting the CDOUT of one mathblock to the CDIN of another mathblock. Refer to "DSP Applications" section for more information on design examples.

Adder/Subtractor

The adder sums the output from the multiplier, C input, CARRYIN, or D input. The final output (P) of the adder is ((A [17:0] × B [17:0]) + C [43:0] + D [43:0] + CARRYIN).

The mathblock can be configured as a 2-input or 3-input adder.

- As a 2-input adder, the mathblock computes A × B + C or A × B + D.
- As a 3-Input adder, the mathblock computes A × B + C + D.

If the adder is configured as a subtractor, the adder output is ((C [43:0] + D [43:0] + CARRYIN) – (A[17:0] × B[17:0])).

I/O, Control Registers

Mathblocks have built-in registers on data inputs (A, B, C), data output (P), and control signals. If required, these registers can be bypassed. All the registers in the mathblock have clock gating capability to reduce the power consumption.

Mathblocks do not have a pipeline register at the cascade input (CDIN). So, pipeline registers can be added from the fabric when multiple mathblocks are cascaded to implement higher bit-width multiplications.

A Input Register, B Input Register, and C Output Register

A and B are the input registers with 18-bit data width and P is output register with 44-bit data width of the mathblock.

C Input

The C input port allows the formation of many 3-input mathematical functions, such as 3-input addition or 2-input multiplication with an addition. The CARRYIN signal is the carry input of the adder or accumulator. The C input can also be used as a dynamic input achieving the following functionalities:

- Wrapping-around the cascade chain of mathblocks from one row to the next row through the fabric
- Rounding of multiplication outputs
- Trimming of lower order bits of the final sum or partial sum or the product

Rounding

Rounding can be computed by adding a fixed term and a variable term to the input value to be rounded and then truncating. The fixed term can be fed from the C-Input of the mathblock and the value depends on the number of decimal points required after rounding. The variable term is always a single bit in the least-significant position whose value may be determined from the input value based on the type of rounding.

Types of rounding are:

- Round to the adjacent even integer: The variable term is determined from the 2^0 bit of the input value.
- Round towards zero: The variable term is determined from the sign bit of the input value. For example, 1.5 rounds to 1 and -1.5 rounds to -1.

Table 3 provides examples for 6-bit values including three fraction bits 000.001.

Table 3 • **Rounding Examples**

Input Value		Fixed Term C-Input	Round To Even				Round Toward Zero			
Decimal	Binary		Variable Term	Sum	Truncated Sum	Decimal	Variable Term	Sum	Truncated Sum	Decimal
2.5	010.100	0.011	000.000	010.111	010	2	000.000	010.111	010	2
1.5	001.100	0.011	000.001	010.000	010	2	000.000	001.111	001	1
-1.5	110.100	0.011	000.000	110.111	110	-2	000.001	111.000	111	-1
-2.5	101.100	0.011	000.001	110.000	110	-2	000.001	110.000	110	-2

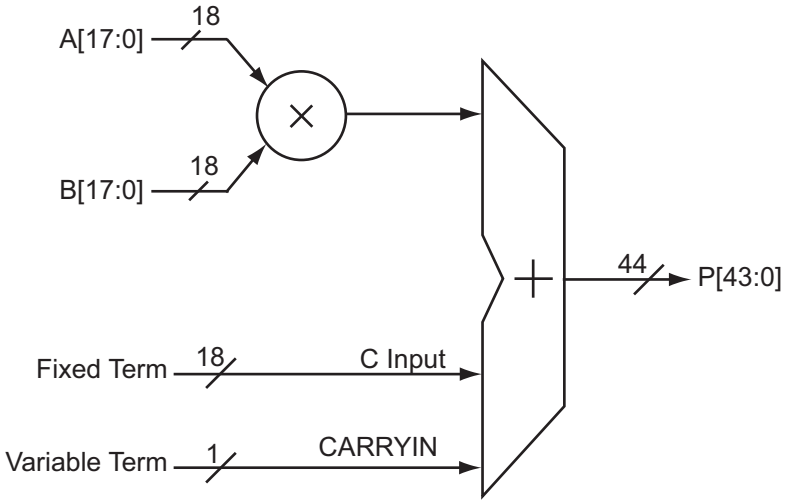

***Figure 4* • Rounding using C-Input and CARRYIN**

Trimming

Trimming of the Final Sum: Applications like IIR and FFT often requires the rounding and trimming of the final result (for example, last output of a cascade chain or the final value read from an accumulator). The addition of the rounding terms can be done as shown in Figure 5 and final results can be trimmed in the fabric.

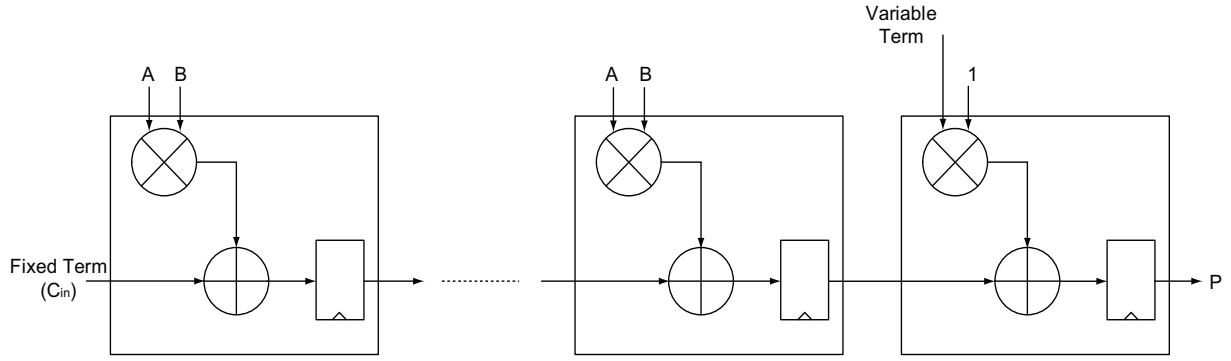

***Figure 5* • Rounding and Trimming of the Final Sum**

Note: The Fixed Term is connected to Cin of first mathblock in a cascaded chain. The Variable Term is connected to the multiplier input (A or B) of the last mathblock in a cascaded chain as shown in Figure 5.

Trimming of Grouped Sums: When computing very large dot products (for example, a large, fully-enumerated FIR) it is good to avoid overflow by breaking the sum into a few groups, trimming the sum for each group, and only then combining the groups' sums into a final result. The rounding of each group's sum can be done as shown in Figure 5. The trimming of each group's sum and summation of the final result can be done in the fabric. Trimming can be done between the output of each cascade and the final fabric adder.

Trimming of Products: Figure 6 shows the implementation of rounding all products towards zero and then trimming the least significant m bits of the product. As long as there are no additive terms other than the products, it is possible to equivalently trim the partial sums instead of the products. Round towards zero can be done using sign bit of the product (A*B) from the sign bits of the incoming factors A and B using an EXOR.

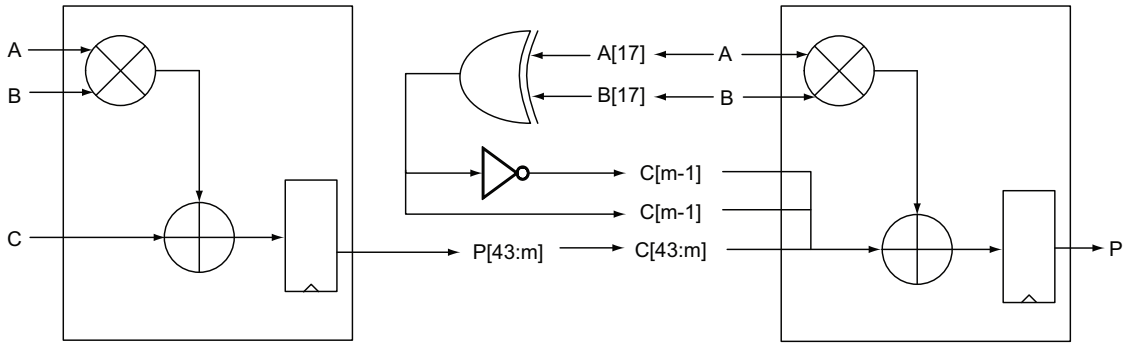

***Figure 6* • Rounding and Trimming of Products**

Cascaded Input, Output, and Selection

Higher level DSP functions are supported by cascading individual mathblocks in a row. The two data signals, CDIN [43:0] and CDOUT [43:0], provide the cascading capability with a cascade select input (CDSEL). Table 4 shows the selection of CDSEL for propagating CDIN to the D input of the adder. To cascade mathblocks, the CDOUT of one block must feed the CDIN of another block. CDOUT to CDIN is a hardwired connection between the blocks within a row.

Two different rows can be cascaded using the fabric routing between the two rows. Extra pipeline registers may be needed to compensate for the extra delays added due to the fabric routing, which in turn will increase the latency of the chain.

The ability to cascade mathblocks is useful in filter designs. For example, an finite impulse response (FIR) filter design can use cascading inputs (CDINs) to arrange a series of input data samples and cascading outputs (CDOUTs) to arrange a series of partial output results. The ability to cascade provides a high-performance and low power implementation of DSP filter functions because the general routing in the fabric is not used. For more details refer to "DSP Applications" section on page 19 section.

Overflow Output

Each mathblock has an overflow signal, OVFL_CARRYOUT. This signal indicates any overflow from the additional operation performed by the adder. This signal is also used to extend the adder data widths from the existing 44 bits using fabric. The overflow signal is also used for the implementation of saturation capabilities. Saturation refers to catching an overflow condition and replacing the output with either the maximum (most positive) or minimum (most negative) value that can be represented. In the IGLOO2 mathblocks, this capability is implemented using the adder's output sign bit (MSB [43] bit of the P output) and the overflow signal.

Shift Input

For multi-precision arithmetic, mathblocks provide a right-wire-shift by 17, which is controlled by the ARSHFT17 input. Thus, a partial product from one mathblock can be shifted to the right and added to the next partial product computed in an adjacent mathblock. Using this technique, mathblocks can be used to build bigger multipliers.

Feedback Select Input

For accumulation operations, the mathblock output needs to loopback to the D input of the adder block. Selection of the D input is controlled by the feedback select (FDBKSEL) input. Table 4 shows the selection of FDBKSEL for loopback.

Table 4 • **Truth Table for Propagating Operand D of the Adder/Accumulator**

CDSEL	FDBKSEL	ARSHFT17	Operand D
0	0	0	0
0	0	1	0
1	X	0	CDIN[43:0]
1	X	1	{{17{CDIN[43]}}, CDIN[43:18]}
0	1	0	P[43:0]
0	1	1	{{17{P[43]}}, P[43:18]}

Design Methodologies

Following are the different design methodologies for developing the DSP based applications using Microsemi SoCs or FPGAs:

- RTL Based Design
- Model-Based Design (Synphony ME Compiler)
- C-Based Design

RTL Based Design

Register transfer level (RTL) based design is a conventional approach for developing the FPGA based designs. In the DSP applications, mathblock can be used by inferring through an RTL coding style or by the mathblock primitives available in the Libero® System-on-Chip (SoC) Smart cores. The Synplify Pro® automatically infers mathblock, if the design contains multiply, multiply-accumulate, and multiply-add/subtract operators.

The mathblock primitive available in the Libero SoC IP catalog is called MACC. The mathblock primitive can be used in the designs by SmartDesign for schematic-based design entry or by directly instantiating the mathblock wrapper in a hardware description language (HDL) file as a component.

For more information on VHDL/Verilog coding styles for inferring mathblocks, refer to the *Inferring Microsemi SmartFusion2 MACC Blocks*.

Smart Cores for Mathblock Configuration

The SmartFuison2/IGLOO2 devices have embedded hard mathblocks for simple to complex arithmetic functions used for the DSP applications. The Libero tool has Arithmetic Catalog which includes the following cores to configure the embedded hard mathblock.

- Hard Multiplier Accumulator
- Hard Multiplier AddSub
- Hard Multiplier Signed

Hard Multiplier Accumulator

The hard multiplier accumulator for the SmartFusion2/IGLOO2 devices supports Normal (Figure 7 on page 13) and Dot Product (Figure 8) mode multiplications. In Figure 7 on page 13, the control registers are in blue color and the data registers are in brown color.

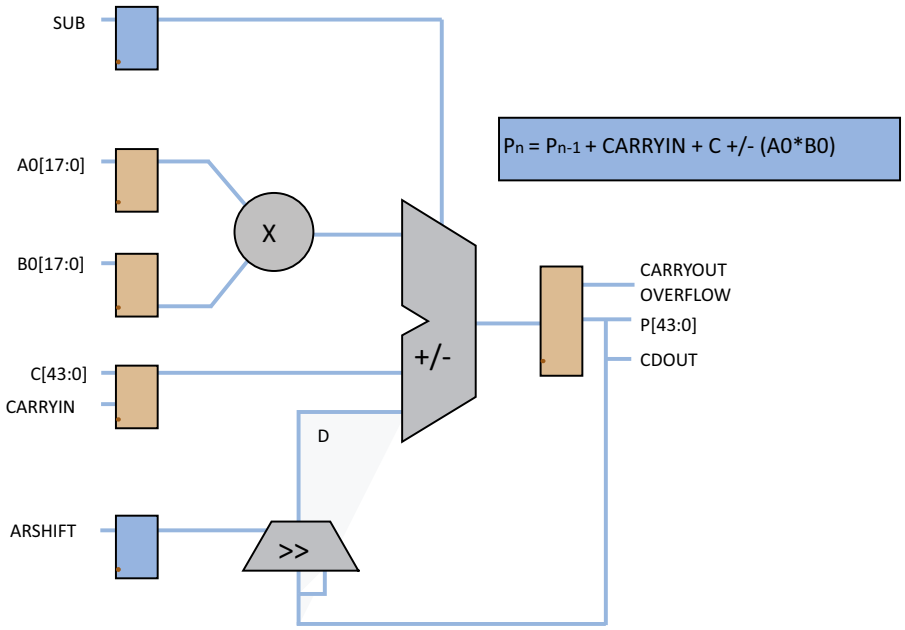

***Figure 7* • Hard Multiplier Accumulator - Normal Mode**

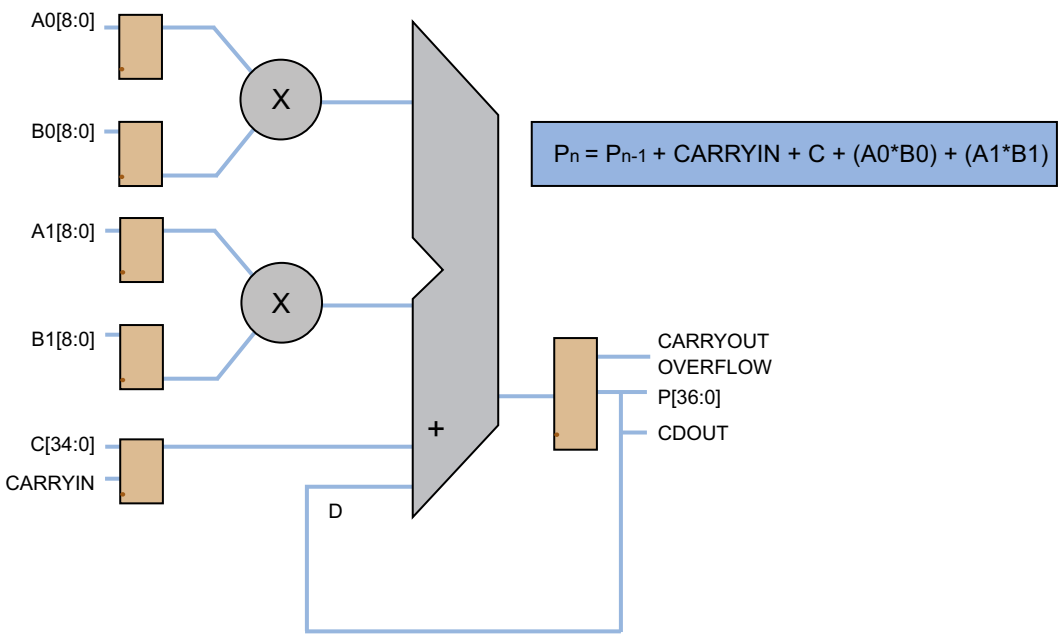

***Figure 8* • Hard Multiplier Accumulator - DOTP Mode**

Smart Cores Key Features

The hard multiplier accumulator supports two operating modes: Normal and Dot Product.

- A structural netlist is generated in either Verilog or VHDL
- Individual inputs and outputs can be optionally registered with:
  - A common rising-edge clock
  - The independent active-low asynchronous and synchronous clear controls
  - The independent active-high enable controls
- An additional cascade output CDOUT can be enabled. This is the sign-extended 44-bit copy of output P
- An additional Carry In input can be enabled
- An additional Carry Out or Overflow output can be enabled
- Normal mode features:
  - Configurable operand widths for A0 and B0 between 2 and 18
  - Configurable operand width for C between 2 and 44
  - Optional assignment of operand A0 to an 18-bit two's complement constant
  - Optional assignment of operand C to a 44-bit two’s complement constant
  - Option to select between multiplier followed by adder, subtractor, or dynamic addsub
  - Optional Arithmetic Right Shift by 17 bits of the feedback input
- DOTP mode features:
  - Configurable operand widths for A0, B0, A1, B1 between 2 and 9.
  - Configurable operand width for C between 2 and 35.
  - Optional assignment of operand A0 and A1 to a 9-bit two's complement constant
  - Optional assignment of Operand C to a 35-bit two’s complement constant

For usage and configuration of Hard Multiplier Accumulator, refer to the *IGLOO2/SmartFusion2 Hard Multiplier Accumulator Configuration User Guide*.

Hard Multiplier AddSub

The hard multiplier addsub for the SmartFusion2/IGLOO2 devices support Normal (Figure 9) and Dot Product (Figure 10 on page 16) mode multiplication. In Figure 9 and Figure 10 on page 16, the control registers are in blue color and the data registers are in brown color.

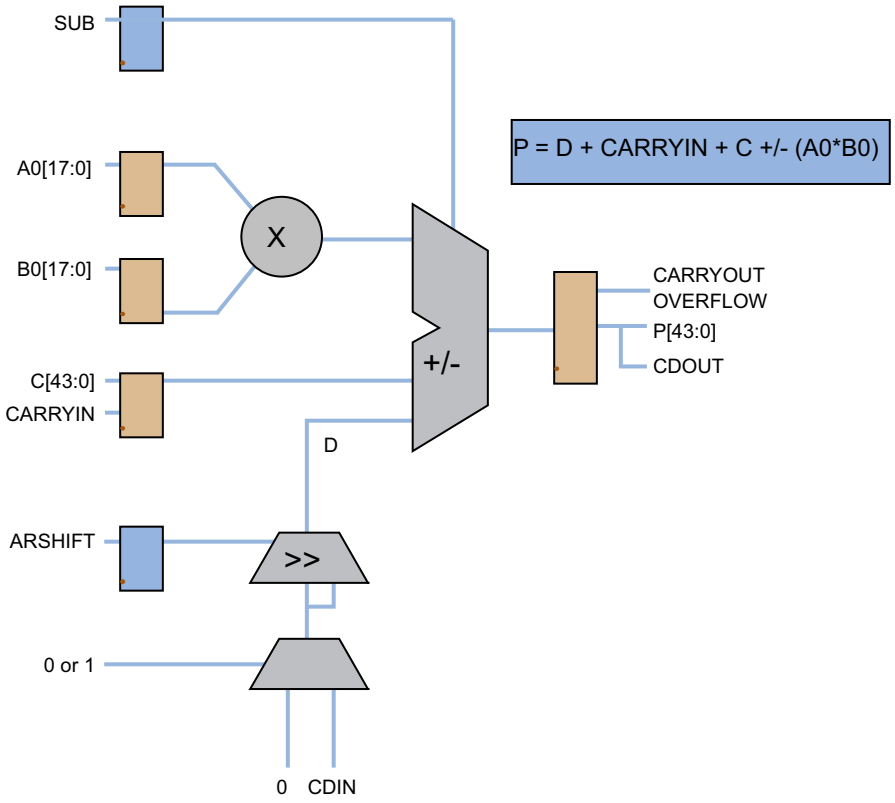

***Figure 9* • Hard Multiplier Addsub - Normal Mode**

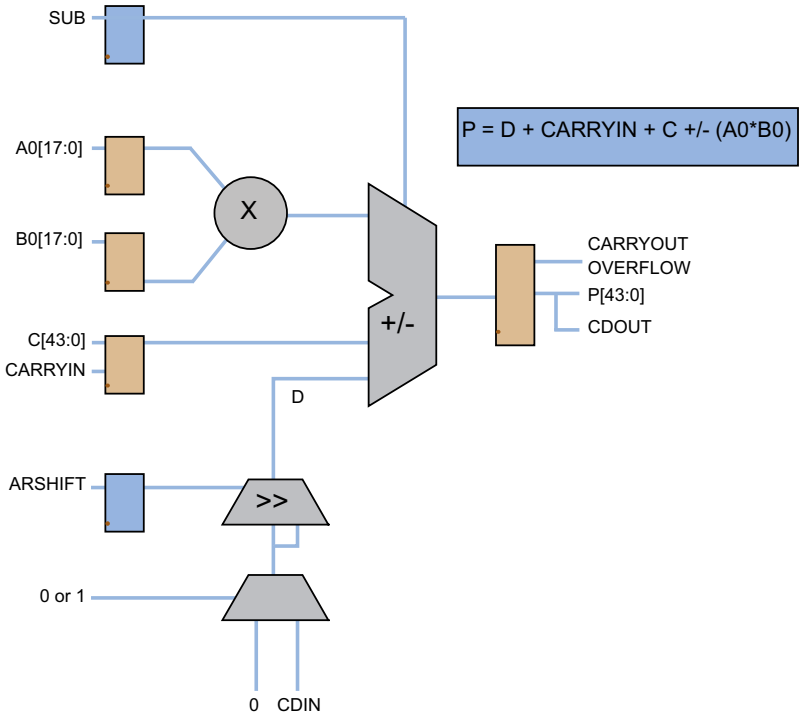

Figure 10 • **Hard Multiplier Addsub - DOTP Mode**

Smart Core Hard Multiplier AddSub Key Features

The hard multiplier addsub supports two operating modes: Normal and Dot Product.

- A structural netlist is generated in either Verilog or VHDL.
- Individual inputs and outputs can be optionally registered with:
  - A common rising-edge clock
  - Independent active-low asynchronous and synchronous clear controls
  - Independent active-high enable controls
- An additional cascade output CDOUT can be enabled. This is the sign-extended 44-bit copy of output P.
- An additional cascade input CDIN from previous mathblock can be enabled.
- An additional Carry In input can be enabled.
- An additional Carry Out or Overflow output can be enabled.
- Normal mode features:
  - Configurable operand widths for A0 and B0 between 2 and 18
  - Configurable operand width for C between 2 and 44
  - Optional assignment of operand A0 to an 18-bit two's complement constant
  - Optional assignment of operand C to a 44-bit two’s complement constant
  - Option to select between multiplier followed by adder, subtractor or dynamic addsub
  - Optional Arithmetic Right Shift by 17 bits of the Cascade input
- Dot Product mode features:
  - Configurable operand widths for A0, B0, A1, B1 between 2 and 9
  - Configurable operand width for C between 2 and 35
  - Optional assignment of operand A0 and A1 to a 9-bit two's complement constant
  - Optional assignment of operand C to a 35-bit two’s complement constant

For usage and configuration of hard multiplier accumulator, refer to the *IGLOO2/SmartFusion2 Hard Multiplier Accumulator Configuration User Guide.*

Hard Multiplier Signed

The hard multiplier for the SmartFusion2/ IGLOO2 devices support two's complement Normal (Figure 11) and Dot Product (Figure 12) mode multiplication.

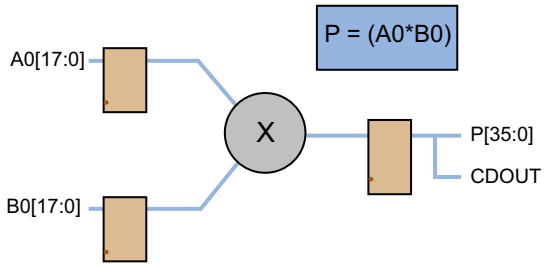

***Figure 11* • Hard Multiplier Signed - Normal Mode**

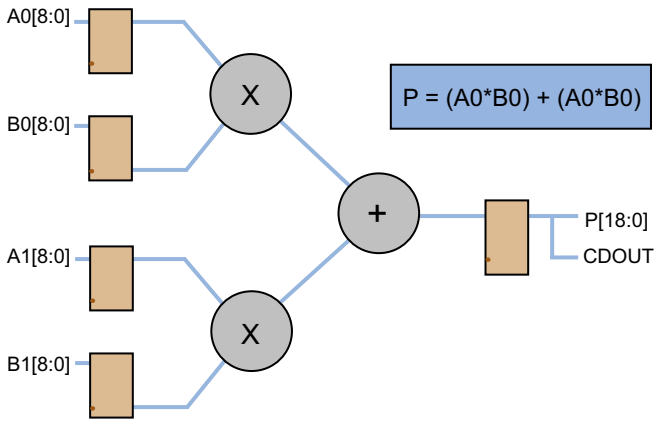

***Figure 12* • Hard Multiplier Signed - DOTP Mode**

Smart Core Key Features

The hard multiplier supports two operating modes: Normal and Dot Product.

- A structural netlist is generated in either Verilog or VHDL.
- Individual inputs and outputs can be optionally registered with:
  - A common rising edge clock
  - Independent active-low asynchronous and synchronous clear controls
  - Independent active-high enable controls
- Additional cascade output CDOUT can be enabled. This is the sign-extended 44-bit copy of output P.
- Normal mode features:
  - Configurable operand widths for A0 and B0 between 2 and 18
  - Optional assignment of operand A0 to an 18-bit two's complement constant.
- Dot Product mode features:
- Configurable operand widths for A0, B0, A1, B1 between 2 and 9
- Optional assignment of operand A0 and A1 to a 9-bit two's complement constant

For usage and configuration of hard multiplier accumulator, refer to the *IGLOO2/SmartFusion2 Hard Multiplier Accumulator Configuration User Guide*.

Model-Based Design (Synphony ME Compiler)

Synphony ME compiler is a Synopsys Microsemi Edition DSP tool for FPGA-based designs in math-works model-based design environment (Matlab-Simulink). This enables the DSP designer to evaluate an algorithm at a higher level of abstraction using MATLAB and Simulink along with an exhaustive set of DSP blockset and Microsemi IPs. Synphony Simulink blockset is a DSP library that includes DSP functions and algorithms. These functions include the common DSP building blocks such as adders, multipliers, and registers. A set of complex DSP building blocks such as forward error correction blocks, FFTs, filters, and memories is also included. Designs are captured in the DSP Simulink modeling environment using a Synphony ME compiler blockset. Synphony ME Compiler automatically converts the high-level system DSP model to RTL. The RTL can be synthesized to Microsemi FPGA/SoC using Synopsys high-level synthesis tool, Synplify Pro ME.

The Synphony ME model compiler provides a system integration platform for the DSP designs on FPGAs that integrates the RTL, Simulink, MATLAB, and C/C++ components of a DSP system and it provides a single simulation and implementation environment. The Synphony ME model compiler supports a black box block that allows RTL to be imported into Simulink and co-simulated. Figure 13 shows the DSP design flow using the Synphony ME model compiler and the Libero design tools.

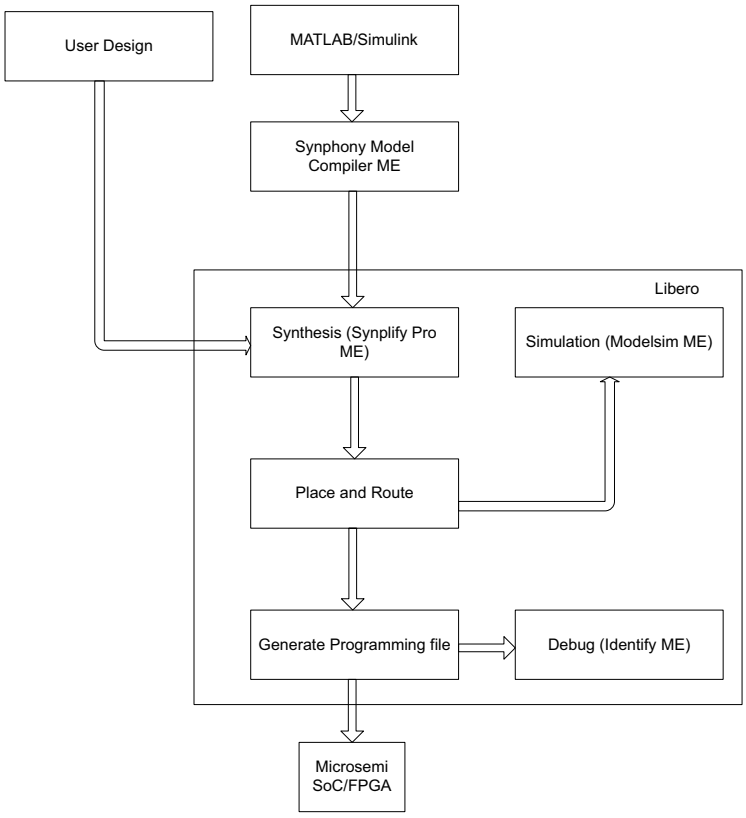

***Figure 13* • DSP Design Flow using Synphony ME Model Compiler and Libero Tools**

For more information on DSP design flow using Synphony Model compiler, refer to the *IGLOO2/SmartFusion2 DSP Flow Tutorial*.

C-Based Design

The SmartFusion2 device has a built-in Cortex-M3 processor, which uses the cortex microcontroller software interface standard (CMSIS)-DSP library to develop DSP based applications. The CMSIS-DSP library includes vector operations, matrix computing, complex arithmetic, filter functions, control functions, PID transforms, fourier transforms, and many other frequently used DSP algorithms. Most algorithms are available in floating-point and various fixed-point formats and are optimized for the Cortex-M series processors. For more information on CMSIS-DSP library and usage, visit ARM website *www.arm.com/cmsis*.

DSP Applications

This section describes how to implement DSP applications and provide example applications that uses SmartFusion2/IGLOO2 mathblocks. This sections has the following subsections:

- Advanced Math Functions
- Filter Applications
- Transform Applications

Notes:

1. *The Resource Utilization and Timing Summary given are specific to the SmartFusion2 devices. These reports apply for IGLOO2 devices also.*
2. *The VHDL design files are available as part of the document. The Verilog files will be provided on request.*

Top-level Directory Structure of the Design Files

The design files are provided for example applications. Figure 14 shows the top-level structure of the design files:

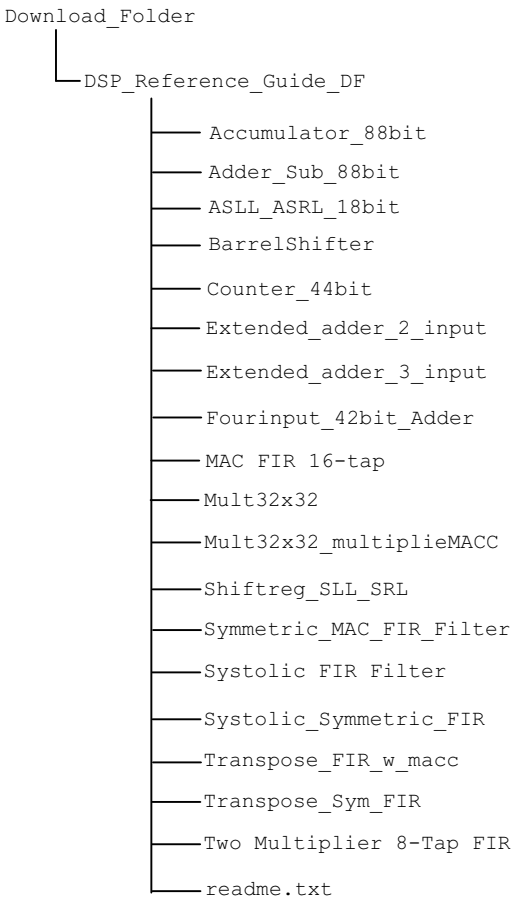

Figure 14 • **Top-level Directory Structure**

Advanced Math Functions

The SmartFusion2/IGLOO2 mathblock efficiently performs a wide range of math functions including adder, subtractor, multiplier, divider, accumulator, multiply and accumulate (MAC), counters, and shifters etc. The pipeline stages within the embedded mathblock ensure high performance arithmetic functions. The cascaded feature of mathblock and associated routing provides fast routing between the SmartFusion2/IGLOO2 mathblocks with less routing congestion to the FPGA fabric. This section describes the realization of some of the math functions using the SmartFusion2/IGLOO2 mathblocks. It has the following sub sections:

- Barrel Shift Register
- 18-Bit Shift Register
- Wide-Multiplier
- Extended Addition
- 44-Bit Counter
- 88-Bit Accumulator

Barrel Shift Register

The barrel shift register can be implemented using the SmartFusion2/IGLOO2 mathblocks and is useful to shifting the data quickly. Data shifting is required in many operations like address generation and other arithmetic functions. Using a barrel shifter, data can be shifted or rotated by any number of bits in a single operation.

The barrel shifter shifts the 18-bit value to the left by the value K. The bits shifted out of the most significant part reappear in the least significant bit (LSB) of the result, completing the circular shift.

Figure 15 shows the 18-bit barrel shifter using two SmartFusion2/IGLOO2 mathblocks.The 18-bit barrel shifter (ROL) can be implemented using the following equation:

Result $\Rightarrow$ A*2K + A*(2K-1 / 2^17)

$\Rightarrow$ A*2K + (A/2)*(2K) / 2^17

$\Rightarrow$ A*2K + {(A/2)*(2K)} >> 17

EQ 2

For example, Figure 15 shows that input A is left shift 3CCAD by 5 bits and input B is shifted value 2^5 to first mathblock. The result achieved by 5 bits shifted output is stored in 195AF. Here, input A is an unsigned number. Refer to "Example2: Arithmetic Shift Left/Arithmetic Shift Right" section for signed number.

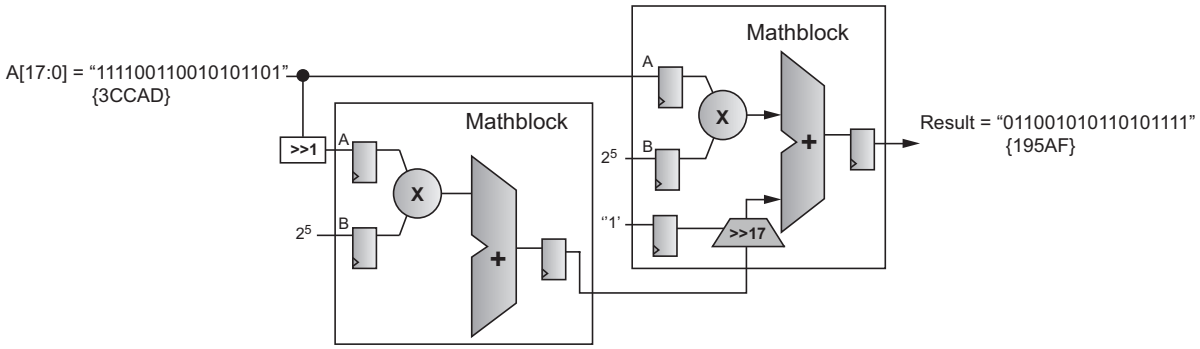

Figure 15 • **18-Bit Barrel Shifter using Mathblock**

EQ 3

Design Files

For the implementation code, refer to the `Barrelshifter_18bit.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\BarrelShifter\hdl\BarrelShifter_18bit.vhd`

Resource Utilization and Timing Summary

Report 1 shows the resources utilized by the m2s050fbga896-1 device for the Barrelshifter_18bit implementation.

Type	Used	Total	Percentage
COMB	73	56340	0.13
SEQ	90	56340	0.16
IO (W/ clocks)	38	375	10.13
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	2	72	2.78
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

Report 1 **• Resource Utilization for 18-bit Barrel Shifter**

Table 5 shows the timing summary for the BarrelShifter_18bit implemented on the m2s050fbga896-1 device.

Table 5 **• Timing Summary for 18-bit Barrel Shifter**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
CLK	4.094	244.260	5.000	200.000	1.356	0.814	5.210	9.064

18-Bit Shift Register

The SmartFusion2/IGLOO2 mathblock 18x18 multiplier can be used to perform the shift operations.

To shift left by K bits and shift right by (18-K) bits:

1. Input the value required for shifting in port A.
2. Input the 2K to port B, where K is the value required for left shift. K can be up to 17 bits.
3. The mathblock output is stored in the lower 36 bits of P[43:0], where the lower 18 bits are shifted left by K bits (P[17:0]) and the higher 18 bits (P[35:18]) are shifted right by (18-K) bits and sign extended.

Example1: Shift Left Logic/Shift Right Logic

Figure 16 shows that input A is left shifted by 5 bits and the result is stored in P[17:0]. P[35:18] shows that the input A right shifted by 13 bits (18-K) with the most significant bits (MSBs) as zeros.

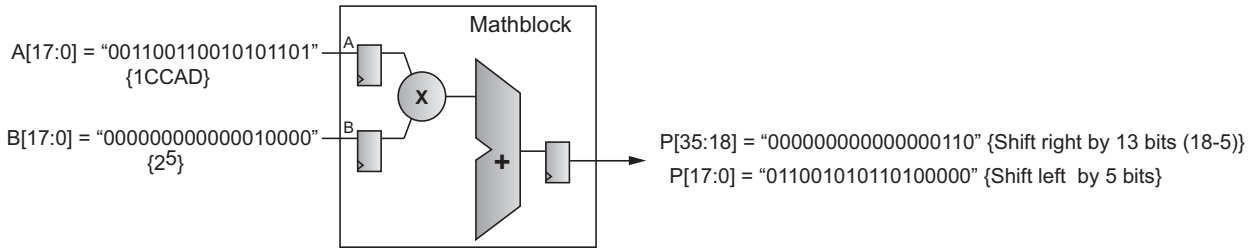

***Figure 16* • Logical Shift Left/Right using Mathblock**

Design Files

For the implementation code, refer to the `SLL_SRL.vhd` design files. Download the design files from: `<download_folder>\DSP_Reference_Guide_DF\Shiftreg_SLL_SRL\hdl\SLL_SRL.vhd`

Synthesis and Place-and-Route Results

Report 2 shows the resources utilized by the m2s050fbga896-1 device for the Shift Left logic 18-bit implementation.

Type	Used	Total	Percentage
COMB	37	56340	0.07
SEQ	53	56340	0.09
IO (W/ clocks)	55	375	14.67
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	1	72	1.39
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 2* • Resource Utilization for Shift Left /Right Logic**

Table 6 shows the timing summary for the Shift Left logic 18-bit implemented on the m2s050fbga896-1 device

***Table 6 •* Timing summary for Shift Left /Right Logic**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
CLK	2.189	456.830	5.000	200.000	0.721	0.808	5.091	9.026

Example2: Arithmetic Shift Left/Arithmetic Shift Right

Figure 17 shows that the input A is left shifted by 5 bits and the result is stored in P[17:0], whereas the P[35:18] shows that the input A is right shifted by 13 bits (18-k) with sign bits extended.

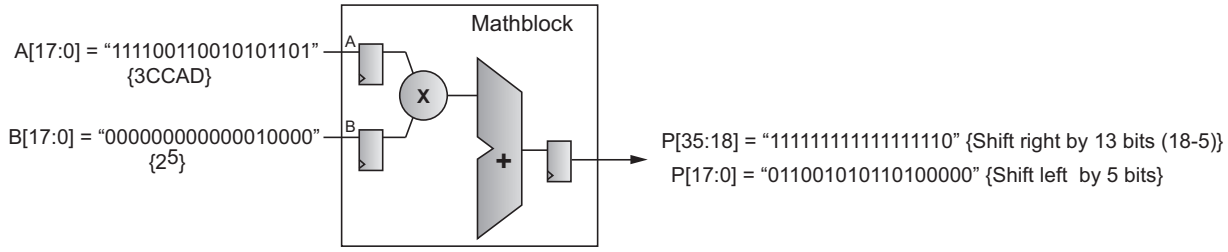

***Figure 17* • Arithmetic Shift Left/Right using Mathblock**

Design Files

For the implementation code, refer to the `ASLL_ASRL.vhd` design files. Download the design files from: `<download_folder>\DSP_Reference_Guide_DF\ASLL_ASRL_18bit\hdl\ASLL_ASRL.vhd`

Synthesis and P-and-Route Results

Report 3 shows the resources utilized by the m2s050fbga896-1 device for the Arithmetic Shift Left logic 18-bit implementation.

Type	Used	Total	Percentage
COMB	37	56340	0.07
SEQ	53	56340	0.09
IO (W/ clocks)	55	375	14.67
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	1	72	1.39
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 3* • Resource Utilization for 18-bit Arithmetic Shift Left/Right Logic**

Table 7 shows the timing summary for the Arithmetic Shift Left logic 18-bit implemented on the m2s050fbga896-1 device.

Table 7 • **Timing summary for 18-bit Arithmetic Shift Left/Right Logic**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
CLK	2.189	456.830	5.000	200.000	0.931	0.830	5.221	9.341

Wide-Multiplier

Wide-multipliers are extensively used in high precision wireless and medical applications where more than 18×18 bits are used. These applications require high precision at every stage when implementing complex arithmetic functions used in the fast fourier transform (FFT), filters, and so on. Military and high-performance computing also require performance and precision requirements, and sometimes require single-precision and double-precision floating-point calculations for implementing complex matrix operations and signal transforms.

To implement the DSP functions that require high precision, the SmartFusion2/IGLOO2 device offers implementing wide-multipliers (that is, operands width more than 18×18) with the SmartFusion2/IGLOO2 mathblock. The wide-multipliers are implemented by cascading multiple SmartFusion2/IGLOO2 mathblocks using CDOUT and CDIN to propagate the result and to achieve the best performance results.

This section describes wide-multiplier guidelines and different implementation methods with design examples to achieve the best performance results.

Guidelines

Following are some important recommendations for implementing a wide-multiplier to achieve the best results:

- The I/Os are registered with the same clock
- Add pipeline stages in RTL, so that the synthesis tool can automatically infer registers of mathblock or register the I/Os of mathblock, if arithmetic cores (mathblock) are used.
- CDOUT of one mathblock is connected to the CDIN of another mathblock.

Design Examples

This section explains the 32×32 multiplier implementation with multiple and single mathblock. It also shows the performance results for both the implementations.

This section shows the extended addition using the following design examples:

- Example1: Multiplier 32×32 Implementation Using Multiple Mathblocks
- Example 2: 32×32 Multiplier Implementation Using Single Mathblock

Example1: Multiplier 32×32 Implementation Using Multiple Mathblocks

The following section explains the 32×32 multiplier implementation with multiple mathblocks and shows the performance results.

The 32×32 multiplier is implemented using the following algorithm:

$$A = (AH \times 217) + AL;$$

$$B = (BH \times 217) + BL;$$

$$A \times B = (AH \times 217 + AL) \times (BH \times 217 + BL)$$

$$= ((AH \times BH) \times 234) + ((AH \times BL + AL \times BH) \times 217) + AL \times BL$$

The 32×32 multiplier is implemented efficiently using four mathblocks without using fabric resources to produce 64-bit result as shown in Figure 18 and Figure 19 on page 29. To achieve the best performance results, use mathblock I/O registers.

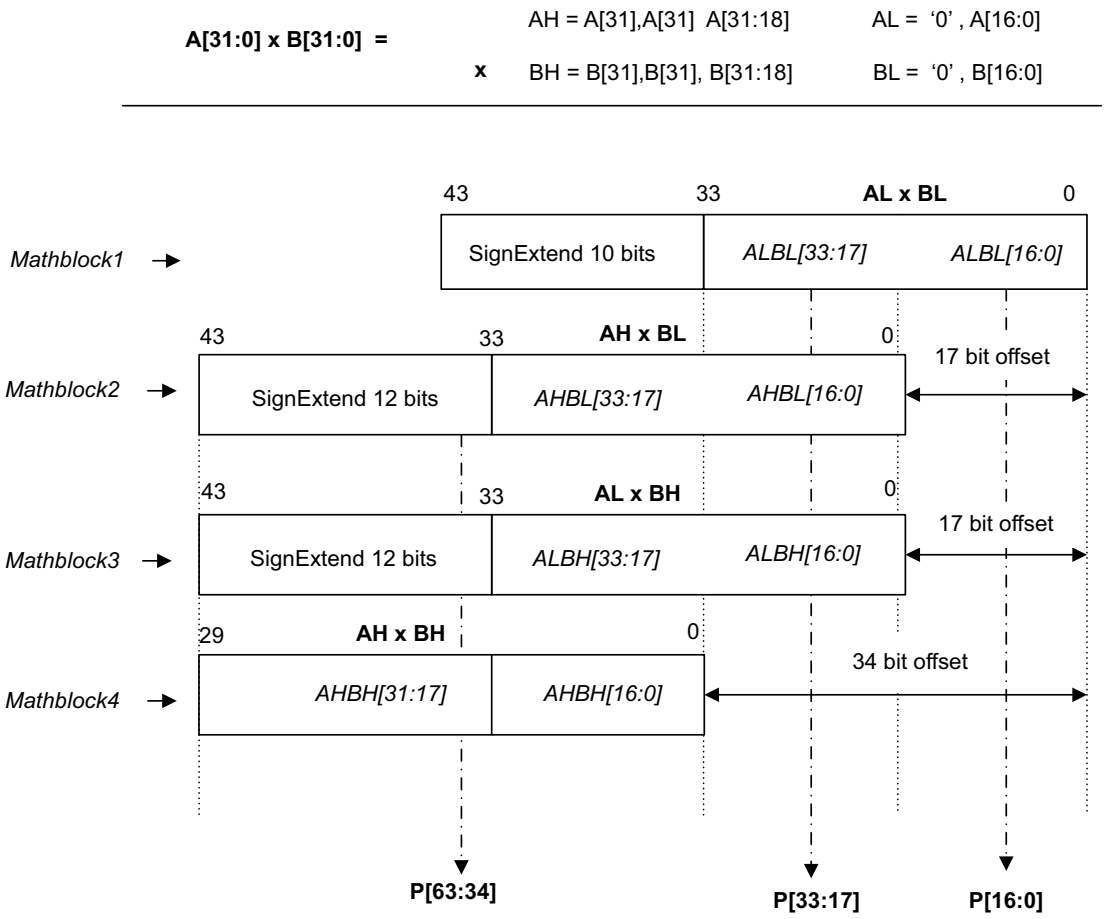

***Figure 18* • 32x32 Multiplication**

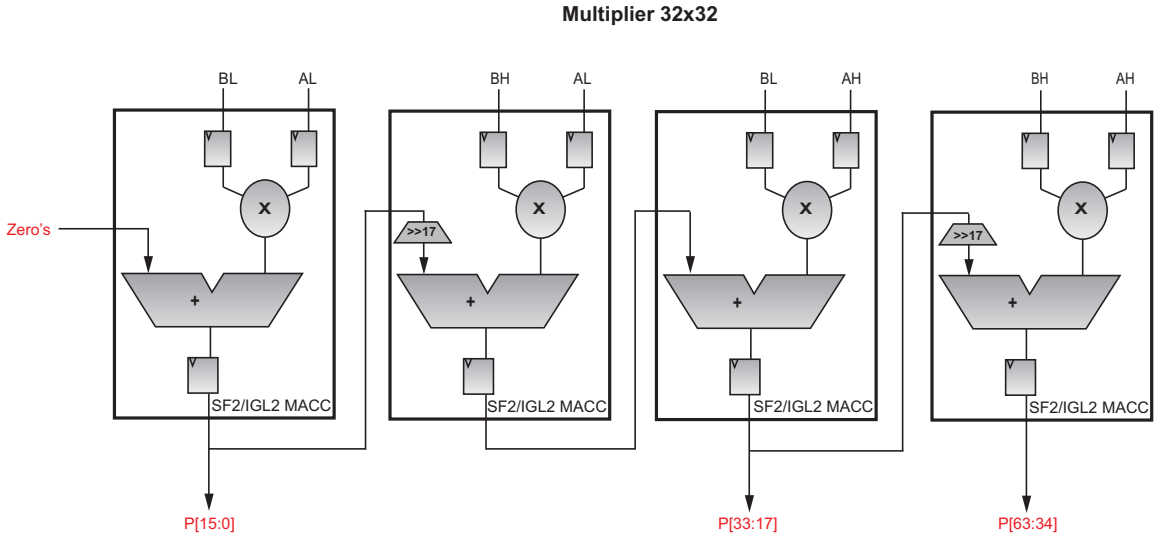

***Figure 19* • Implementation of 32x32 Multiplier using Mathblock**

When implementing using HDL, to infer mathblock I/O registers by synthesis tool, pipeline stages are added at output and input to achieve maximum throughput. In this design, two pipeline stages are added at input and output. Refer to design files for information on implementation of 32x32 multiplier.

Design Files

For the implementation code, refer to the `Mult32x32_multipleMACC.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Mult32x32_multiplieMACC\hdl\Mult32x32_multipleMACC.vhd`

Hardware Configuration

For 32×32 multiplier using a single mathblock, the mathblock is configured to function as: Normal Multiplier Accumulator -> Pn = Pn-1 + CARRYIN + C +/- A0×B0

Normal Multiplier Addsub	-> Pn = D + CARRYIN + C +/- A0×B0 (if ARSHFT is disabled)
	-> Pn = (D>>17) + CARRYIN + C +/- A0×B0 (if ARSHFT is enabled)
Normal Multiplier	-> P = A0×B0

Resource Utilization and Timing Summary

Figure 4 shows the 32×32 multiplier resource utilization when using multiple mathblocks.

Report 4 shows the resources utilized by the m2s050fbga896std device for the Mult32x32_multiple-MACC implementation.

Type	Used	Total	Percentage
COMB	145	56340	0.26
SEQ	290	56340	0.51
IO (W/ clocks)	130	375	34.67
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	4	72	5.56
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
SERDESIF	0	2	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

Report 4 • **Resource Utilization for Multiple Mathblocks**

Table 8 shows the timing summary for the Mult32x32_multipleMACC implemented on the m2s050fbga896std device.

Table 8 • **Timing Summary for 32×32 With Multiple Mathblock**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	2.641	378.644	2.857	350.018	3.981	0.782	4.782	10.858

Example 2: 32×32 Multiplier Implementation Using Single Mathblock

This section explains the 32×32 multiplier implementation with single mathblocks and also shows the performance results.

The 32×32 multiplier is implemented using the same algorithm

$$A \times B = ((AH \times BH) \times 2^{34}) + ((AH \times BL + AL \times BH) \times 2^{17}) + AL \times BL$$

$$= ((AH \times BH) \times 2^{34}) + (AH \times BL \times 2^{17}) + (AL \times BH \times 2^{17}) + AL \times BL$$

In this implementation, the four multiplications are computed using a single mathblock sequentially. The control finite-state machine (FSM) in the design provides inputs to the mathblock sequentially in four consecutive states as shown in Figure 18 on page 28 and appropriately enables the shift operation in the corresponding state. The mathblock used in this design is configured as a normal multiplier accumulator available in the Arithmetic IP core. Refer to the *Hard Multiplier accumulator User Guide* for configuration.

The time taken to generate output = 4 clock cycles for providing inputs

+ 2 clock cycles as the inputs and output is registered

+ 2 clock cycles by mathblock at input and output

= 8 clock cycles.

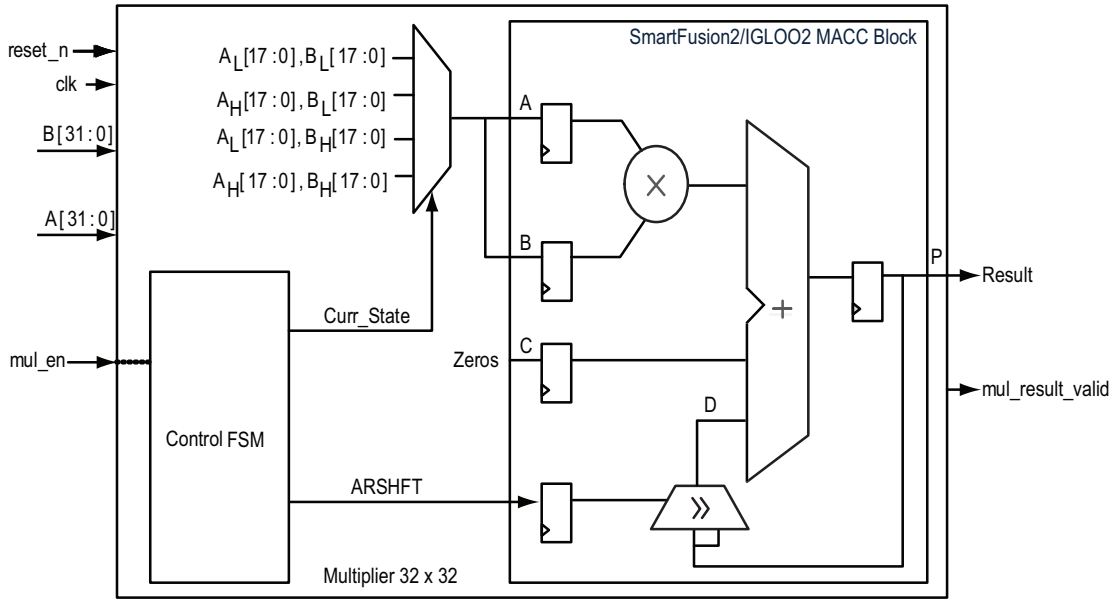

***Figure 20* • Multiplier 32×32 using Single Mathblock**

Design Files

For the implementation code, refer to the `Mult32x32_SingleMACC.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Mult32x32\hdl\Mult32x32_SingleMACC.vhd`

Resource Utilization and Timing Summary

Report 5 shows the resources utilized by the m2s050fbga896std device for the Mult32x32_SingleMACC implementation.

Type	Used	Total	Percentage
COMB	84	56340	0.15
SEQ	141	56340	0.25
IO (W/ clocks)	132	375	35.20
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	1	72	1.39
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
SERDESIF	0	2	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 5* • Resource Utilization for 32×32 Multiplier using Single Mathblock**

Table 9 shows the timing summary for the Mult32x32_SingleMACC implemented on the m2s050fbga896std device.

***Table 9* • Timing Summary for 32×32 Multiplier with Single Mathblock**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	2.641	378.644	2.857	350.018	1.075	0.614	4.679	11.023

Extended Addition

Mathblock has a 3-input adder and supports accumulation of up to 44 bits. In some applications such as floating point multiplication, complex-FFT and filters, high-precision data has to be maintained at every stage. These DSP functions require more than 44-bit addition (extended addition) which can be realized using the SmartFusion2/IGLOO2 mathblock (3-input adder) and fabric logic. The extended addition is implemented by dividing the addition into two parts. The lower part (LSB) of addition is implemented using the SmartFusion2/IGLOO2 mathblock and the upper part (MSB) of addition is implemented using minimal fabric adder logic.

For a 2-input addition, the inputs can be chosen from the following:

- CDIN and C input or
- Multiplier output and CDIN or
- Multiplier output and C input or

For a 3-input addition, the inputs are from the multiplier output, CDIN, and C-input. To perform arithmetic additions, the SmartFusion2/IGLOO2 mathblock provides Carryin signal and Carryout signal for propagating the carry from one mathblock to another mathblock or from mathblock to fabric logic. the mathblock is configured in Normal mode to function as a normal multiplier addsub.

Design Examples

This section shows the extended addition using the following design examples:

- Example 1: 2-Input Signed Extended Addition
- Example 2: 3-Input Signed Extended Addition
- Example 3: 4-Input 42-Bit Adder
- Example 4: 88-Bit Adder/Subtractor

Example 1: 2-Input Signed Extended Addition

This section describes a 2-input extended signed addition—if one operand is wider than 44 bits. It also shows that the 2-input extended signed addition implementation logic with fabric resources is implemented with the multiplier adder.

2-Input Addition

For computing 2-input extended signed addition Z = U + V, with one operand width more than the mathblock output width 44, the logic as shown in Figure 22 should be implemented in fabric.

	U_{m-1}	U_{m-2} ...	U_{n+2}	U_{n+1}	U_n	U_{n-1}	U_{n-2}	...	U_0
+	V_{n-1}	V_{n-1} ...	V_{n-1}	V_{n-1}	V_{n-1}	V_{n-1}	V_{n-2}	...	V_0
	Z_{m-1}	Z_{m-2} ...	Z_{n+2}	Z_{n+1}	Z_n	Z_{n-1}	Z_{n-2}	...	Z_0

Figure 21 • **2-input Extended Signed Addition**

U denotes an m-bit value (where, m > 44) and V is a sign-extended n-bit value (where n < 44). The 2-input extended signed addition is divided in to two parts. The lower part (Sumlower) is computed in the mathblock and the upper part (Sumupper) is computed in the fabric.

Z = (Sumupper, Sumlower)

EQ 4

The lower part of the sum, Z = U + V, is calculated by providing the U[(n-1): 0], V[(n-1): 0] inputs to mathblock, where n = 44 which is the output width of the mathblock.

Sumlower = U[(n-1): 0] + V[(n-1): 0]

EQ 5

The upper part of sum, Z = U + V, is calculated as mentioned below:

Sumupper = U[m: n] + V[m: n] (where U[m: n], V[m: n] are the MSB bits)

EQ 6

V[m: n] = {S, S….S, X},

S = P[n-1] AND X

Where,
P[n-1] is MSB of Sumlower

X is the carryout of the Sumlower (from the mathblock)

S is the sign bit of the adder and (m-n-1) number of S's appended to MSB bits of the V[m:n].

Hardware Implementation

Figure 22 shows the operand width of C as 52 bits and explains the implementation for 2-input extended signed addition. For 3-input addition, mathblock is configured as a multiplier addsub in Normal mode. The upper part and lower part of the sum are mentioned as:

For 52-bit, 2-input extended signed addition,

Sumlower = C[43:0] + A[17:0]×B[17:0]

Sumupper = {C[51:44] + {S, S, S, CARRYOUT}}

Z [51:0] = {Sumupper, Sumlower}

Z [51:0] = {C[51:44] + {S, S, S, CARRYOUT}}, P[43:0]

Where,

S = P[43] AND CARRYOUT

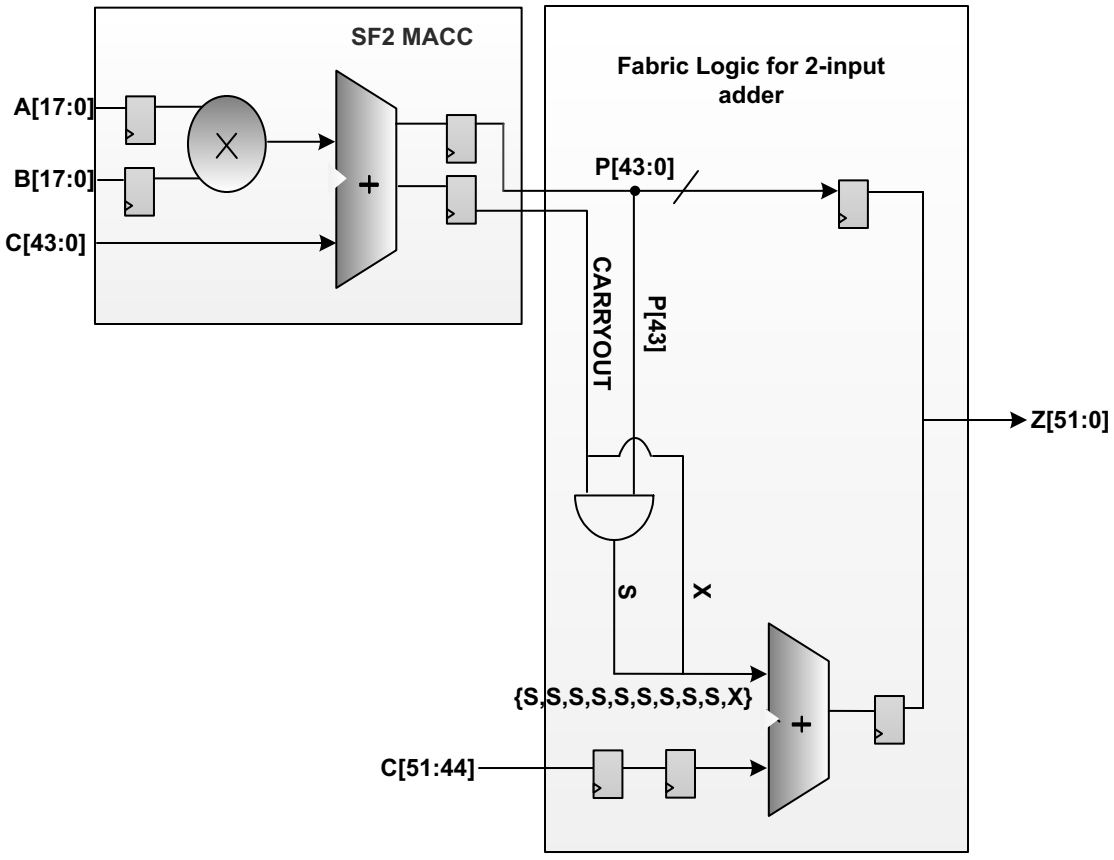

Figure 22 **• Fabric Logic for 2-input Extended Addition using Mathblock**

Design Files

For the implementation code, refer to the `Extended_adder_2_input.vhd` design files. Download the design files from:
`download_folder>\DSP_Reference_Guide_DF\Extended_adder_2_input\hdl\Extended_adder_2_input.vhd`

Resource Utilization and Timing Summary

Report 6 shows the resources utilized by the m2s050fbga896std device for the Extended_adder_2_input implementation.

Type	Used	Total	Percentage
COMB	45	56340	0.08
SEQ	88	56340	0.16
IO (W/ clocks)	142	375	37.87
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	1	72	1.39
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
SERDESIF	0	2	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 6* • Resource Utilization for 2-input Extended Addition with Fabric Resources**

Table 10 shows the timing summary for Extended_adder_2_input implemented on the m2s050fbga896std device.

***Table 10* • Timing summary for 2-input Extended Addition with Fabric Resources**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	2.641	378.644	3.333	300.030	2.567	0.595	5.017	10.947

Example 2: 3-Input Signed Extended Addition

This section explains the 3-input extended signed addition, if one or more operands have a width of more than 44 bits. This section shows the 3-input extended signed addition implementation logic with fabric resources.

3-Input Extended Addition

For performing 3-input extended addition, Z = T + U + V, with two operands having width more than the mathblock input width of 44, the logic shown in Figure 24 should be implemented in fabric

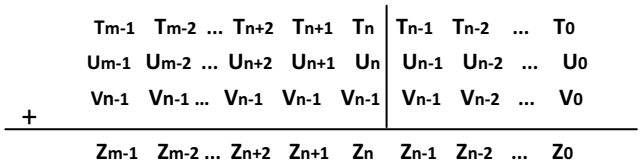

***Figure 23* • 3-input Extended Signed Addition**

T and U are m-bit values (where m > 44) and V is a sign-extended n-bit value (where n < 44). The 3-input extended signed addition is divided in to two parts. The lower part (Sumlower) is computed in the mathblock and the upper part (Sumupper) is computed in the fabric.

Z = {Sumupper, Sumlower}

EQ 7

The lower part of the sum, Z = T + U + V, is calculated by providing the {'0', T[(n-2): 0]}, {'0', U [(n-2}: 0]}, V [(n-1): 0] inputs to the mathblock, where n = 44 that is output width of the mathblock.

Sumlower = {'0', T[(n-2): 0]} + {'0', U[(n-2): 0]} + V[(n-1): 0]

EQ 8

The upper part of sum Z = T + U + V is calculated as shown below

Sumupper = T[m: n-1] + U[m: n-1] + V[m: n]

EQ 9

Where T[m: n], U[m: n], V[m: n] are the MSB bits

V [m: n] = {S, S….S, X, P [n-1]}

S = P[n-1] AND X

Where'

P [n-1] is the MSB bit of the Sumlower

X is the overflow of the Sumlower (from the mathblock),

(m-n-2) number of Ss should be appended in MSB bits of the V[m: n].

Hardware Implementation

Figure 24 shows the operand widths of C and D (52 bits) and explains implementation for 3-input extended signed addition. For 3-input addition, mathblock is configured as multiplier addsub in Normal mode. For 52-bit, 3-input extended signed addition, Sumlower, and Sumupper are calculated as shown below:

Sumlower = P[43:0] = {'0', C[42:0]} + {'0', D [42:0]} + A[17:0]×B[17:0]

Sumupper = {C[51:44] + {S, S, S, CARRYOUT}}

Z [51:0] = {Sumupper, Sumlower}

Z [51:0] = {C[51:43] + D[51:43] + {S, S, S, S, S, S, S, CARRYOUT, P[43]}}, P[42:0]

Where S = P[43] AND CARRYOUT

.

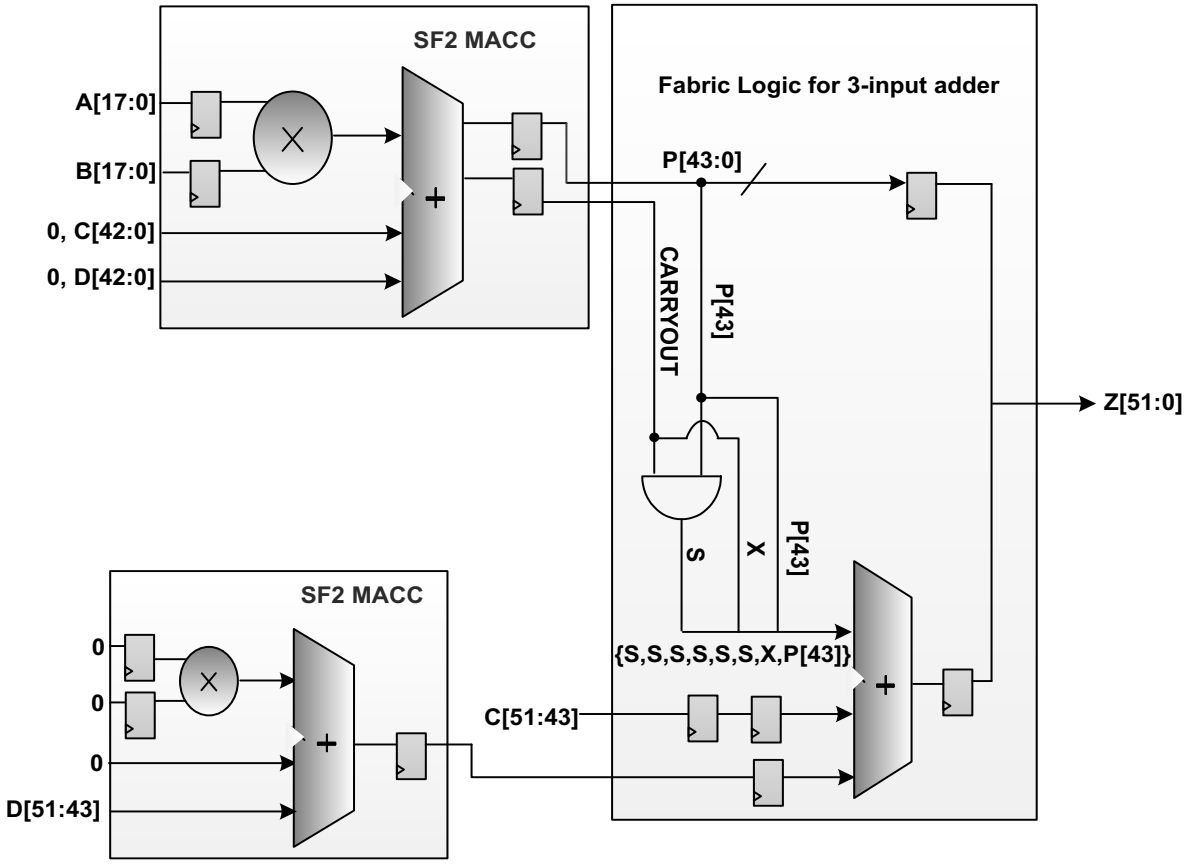

***Figure 24* • 3-input Extended Addition using Mathblock and Fabric Logic**

Design Files

For the implementation code, refer to the `Extended_adder_3_input.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Extended_adder_3_input\hdl\Extended_adder_3_input.vhd`

Resource Utilization and Timing Summary

Report 7 shows the resources utilized by the m2s050fbga896std device for the Extended_adder_3_input implementation

COMB	92	56340	0.16
SEQ	120	56340	0.21
IO (W/ clocks)	194	375	51.73
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	2	72	2.78
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
SERDESIF	0	2	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

Report 7 **• Resource Utilization for 3-input Extended Addition with Fabric Resources**

Example 3: 4-Input 42-Bit Adder

Two SmartFusion2/IGLOO2 mathblocks can be used to build a 4-input adder as shown in Figure 25. First mathblock adder acts as a 2-input adder (that is, 36-bit multiplier output and 44-bit C-input) and the result of first adder output is connected to the CDIN of second mathblock. The second mathblock adder functions as a 3–input adder (that is, 36-bit multiplier output, 44-bit CDIN, and 44-bit C-input). The sum obtained is a 44-bit adder result. In this case, the CARRYOUT is unused.

For proper pipeline balancing at the inputs, external registers are added to the inputs of the second mathblock. Figure 25 shows the 4-input 42-bit adder.

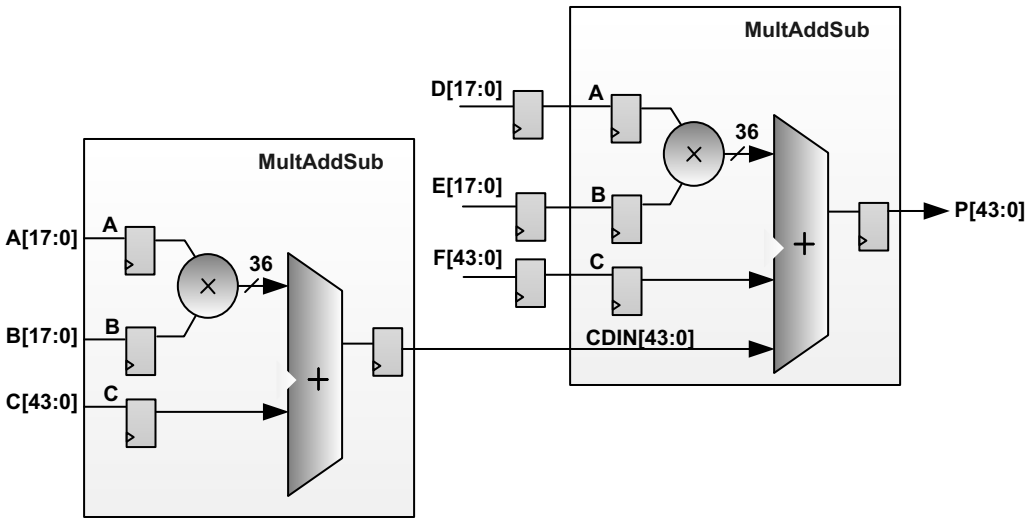

Figure 25 • **4-Input 42-Bit Adder using Mathblocks**

Design Files

For the implementation code, refer to the `Fourinput_42bit_Adder.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Fourinput_42bit_Adder\hdl\Fourinput_42bit_Adder.vhd`

Resource Utilization and Timing Summary

Report 8 shows the resources utilized by the m2s050fbga896std device for the Fourinput_42bit_Adder implementation.

COMB	73	56340	0.13
SEQ	152	56340	0.27
IO (W/ clocks)	251	375	66.93
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	2	72	2.78
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 8* • Resource utilization for 4-Input 42-Bit Adder**

Table 11 shows the timing summary for Fourinput_42bit_Adder implemented on the m2s050fbga896-1 device.

***Table 11* • Timing summary for 4-Input 42-Bit Adder**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
CLK	2.294	435.920	5.000	200.000	1.974	0.777	5.269	10.102

Example 4: 88-Bit Adder/Subtractor

The SmartFusion2/IGLOO2 mathblocks can be cascaded together to implement a large add/subtract function. Figure 26 shows the implementation of a 88-bit adder/subtractor or 88-bit extended addition performed using two mathblocks. The MSB bits addition is performed using the MultAddSub0 mathblock and LSB bits addition is performed using the MultAddSub1 mathblock. The CARRYOUT signal of the MultAddSub0 mathblock is cascaded to CARRYIN of the MultAddSub1 mathblock.

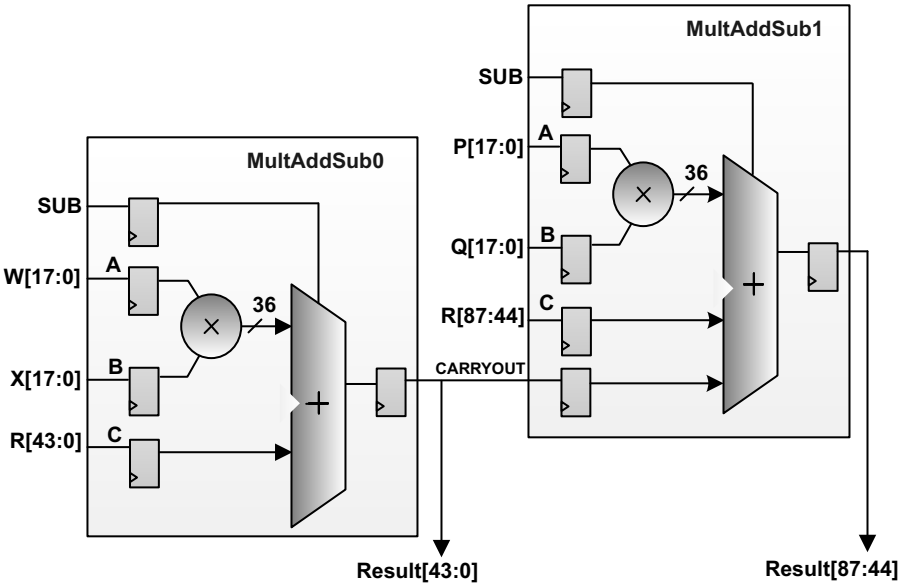

Figure 26 • **88-Bit Adder/Subtractor using Mathblocks**

Design Files

For the implementation code, refer to the `Adder_Sub_88bit.vhd` design files. Download the design files from: `<download_folder>\DSP_Reference_Guide_DF\Adder_Sub_88bit\hdl\Adder_Sub_88bit.vhd`

Resource Utilization and Timing Summary

Report 9 shows the resources utilized by the m2s050fbga896-1 device for the Adder_Sub_88bit implementation.

Type	Used	Total	Percentage
COMB	73	56340	0.13
SEQ	232	56340	0.41
IO (W/ clocks)	250	375	66.67
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	2	72	2.78
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 9* • Resource Utilization for 88-bit Adder/Subtractor**

Table 12 shows the timing summary for Adder_Sub_88bit implemented on the m2s050fbga896-1 device.

***Table 12* • Timing summary for 88-bit Adder/Subtractor**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
CLK	2.842	351.865	5.000	200.000	1.819	0.770	5.233	9.914

44-Bit Counter

The counter is one of the most widely used functions in digital applications. The SmartFusion2/IGLOO2 mathblock can be used as a high-speed counter, with the speed of up to 350 MHz. Figure 27 shows an implementation of 44-bit binary counter using the mathblock. The up counter, down counter, and a loadable counter can be implemented using SmartFusion2/IGLOO2 mathblock. Figure 28 shows the implementation of a count by M, 44-bit counter using the mathblock.

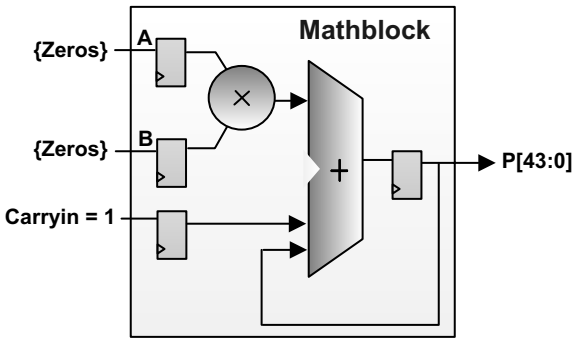

***Figure 27* • Binary Counter using Mathblock**

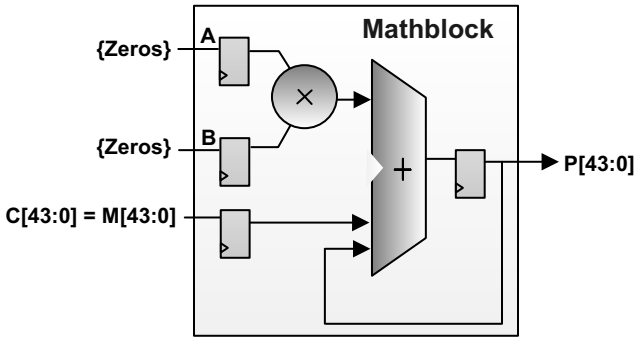

***Figure 28* • Count by M 44-Bit Counter using Mathblock**

Design Files

For the implementation code, refer to the `Counter44_bit.vhd` design files. Download the design files from: `<download_folder>\DSP_Reference_Guide_DF\Counter_44bit\hdl\Counter_44bit.vhd`

Synthesis and P-and-Route Results

Report 10 shows the resources utilized by the m2s050fbga896-1 device for the Counter_44bit implementation.

Type	Used	Total	Percentage
COMB	83	56340	0.15
SEQ	89	56340	0.16
IO (W/ clocks)	93	375	24.80
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	1	72	1.39
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 10* • Resource Utilization for 88-bit Adder/Subtractor**

Table 13 shows the timing summary for Counter_44bit implemented on the m2s050fbga896-1 device.

***Table 13* • Timing summary for 44-bit Counter**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
CLK	3.030	330.033	5.000	200.000	1.804	0.690	5.119	9.160

88-Bit Accumulator

Two SmartFusion2/IGLOO2 mathblocks can be cascaded together to implement a 88-bit accumulator. Here, the CARRYOUT signal is used to cascade the mathblocks. The MultACC0 provides a LSB 44 bits result as output and the MultACC1 provides a MSB 44 bits result as output. The initial input value to the accumulator is the C input of mathblock. Figure 29 shows the implementation of an 88-bit accumulator function.

Algorithm:

Result [87:0] = Result [87:0] + C [87:0]

= Result [87:43] * 2^{43} + Result [43:0] + C [87:43] * 2^{43} + C [43:0]

= Result [87:43] * 2^{43} + C [87:43] * 2^{43} + C [43:0] + Result [43:0]

= {(Result [87:43] + C [87:43]) * 2^{43} + { C [43:0] + Result [43:0]}

{MultACC1} {MultACC0}

EQ 10

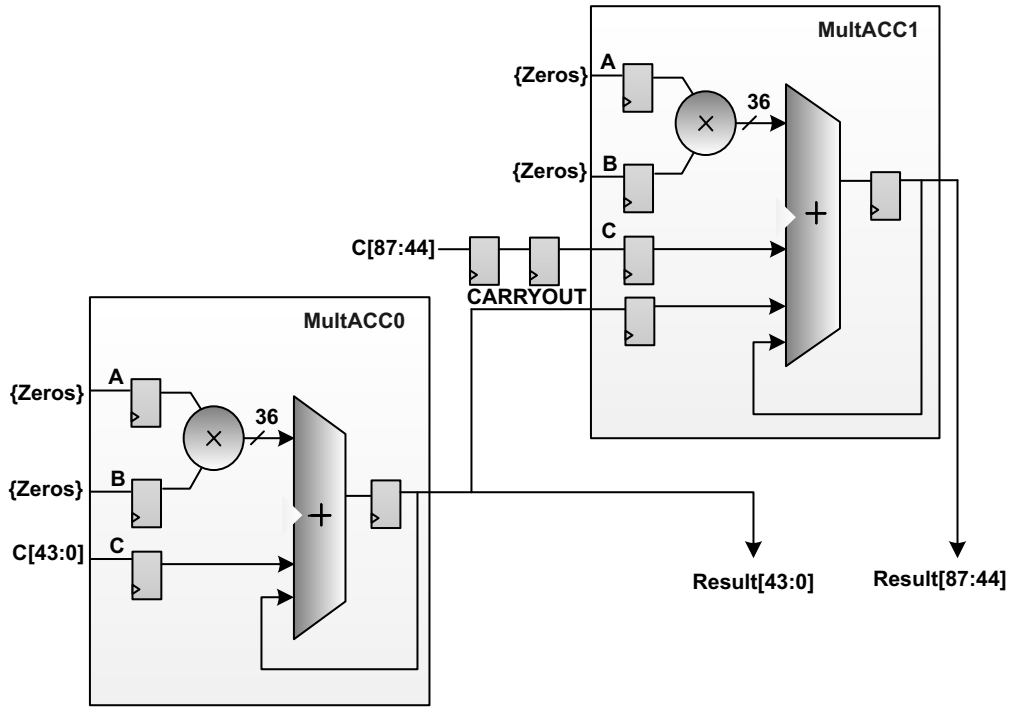

Figure 29 • **88-Bit Accumulator using Mathblocks**

Design Files

For the implementation code, refer to the `Accumulator_88bit.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Accumulator_88bit\hdl\Accumulator_88bit.vhd`

Synthesis and P-and-Route Results

Report 11 shows the resources utilized by the m2s050fbga896-1 device for the Accumulator_88bit implementation.

Type	Used	Total	Percentage
COMB	73	56340	0.13
SEQ	248	56340	0.44
IO (W/ clocks)	178	375	47.47
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	2	72	2.78
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 11* • Resource Utilization for 88-bit Accumulator**

Table 14 shows the timing summary for Accumulator_88bit implemented on the m2s050fbga896-1 device.

***Table 14* • Timing summary for 88-bit Accumulator**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
CLK	2.750	363.636	5.000	200.000	1.109	0.751	4.729	10.041

Filter Applications

A wide variety of filter architectures can be implemented using the Microsemi FPGAs. The SmartFusion2/IGLOO2 mathblock architecture is simple to use, easy to adapt, and build finite impulse response (FIR) filters depending on the requirement of the application.

A FIR filter is a convolution of an input signal and impulse response as shown in EQ 11.

$$Y_n = \sum_{k=0}^{N-1} X(n-k) * h(k)$$

EQ 11

Where X (n-k) represents input signal
H (k) represents impulse response or coefficient
N is the filter length or the filter order

In EQ 11, a set of N coefficients is multiplied by respective N data samples and the products are summed together to form an individual result. The coefficients determine the characteristics of the filter (for example, low-pass filter, band-pass filter, high-pass filter). The FIR equation can be implemented using different architectures (sequential, parallel, and semi-parallel).

This section describes the architecture of the following FIR filters and their implementation using the SmartFusion2/IGLOO2 mathblock:

- MAC FIR Filters
- Parallel FIR Filters
- Semi-Parallel FIR Filters

MAC FIR Filters

The MAC FIR is one of the DSP filter structures that uses a single multiplier with an accumulator to implement a sequential FIR filter. The MAC FIR filter can be implemented using a single SmartFusion2/IGLOO2 mathblock as multiplier-accumulator. This architecture is suitable for applications with slow sample rates and many coefficients. Following are the two MAC FIR filter architectures and their implementation using the SmartFusion2/IGLOO2 mathblock:

- Single MAC FIR Filter
- Symmetric MAC FIR Filter

Single MAC FIR Filter

The Single-MAC FIR filter is useful when the ratio of clock to sample rate is greater than or equal to the filter order/filter length [(f_{clk}/f_{sample})>=Filter order]. Figure 30 shows the general form of multiply–accumulate (MAC) based FIR filter structure utilizing single MAC engine.

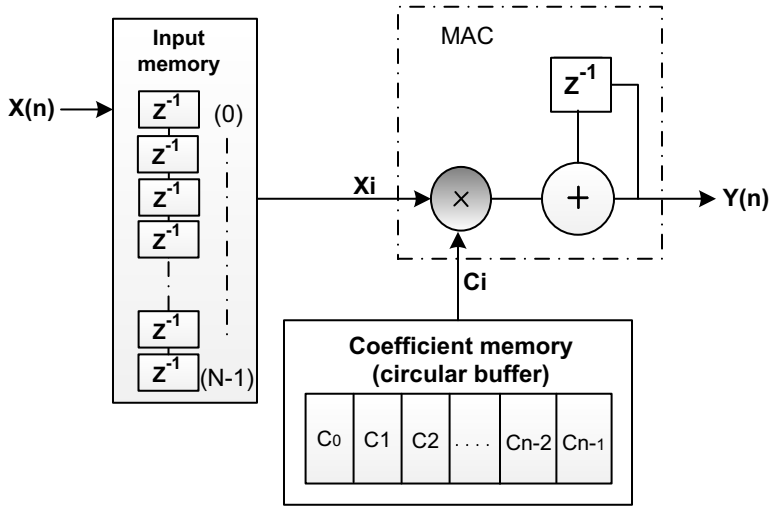

Figure 30 • **N-Tap Single MAC FIR Structure**

Figure 31 shows the implementation of N-Tap single MAC FIR filter using a mathblock and a uSRAM is used for coefficient and input data storage. The control logic is used to generate the required control signals to perform filter operations. Thus single-MAC FIR filter architecture saves the math resources by a factor of the number of filter taps.

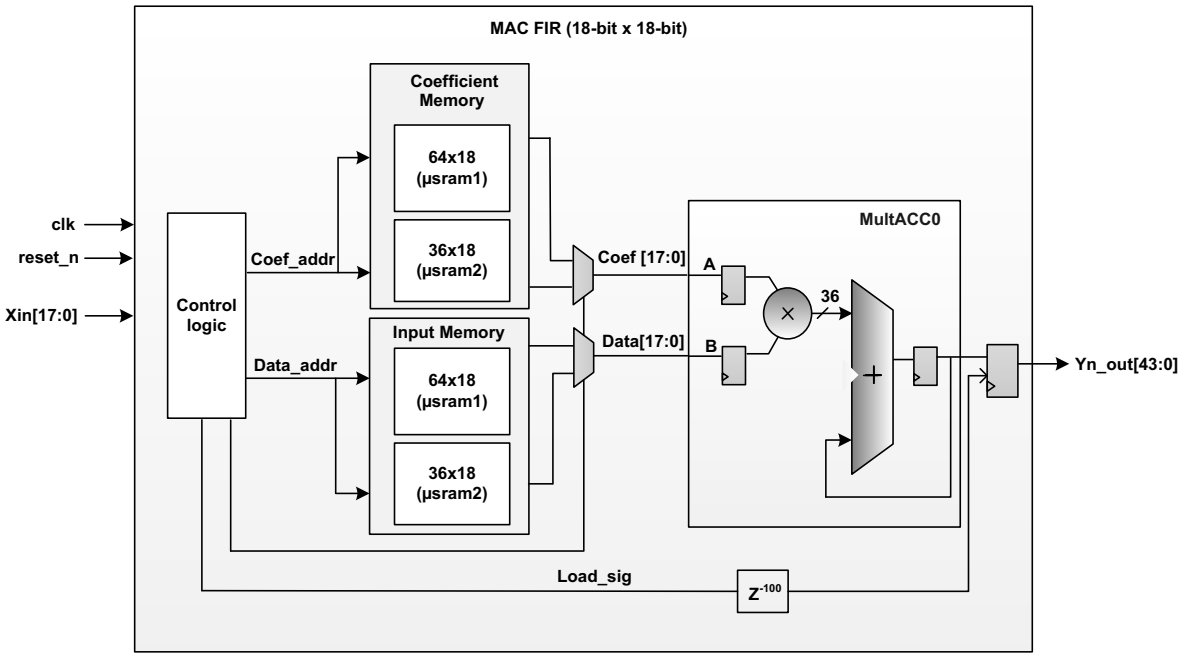

Figure 31 • **N-Tap Single MAC FIR Filter using Mathblock**

The number of clock cycles required to compute N-Tap single MAC FIR filter

(Or)

Maximum input sample rate = Clock speed / (N + 1)

For example, for a 100-Tap MAC FIR filter, maximum input sample rate = Clock speed / (100 + 1)
Therefore, a Single MAC FIR filter input sample rate is same as the number of coefficients in a filter.

Design Files

For the implementation code, refer to the `MAC_FIR.vhd`, design files.Download the design files from the below location,

`<download_folder>\DSP_Reference_Guide_DF\MAC FIR 16-tap\MAC_FIR\hdl\MAC_FIR.vhd`

Resource Utilization and Timing Summary

Report 12 shows the resources utilized by the m2s050fbga896-1 device for the MAC_FIR implementation.

Type	Used	Total	Percentage
COMB	362	56340	0.64
SEQ	396	56340	0.70
IO (W/ clocks)	67	375	17.87
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	4	72	5.56
RAM1K18	0	69	0.00
MACC	1	72	1.39
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 12* • MAC FIR**

Table 15 shows the timing summary for MAC_FIR implemented on the m2s050fbga896-1 device.

***Table 15* • MAC FIR Timing Summary**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	4.000	250.000	5.000	200.000	1.569	0.879	4.633	8.240

Symmetric MAC FIR Filter

Figure 32 shows the general form of MAC based symmetric FIR filter structure utilizing a MAC engine.

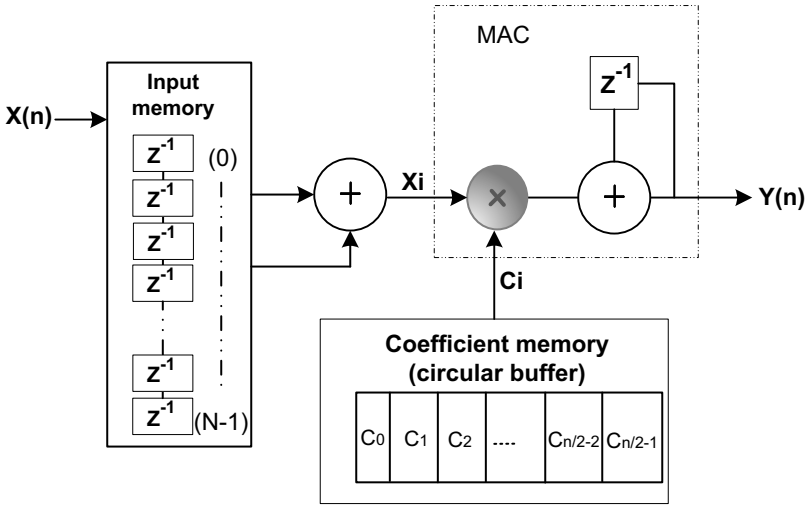

***Figure 32* • Symmetric MAC FIR Filter using Mathblock**

For N-Tap symmetric FIR filter implementation, an extra adder is used for summing up the samples of the symmetric coefficients and the remaining logic remains same as a single-MAC FIR filter as shown in Figure 33. Moreover, only one uSRAM is required for coefficient memory instead of two as used in single MAC FIR filter and thus saving the memory resources.

Yn = (X0*C0) + (X1*C1) + ……+ (Xn-1*Cn-1) + (Xn*Cn)

For Symmetric filter, C0 = Cn, C1 = Cn-1, C2 = Cn-2 ….etc

Yn = (X0 + Xn)*C0 + (X1+ Xn-1)*C1 + … (Xm + Xn-m)*Cm +...

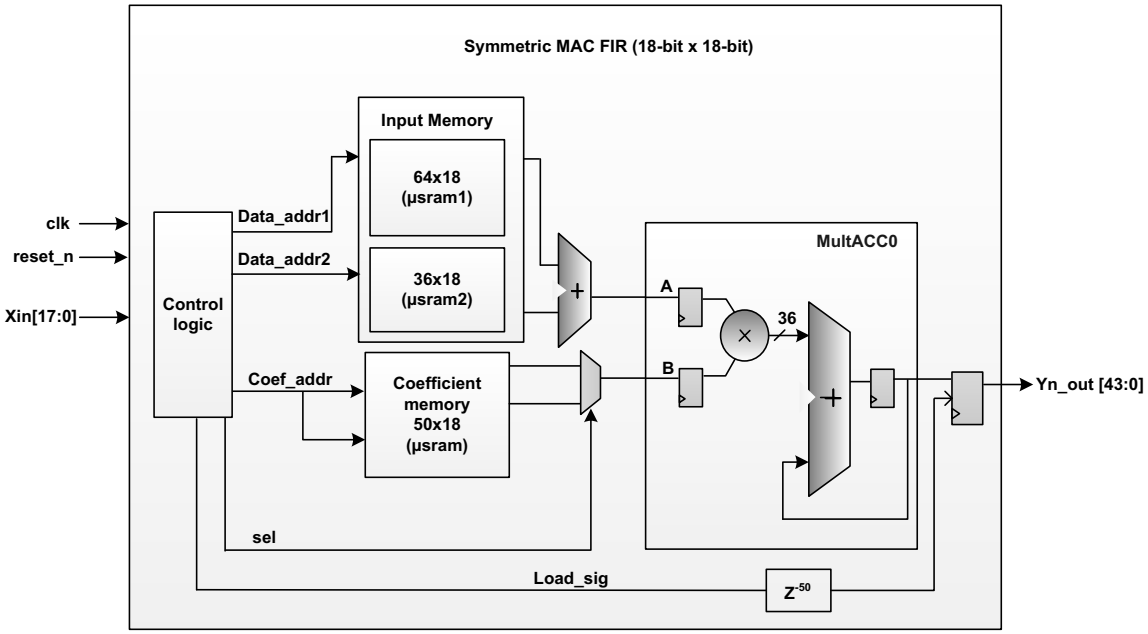

***Figure 33* • Symmetric MAC FIR using Mathblock**

The number of clock cycles required to compute N-Tap symmetric MAC FIR filter

(Or)

Maximum input sample rate = Clock speed / (N/2) + 1

For example:

For a 100-Tap symmetric MAC FIR filter, maximum input sample rate = Clock speed / 50 + 1

Therefore, input sample rate of symmetric MAC filter = 2 * input sample rate of single-MAC FIR filter.

Note: There is a limitation in using symmetric MAC FIR filter. Due to the 1-bit growth from the pre-adder, the input data to the filter must be less than 18 bits to fit into a mathblock, that is, maximum input data width is 17 bits for unsigned, and 16 bits for signed. The pre-adder can also be implemented using another mathblock to save the fabric resources because the fabric adder may limit maximum clock frequency.

Design Files

For the implementation code, refer to the `Symmetric_MAC_FIR.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Symmetric_MAC_FIR_Filter\hdl\Symmetric_MAC_FIR.vhd`

Resource Utilization and Timing Summary

Report 13 shows the resources utilized by the m2s050fbga896-1 device for the Symmetric_MAC_FIR implementation.

Type	Used	Total	Percentage
COMB	424	56340	0.75
SEQ	373	56340	0.66
IO (W/ clocks)	67	375	17.87
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	3	72	4.17
RAM1K18	0	69	0.00
MACC	1	72	1.39
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 13* • Symmetric MAC FIR**

Table 16 shows the timing summary for Symmetric_MAC_FIR implemented on the m2s050fbga896-1 device.

***Table 16* • Symmetric MAC FIR Timing Summary**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	4.305	232.288	5.000	200.000	0.631	0.861	4.678	9.008

Parallel FIR Filters

The Parallel FIR filter uses N multipliers and N-1 adders as shown in Figure 34. The parallel FIR filter can be realized using the SmartFusion2/IGLOO2 mathblocks and is well suited in applications for high sample rate requirements.

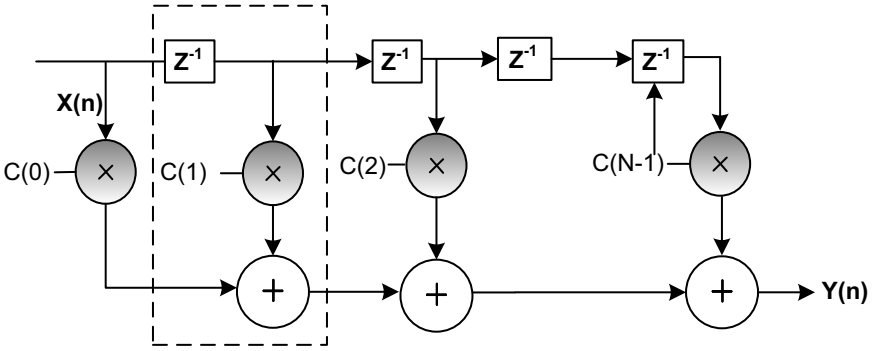

***Figure 34* • Direct Form FIR Filter Structure**

The following Parallel FIR filter architectures and their implementation using the SmartFusion2/IGLOO2 mathblock are described in this section.

- Transpose - Non-symmetry
- Transpose - Symmetry
- Systolic - Non-Symmetry
- Systolic - Symmetry

Transpose - Non-symmetry

The transpose FIR architecture is used in high performance filter applications. This architecture is realized from the Direct Form I structure as shown in Figure 35. In the Transpose architecture, the same input is shared to all the multipliers, thus increasing the fan-out at the input.

For N-tap Transpose FIR filter, the total initial latency taken = (N – 1) clock cycles.

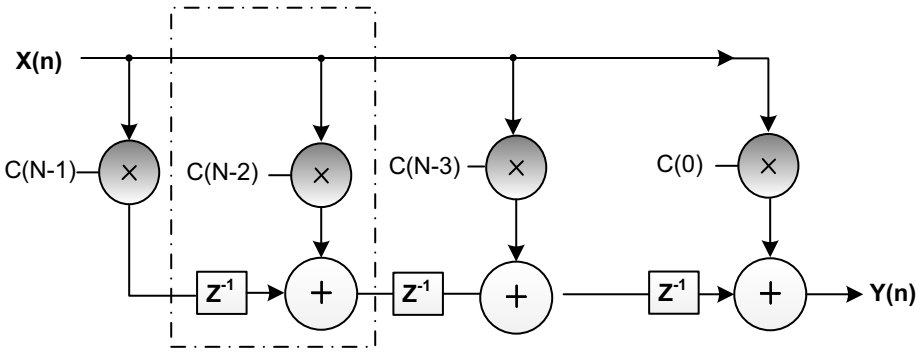

***Figure 35* • N-Tap Non-Symmetric Transpose FIR Filter Structure**

In transpose FIR filter, each multiplier-adder block is realized using one mathblock. Hence, the N-tap transpose FIR filter utilizes only N mathblocks. Figure 36 shows implementation of the 16-tap Non-Symmetric Transpose FIR filter using the SmartFusion2/IGLOO2 mathblocks.

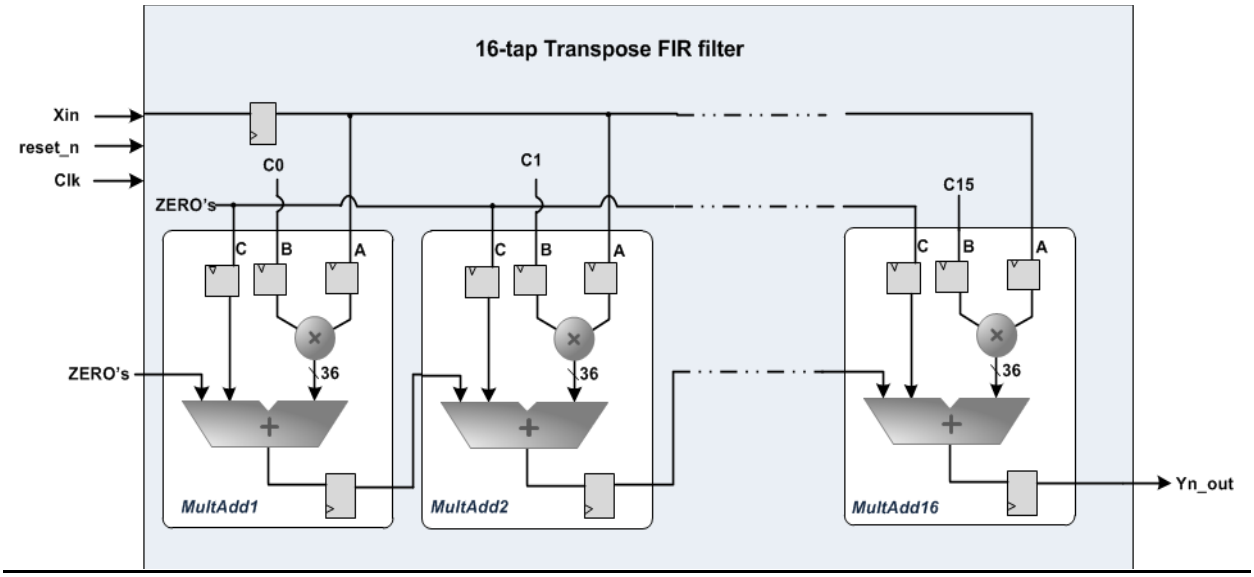

***Figure 36* • 16-Tap Non-Symmetric Transpose FIR Filter using Mathblock**

Design Files

For the implementation code, refer to the `Transpose_FIR.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Transpose_FIR_w_macc\hdl\Transpose_FIR.vhd`

Resource Utilization and Timing Summary

Report 14 shows the resources utilized by the m2s050fbga896-1 device for the Transpose_FIR implementation.

Type	Used	Total	Percentage
COMB	577	56340	1.02
SEQ	594	56340	1.05
IO (W/ clocks)	64	375	17.07
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	16	72	22.22
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

Report 14 • **Transpose FIR Filter**

Table 17 shows the timing summary for Transpose_FIR implemented on the m2s050fbga896-1 device.

Table 17 • **Transpose FIR Filter Timing Summary**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	3.635	275.103	4.000	250.000	0.066	0.829	4.652	8.236

Transpose - Symmetry

The transpose symmetry architecture is same as the transpose architecture but the coefficients are shared to the respective inputs of symmetric taps. Figure 37 on page 56 and Figure 38 on page 56 show the Odd tap and even tap Symmetric Transpose FIR filter architectures.

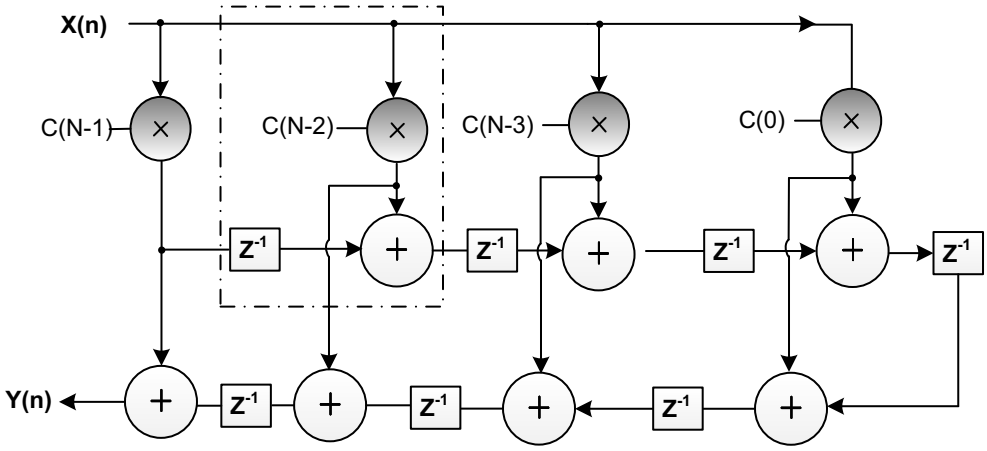

Figure 37 • **ODD TAP Symmetric Transpose FIR Filter Structure**

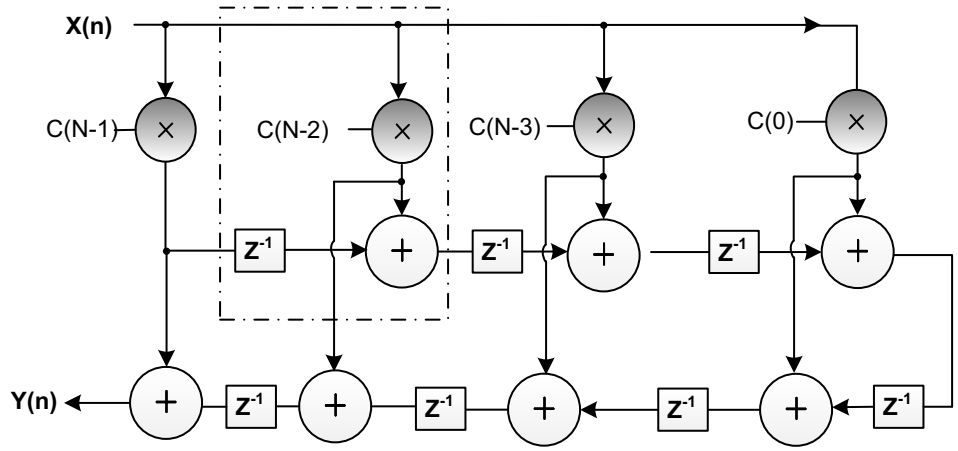

Figure 38 • **Even TAP Symmetric Transpose FIR Filter Structure**

In symmetric transpose FIR filter, each multiplier-adder block is realized using one mathblock. Hence, the N-tap transpose FIR filter utilizes only N mathblocks. Figure 39 shows implementation of a 16-tap Symmetric Transpose FIR filter using the mathblocks.

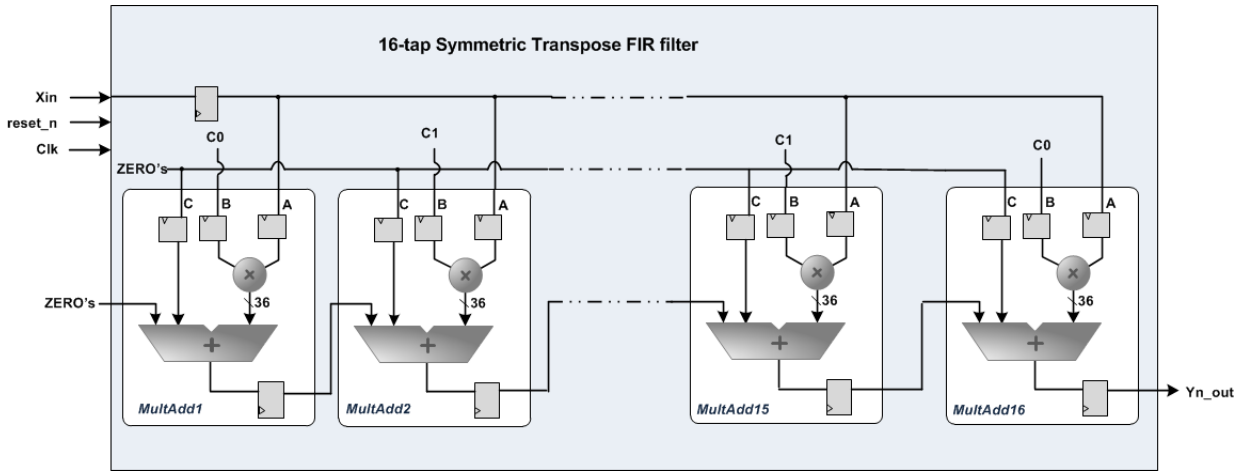

***Figure 39* • Symmetric Transpose FIR Filter using Mathblock**

Design Files

For the implementation code, refer to the `Transpose_Sym_FIR.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Transpose_Sym_FIR\hdl\Transpose_Sym_FIR.vhd`

Resource Utilization and Timing Summary

Report 15 shows the resources utilized by the m2s050fbga896-1 device for the Transpose_Sym_FIR implementation.

Type	Used	Total	Percentage
COMB	577	56340	1.02
SEQ	594	56340	1.05
IO (W/ clocks)	64	375	17.07
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	16	72	22.22
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

Report 15 • **Symmetric Transpose FIR Filter**

Table 18 shows the timing summary for Transpose_Sym_FIR implemented on the m2s050fbga896-1 device.

Table 18 • **Symmetric Transpose FIR Filter Timing Summary**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	3.635	275.103	4.000	250.000	0.066	0.829	4.652	8.236

Systolic - Non-Symmetry

The systolic FIR architecture is used in high performance filter applications. The systolic FIR structure is realized from the Direct form structure by adding an extra pipeline register on each MAC stage (that is, a register at input and a register at output). Hence, the maximum performance is achieved with this architecture.

Figure 40 shows the Non-Symmetric Systolic FIR filter architecture. For tap Non-Symmetric Systolic FIR filter, the total initial latency taken = 2*(N – 1) clock cycles.

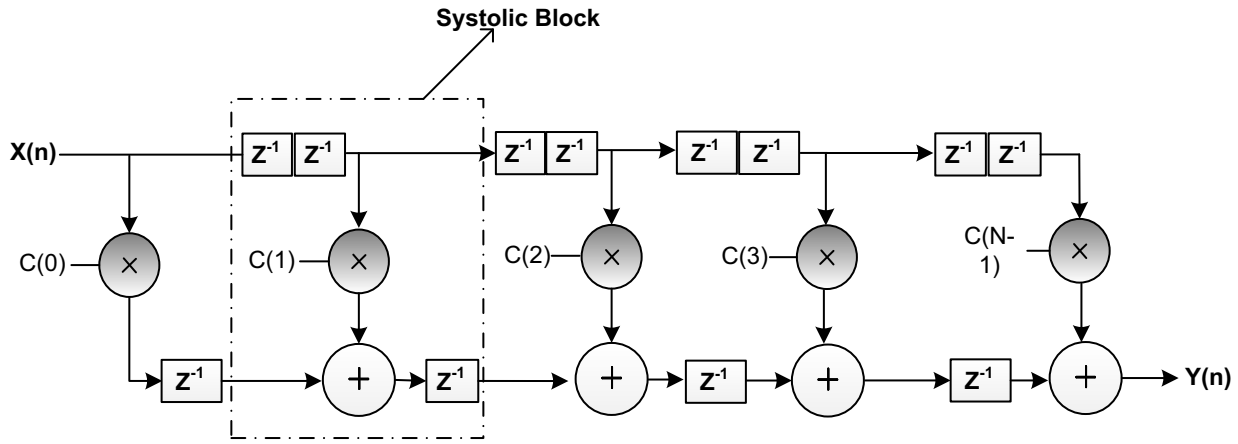

***Figure 40* • Non-Symmetric Systolic FIR Filter Structure**

In systolic FIR filter, each systolic block is realized using a mathblock and two fabric registers at the input stage. Hence, the n-tap systolic FIR filter utilizes n mathblocks and minimum 2*(n-1) fabric registers. Figure 41 shows implementation of 16-tap Non-Symmetric Systolic FIR filter using mathblocks.

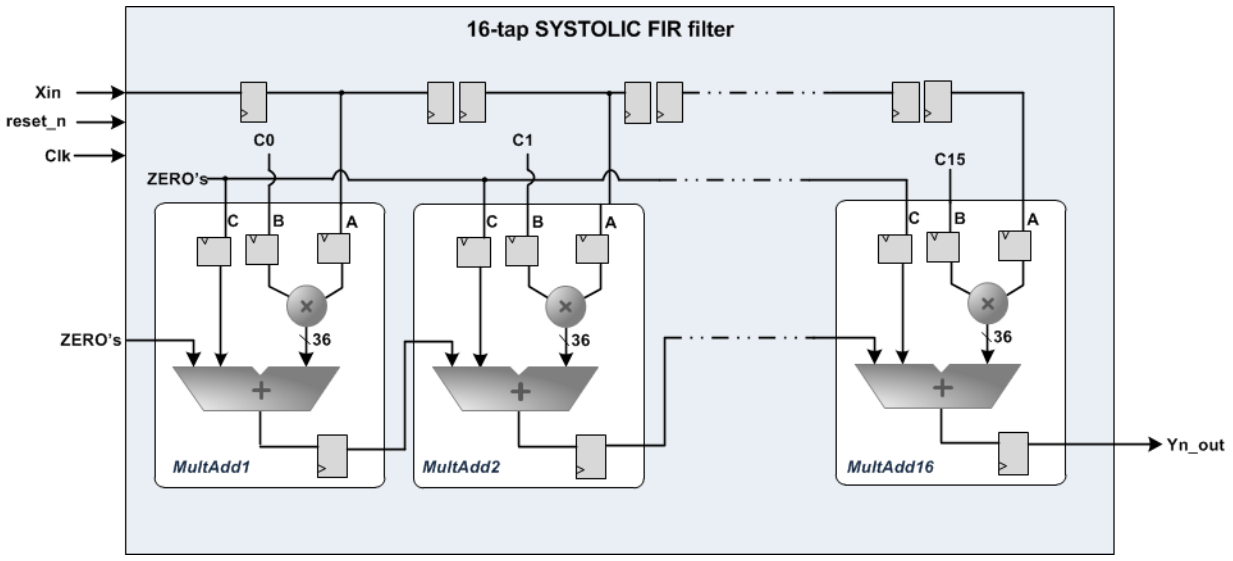

***Figure 41* • 16-Tap Non-Symmetric Systolic Parallel FIR Filter using Mathblock**

Design Files

For the implementation code, refer to the `Systolic_FIR_Filter.vhd`, design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\SystolicFIR_Filter\Systolic_FIR_Filter\hdl\Systolic_FIR_Filter.vhd`

Resource Utilization and Timing Summary

Report 16 shows the resources utilized by the m2s050fbga896-1 device for the Systolic_FIR_Filter implementation.

Type	Used	Total	Percentage
COMB	577	56340	1.02
SEQ	1134	56340	2.01
IO (W/ clocks)	64	375	17.07
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	16	72	22.22
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 16* • Systolic FIR Filter**

Table 19 shows the timing summary for Systolic_FIR_Filter implemented on the m2s050fbga896-1 device.

***Table 19* • Systolic FIR Filter Timing Summary**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
Clk	2.245	445.434	4.000	250.000	-0.024	0.970	5.292	9.414

Systolic - Symmetry

In the symmetric systolic architecture, each systolic can be realized using a fabric pre-adder and a mathblock. A pre-adder is used to sum-up two inputs for symmetric coefficients (for example, C0=cn-1). An n-tap symmetric FIR filter uses N/2 multipliers, N/2-1 adders, and N/2 pre-adders. Figure 42 and Figure 43 show the even-tap and odd tap Symmetric Systolic FIR filter.

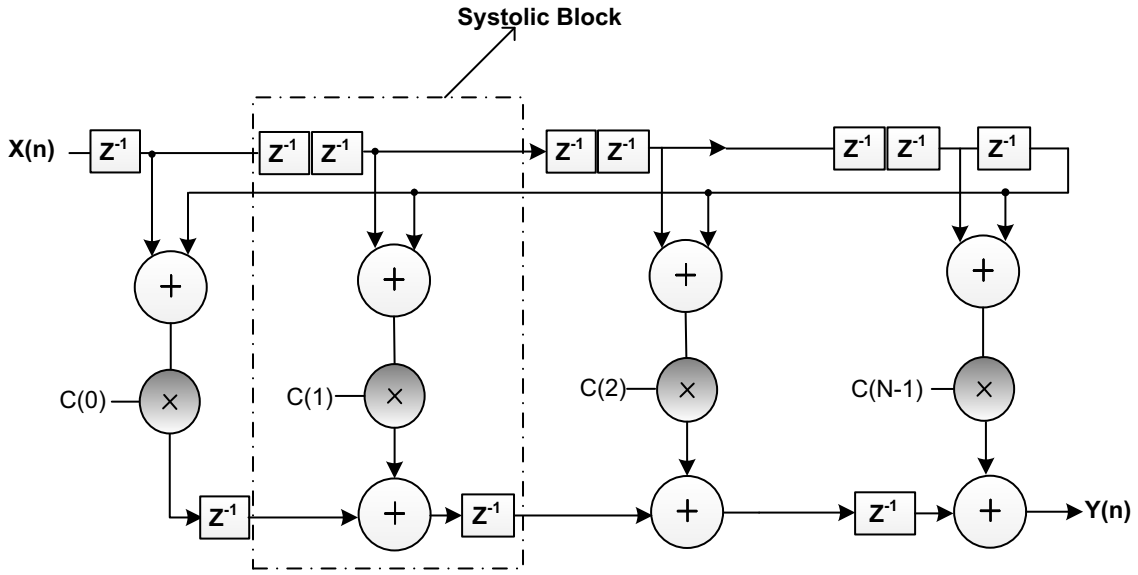

***Figure 42* • Even-tap Symmetric Systolic FIR Filter Structure**

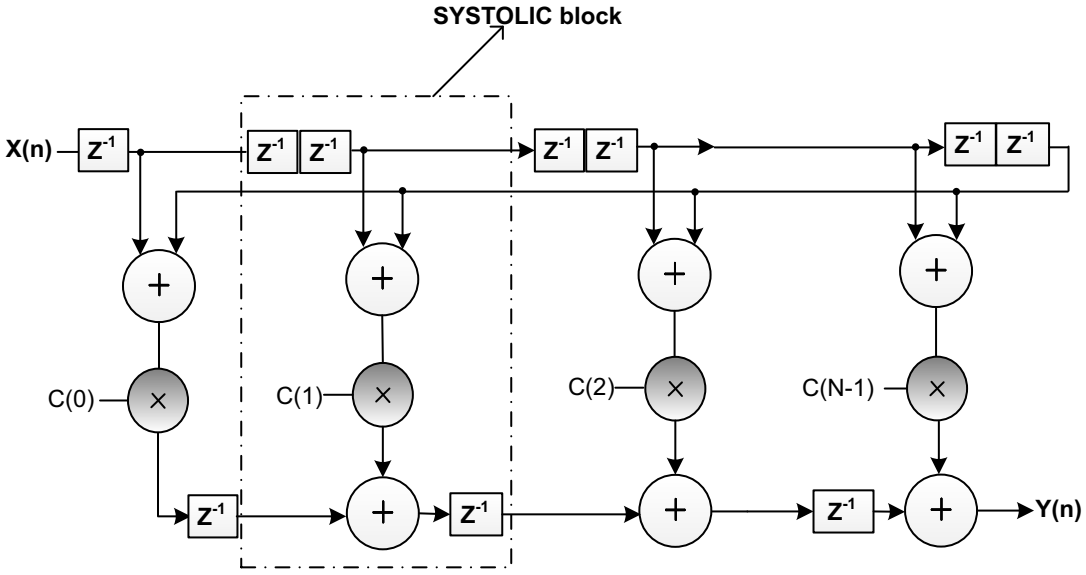

***Figure 43* • ODD-tap Symmetric Systolic FIR Filter Structure**

In symmetric systolic FIR filter, each systolic block is realized using a mathblock, a fabric pre-adder and two fabrics registers at the input stage. Hence, the N-tap systolic FIR filter utilizes N/2 mathblocks and N/2 fabric pre-adders. Thus, this architecture utilizes half the mathblock resources when transposed with symmetric architecture. Figure 44 and Figure 45 on page 63 show the implementation of 16-tap Symmetric Systolic FIR filter using the SmartFusion2/IGLOO2 mathblocks.

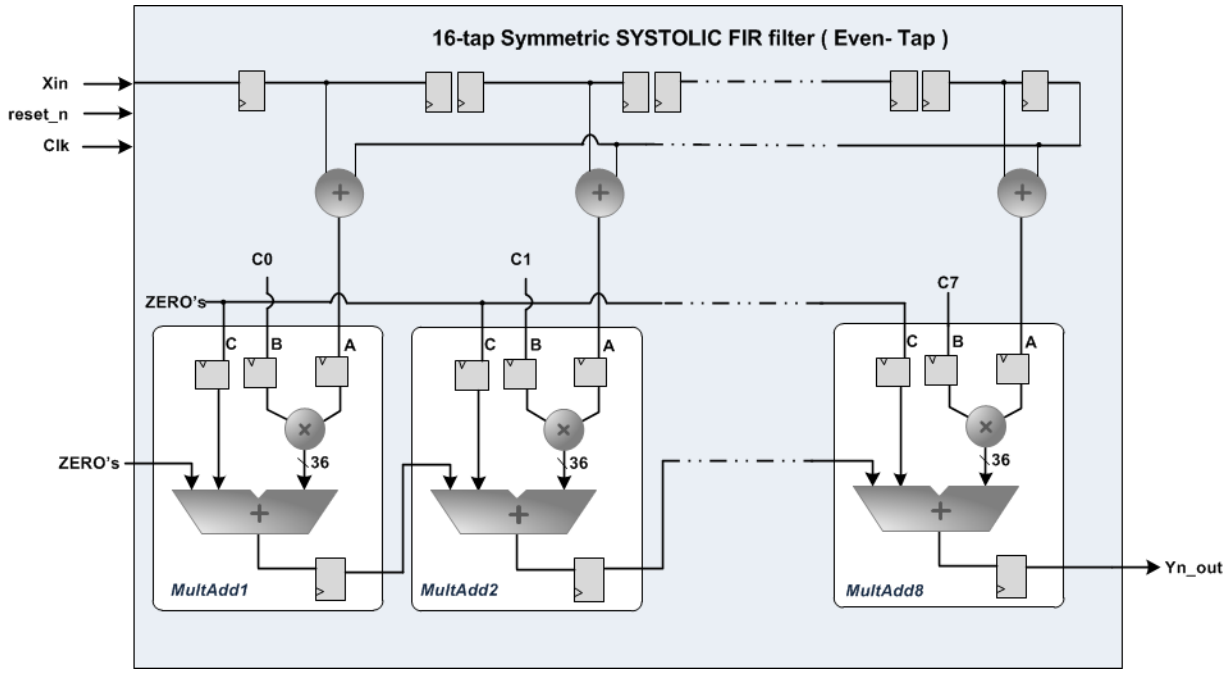

***Figure 44* • 16-Tap Even-Tap Symmetric Systolic FIR Filter using Mathblock**

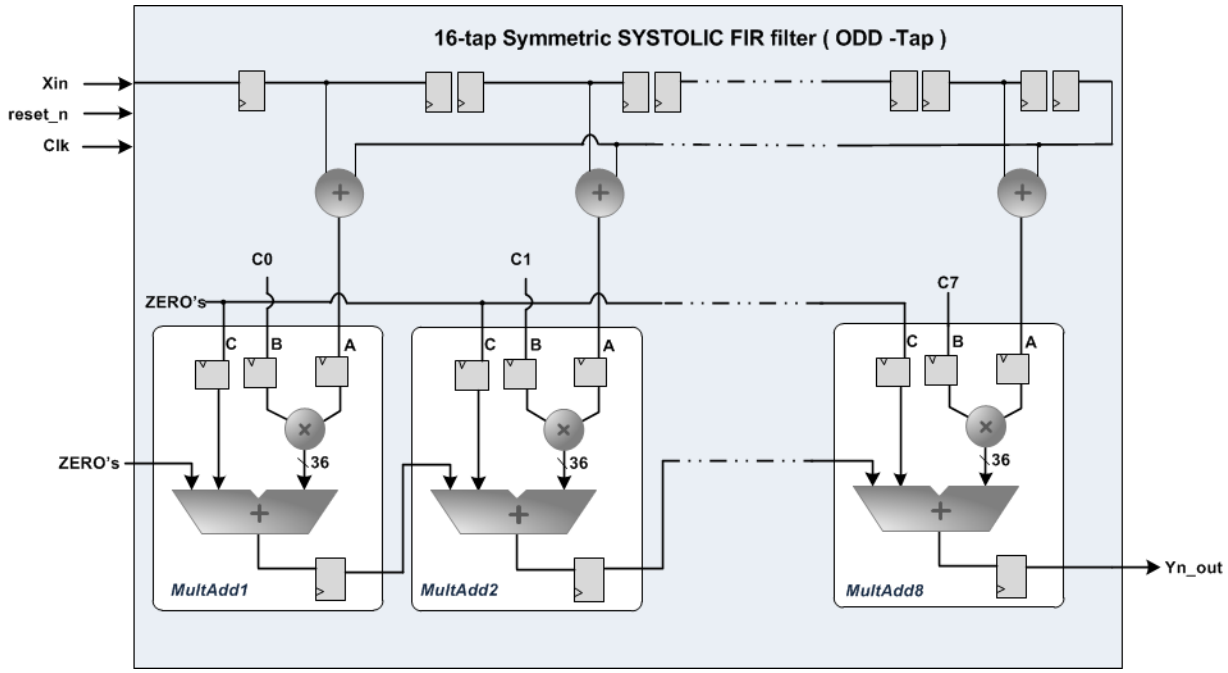

***Figure 45* • 16-Tap ODD-Tap Symmetric Systolic FIR Filter using Mathblock**

Design Files

For the implementation code, refer to the `Systolic_Symmetric_FIR.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\Systolic_Symmetric_FIR\hdl\Systolic_Symmetric_FIR.vhd`

Resource Utilization and Timing Summary

Report 17 shows the resources utilized by the m2s050fbga896-1 device for the Systolic_Symmetric_FIR implementation.

Type	Used	Total	Percentage
COMB	433	56340	0.77
SEQ	722	56340	1.28
IO (W/ clocks)	64	375	17.07
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	8	72	11.11
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 17* • 16-Tap Even-Tap Systolic FIR Filter**

Table 20 shows the timing summary for Systolic_Symmetric_FIR implemented on the m2s050fbga896-1 device.

***Table 20* • 16-Tap Even-Tap Systolic FIR Filter Timing Summary**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	2.729	366.435	4.000	250.000	0.940	0.818	4.927	8.506

CoreFIR is available in Libero catalog which supports transpose and systolic architectures under fully enumerated (Parallel FIR architectures). Refer to CoreFIR Handbook for more information on the usage and configuration.

Semi-Parallel FIR Filters

The Semi-Parallel (folded) single rate FIR is a hybrid architecture and is commonly used to use the mathblocks efficiently according to the design requirements. It utilizes minimal number of MAC blocks that are sufficient to keep up with an average input sample rate. In semi-parallel FIR filter, the folding factor decides the number of coefficients for each MAC.

For example, for a filter length, L and folding factor, m, the semi-parallel FIR architectures consume (L/m) multipliers, (L/m) adders and 1 accumulator. Figure 46 shows the Semi-parallel FIR filter architecture.

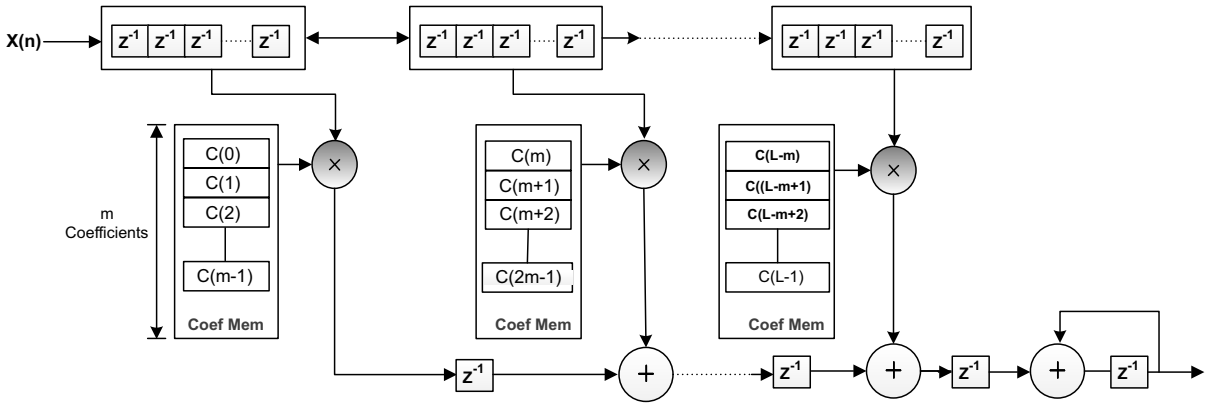

***Figure 46* • Semi-parallel FIR Filter Structure**

In this design, each MACC is realized using mathblock and uses uSRAM for coefficients and input data storage. Each coefficient from uSRAM and input from uSRAM is appropriately fed to each mathblocks and intermediate sums are accumulated to generate final result. Figure 47 shows implementation of a 8-tap two Mathblock FIR filter.

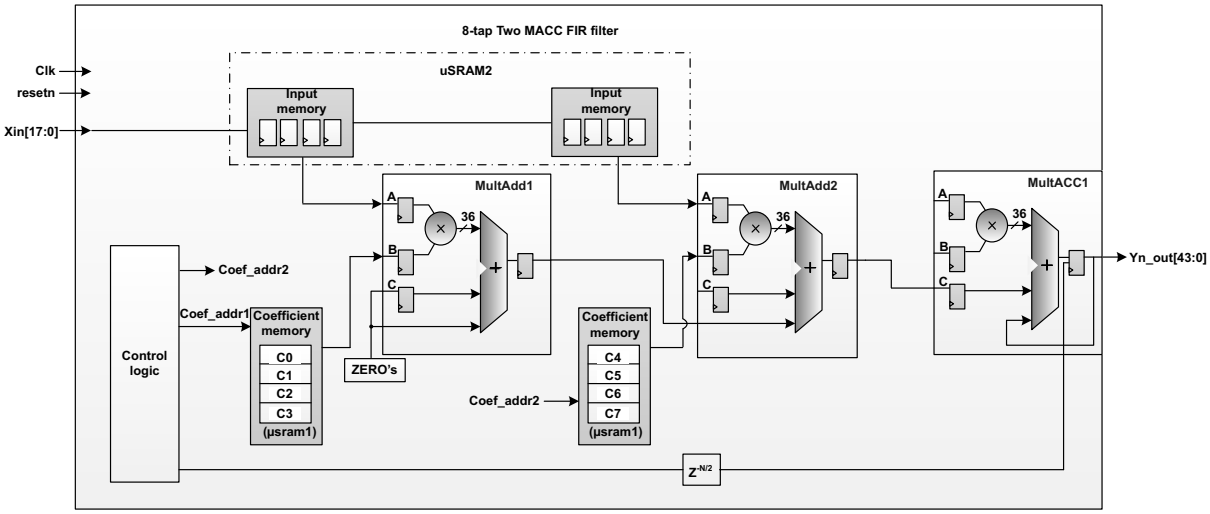

***Figure 47* • 8-Tap Semi-Parallel FIR Filter Structure using Mathblock**

In a similar manner, a 16-tap four MACC FIR filter can be implemented as shown in Figure 48. In this implementation, five mathblocks are required for computing the final result.

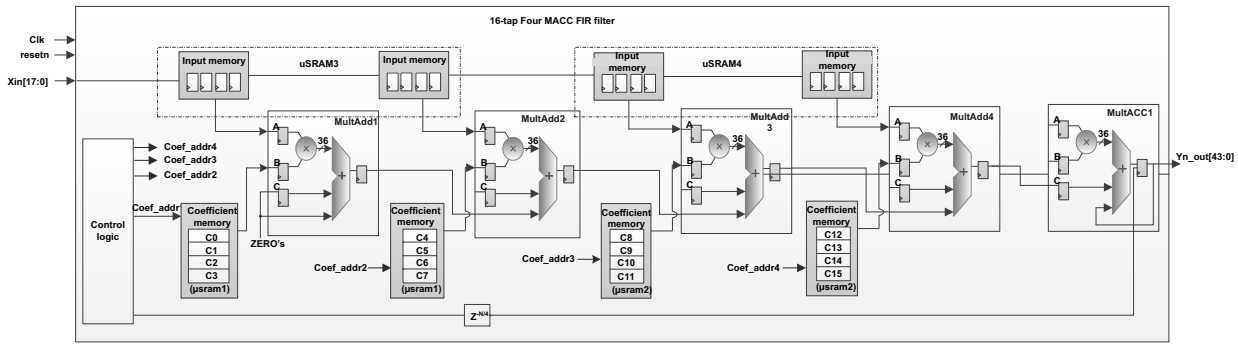

***Figure 48* • 16-Tap Semi-Parallel FIR Filter using Mathblock**

Design Files

For the implementation code, refer to the `TwoMult_8_Tap_FIR.vhd` design files. Download the design files from:

`<download_folder>\DSP_Reference_Guide_DF\TwoMultiplier8-Tap FIR\TwoMult_8Tap_FIR\hdl\TwoMult_8_Tap_FIR.vhd`

Resource Utilization and Timing Summary

Report 18 shows the resources utilized by the m2s050fbga896-1 device for the TwoMult_8_tap_FIR implementation.

Type	Used	Total	Percentage
COMB	201	56340	0.36
SEQ	293	56340	0.52
IO (W/ clocks)	65	375	17.33
Differential IO	0	187	0.00
GLOBAL	2	16	12.50
RGB	2	1088	0.18
RAM64x18	2	72	2.78
RAM1K18	0	69	0.00
MACC	3	72	4.17
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
CCC	0	6	0.00
MSS	0	1	0.00
FDDR	0	1	0.00
SYSCTRL	0	1	0.00

***Report 18* • 8-Tap two Mult FIR Filter**

Table 21 shows the timing summary for TwoMult_8_tap_FIR implemented on the m2s050fbga896-1 device.

***Table 21* • 8-Tap Two Mult FIR Filter Timing Summary**

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Min Clock-To-Out (ns)	Max Clock-To-Out (ns)
clk	4.000	250.000	4.000	250.000	0.882	0.887	4.646	8.176

Transform Applications

Fast Fourier Transform

Fast fourier transform (FFT) is used in DSP applications such as digital communication, video/audio processing, industrial control and bio-medical processing. This transform can be designed in the SmartFusion2/IGLOO2 devices using the inbuilt mathblock and memory blocks (LSRAM and uSRAM). This section describes the basic theory on FFT transform and the FFT IP core available in the Libero SoC software.

The Complex FFT transforms two N point time domain signals into two N point frequency domain signals. The complex signal has two parts, real part and imaginary part.

The FFT of N complex data points *x*(n) is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n) W^{\frac{nk}{N}}$$

EQ 12

Where,
k = 0, 1, 2 …
N-1 and $W_N = e^{-2\pi/N}$
W_N is the twiddle factor or coefficient.
Radix-2 FFT is the widely used architecture in FFT implementation. Radix 2 FFT includes butterfly structure that consists of complex adder, subtraction, and a multiplier for the twiddle factors. Figure 49 shows the simple radix2 butterfly structure and Figure 50 shows the Microsemi mathblock architecture for complex multiplier used for butterfly computations.

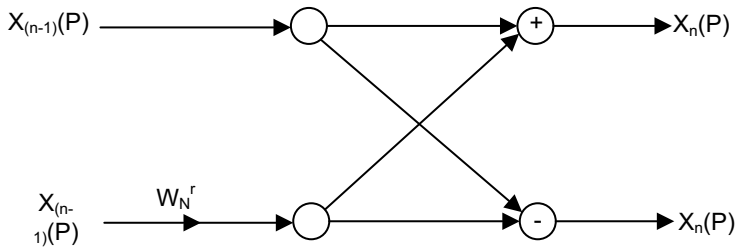

***Figure 49* • Radix-2 Butterfly Structure**

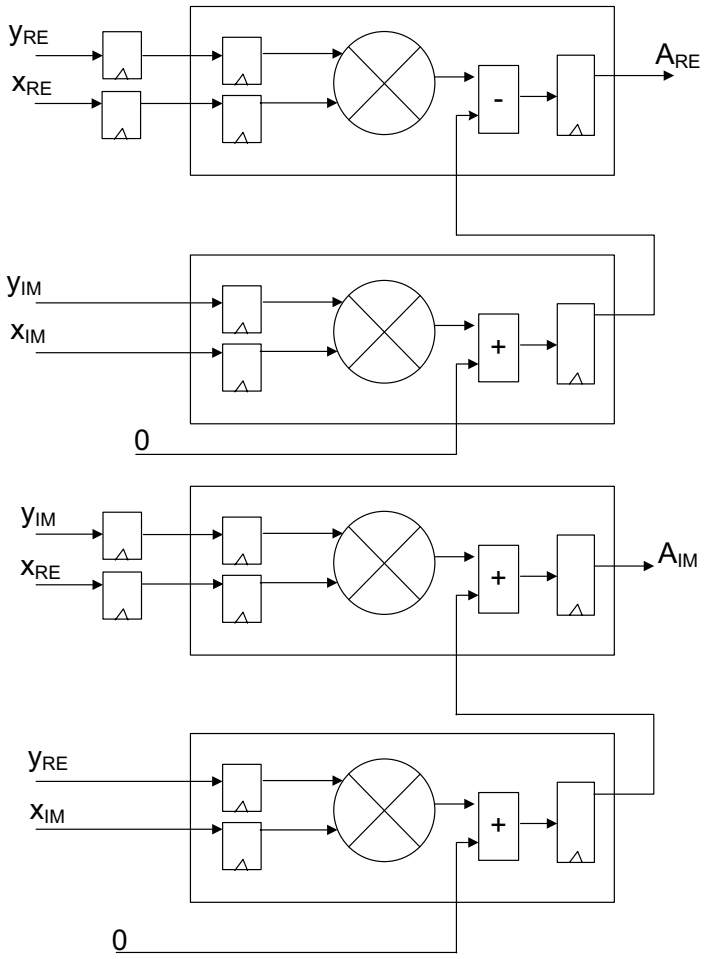

Figure 50 • **Radix-2 Butterfly using Mathblock**

Microsemi has the CoreFFT IP available in the Libero SoC IP catalog, which is highly parameterizable, area efficient and high Performance MAC based FFT, optimized for SmartFusion2/IGLOO2 devices. The CoreFFT has two implementations, Radix-2 decimation-in-time place architecture and radix-2^2 decimation in frequency streaming FFT. CoreFFT supports both the forward and inverse transforms with a length of 2^n, where $5 \le n \le 13$. The main features of CoreFFT are shown in Table 21.

Table 21 • **Core FFT Features**

Feature	In-Place	Streaming
Transform sizes	32-, 64-, 128-, 256-, 512-, 1024-, 2048-, 4096-, and 8192-point	16-, 32-, 64-, 128-, 256-, 512-, and 1024-point
Forward and inverse FFT	Yes	Yes
Input data bit width	8 – 32	8 – 32
Twiddle factor bit width	8 – 32	8 – 32
Input/output data format	Two's complement	Two's complement
Natural output sample order	Yes	Optional

Table 21 • **Core FFT Features (continued)**

Feature	In-Place	Streaming
Conditional block floating point scaling	Yes	No
Pre-defined scaling schedule	No	Yes
Optional minimal or buffered memory configurations	Yes	No
Embedded RAM-block based twiddle look-up table (LUT)	Yes	Yes
Support for refreshing twiddle look-up tables	Yes	Yes
Handshake signals to facilitate easy interface to the user circuitry	Yes	Yes
Run-time forward/inverse transform configuration	No	Yes

Resource Utilization and Timing Summary

For more information on CoreFFT refer to *CoreFFT handbook*.

Appendix 1 – Design Files

The design files (DF) can be downloaded from the Microsemi® SoC Products Group website:

www.microsemi.com/soc/download/rsc/?f=DSP_Reference_Guide_DF

The design file consists of example projects in vhdl. Refer to the readme.txt file that is included in the design file for the directory structure.

List of Changes

The following table lists critical changes that were made in each revision of the reference guide.

Revision	Changes	Page
Revision 2 December 2014	Removed all instances of and references to M2S100 and M2GL100 device from Table 1 (SAR 62858).	5
Revision 1 June 2014	First Release	NA

Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call 800.262.1060
From the rest of the world, call 650.318.4460
Fax, from anywhere in the world, 408.643.6913

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Customer Support website (www.microsemi.com/soc/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the website.

Website

You can browse a variety of technical and non-technical information on the SoC home page, at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
E-mail: sales.support@microsemi.com

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense and security, aerospace, and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs, and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 3,400 employees globally. Learn more at www.microsemi.com.

50200442-2/12.14