Application Note AC398

Implementation of 9x9 Multiplications, Wide-Multiplier, and Extended Addition Using IGLOO2 and SmartFusion2 Mathblock - Libero SoC v11.6

Table of Contents

Purpose

This application note highlights the design guidelines and different implementation methods to achieve better performance results while implementing wide-multipliers, 9-bit×9-bit multiplications, and extended addition with the IGLOO®2 field programmable gate array (FPGA) and SmartFusion®2 system-on-chip (SoC) FPGA mathblock (MACC). The 9-bit×9-bit multiplications, wide-multiplier, and extended addition are ideal for applications with high-performance and computationally intensive signal processing operations. Some of them are finite impulse response (FIR) filtering, fast fourier transforms (FFTs), and digital up or down conversion. These functions are widely used in video processing, 2D or 3D image processing, wireless, industrial applications, and other digital signal processing (DSP) applications.

Introduction

The IGLOO2 and SmartFusion2 mathblock architecture has been 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 result in routing improvements. The IGLOO2 and SmartFusion2 mathblock has a variety of features for fast and easy implementation of many basic math functions. The high-speed multiplier (9×9, 18×18), adder or subtracter, and accumulator in mathblock delivers high speed math functions. For more information on IGLOO2 and SmartFusion2 mathblock, refer to the *UG0445: IGLOO2 FPGA and SmartFusion2 SoC FPGA Fabric User Guide* and for usage of mathblock refer to the *Inferring Microsemi SmartFusion2 MACC Blocks Application Note*.

This application note explains the design considerations and different methods for implementing the following:

- Using 9x9 Multiplier Mode
- Wide-Multiplier
- Extended Addition

References

The following documents are referenced in this document.

- *UG0445: IGLOO2 FPGA and SmartFusion2 SoC FPGA Fabric User Guide*
- *Inferring Microsemi SmartFusion2 MACC Blocks Application Note*
- *IGLOO2/SmartFusion2 Hard Multiplier AddSub Configuration User Guide*
- *IGLOO2/SmartFusion2 Hard Multiplier Accumulator Configuration User Guide*
- *IGLOO2/SmartFusion2 Hard Multiplier Configuration User Guide*

Design Requirements

Table 1 shows the design requirements.

Table 1 • **Design Requirements**

Design Requirements	**Description**
Hardware Requirements	
Host PC	Any 64-bit Windows Operating System
Software Requirements	
Libero® System-on-Chip (SoC)	v11.6
Modelsim®	v10.3c

Using 9x9 Multiplier Mode

Overview

The 9-bit×9-bit multipliers are extensively used in low precision video processing applications. In video applications, the color conversion formats such as YUV to RGB, RGB to YUV, and RGB to YCbCr, NTSC, PAL, and so on of 9-bit×9-bit multipliers are used. In image processing, the operations involving 8-bit RGB such as 3×3, 5×5, 7×7 matrix multiplications, image enhancement techniques, scaling, resizing, and so on 9-bit×9-bit multipliers are used. The IGLOO2 and SmartFusion2 devices address these applications by using the mathblock in dot product (DOTP) mode.

The following sections explain the DOTP configurations and capabilities, guidelines, different implementation methods with design examples, and their performance and simulation results.

The mathblock when configured in DOTP mode has two independent 9-bit×9-bit multipliers followed by adder. The sum of the dual independent 9×9 multiplier (DOTP) result is stored in upper 35-bit of 44-bit register. In DOTP mode, mathblock implements the following equation:

$$\text{Multiplier result} = (A[8:0] \times B[17:9] + A\ 17:9] \times B[8:0]) \times 2^9 \qquad \textit{EQ 1}$$

Configuration

The IGLOO2 and SmartFusion2 mathblock in DOTP mode can be used in three different configurations. These configurations are available in the Libero software, **Catalog > Arithmetic** as given below:

- Multiplier
- Multiplier accumulator
- Multiplier addsub

Figure 1 shows the dot product multiplier adder with the IGLOO2 and SmartFusion2 mathblock.

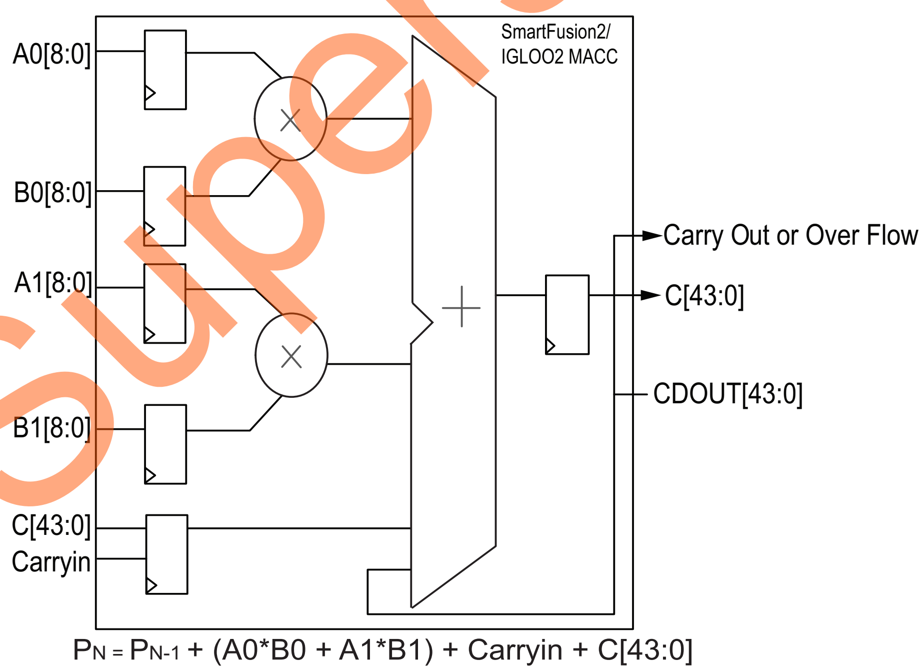

Figure 1 **• Dot Product Multiplier Adder**

Figure 2 shows the dot product multiplier accumulator with mathblock.

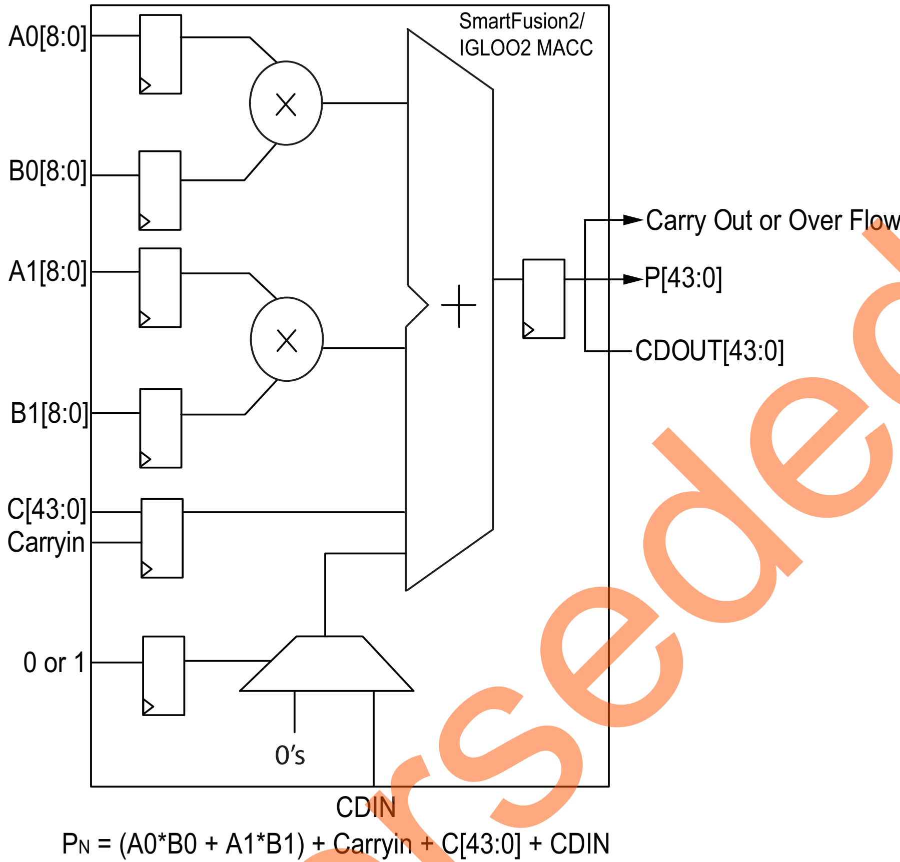

***Figure 2* • Dot Product Multiplier Accumulator**

Figure 3 shows the implemented DOTP multiplier.

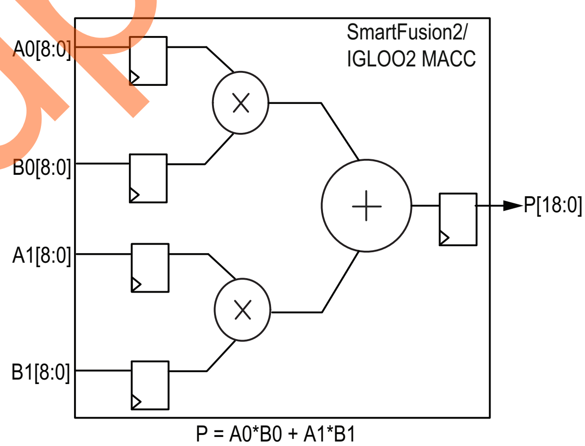

***Figure 3* • Dot Product Multiplier**

Math Functions with DOTP

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

Single Mathblock (DOTP Enabled)

Table 2 • **Math Functions with DOTP**

Conditions	Implemented Equations
P = A[8:0] = B[17:9]; M = A[17:9]; N = B[8:0]	$Y = P^2 + M \times N$
P = A[8:0] = B[17:9]; Q = A[17:9] = B[8:0]	$Y = P^2 + Q^2$
A[8:0] = B[17:9] = 1; B = A[17:9]; Q = B[8:0]	$Y = 1 + Q^2$
A[8:0] = B[17:9] = 1; P = A[17:9]; Q = B[8:0]	$Y = 1 + P \times Q$
P = A[8:0] = A[17:9]; Q = B[17:9] = B[8:0]	$Y = P \times Q + P \times Q = 2 \times P \times Q$

In this method, several 9-bit mathematical functions can be implemented using DOTP mode with a single mathblock.

Guidelines

Microsemi® recommends to use the following when designing with the DOTP multiplier:

- To perform Y = A×B + C×D equation, instantiate Arithmetic IP cores with DOTP enabled for 9×9 multiplications. This avoids inferring two 18×18 multipliers.
- Register the inputs and outputs, when using Arithmetic IP cores (mathblock).
- The registered inputs and outputs must use the same clock.
- Use the cascaded feature to connect the multiple mathblocks. This is achieved by connecting the cascade output (CDOUT) of one MACC block to the cascade input (CDIN) of another mathblock.

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

Design Examples

This section describes the 9×9 Multiplier mode usage with the following design examples:

- Example 1: 6-tap FIR Filter Using Multiple Mathblocks
- Example 2: 6-tap FIR Filter Using Single Mathblock
- Example 3: Alpha Blending

Example 1: 6-tap FIR Filter Using Multiple Mathblocks

This design example (Figure 4 on page 6) shows the 6-tap FIR filter (systolic FIR filter) implementation with multiple mathblocks and also shows the performance results of the implementation.

Design Description

The 6-tap FIR filter design with multiple mathblocks is a systolic architecture implementation, refer Figure 4 on page 6. This architecture utilizes a single IGLOO2 and SmartFusion2 mathblock to perform two independent 9×9 multiplications followed by an addition, instead of using two mathblocks that have a single multiplication unit. With this architecture implementation, only three mathblocks are required to design a 6-tap FIR filter. The 6-tap FIR design uses cascaded chains (CDOUT to CDIN) for propagating the sum to achieve the best performance and reducing fabric resources. In this implementation technique, the mathblock is configured as DOTP multiplier Adder. Eight pipeline registers are added in fabric only at the input.

When designing n-tap systolic FIR filters with IGLOO2 and SmartFusion2 mathblock for 9-bit input data and 9-bit coefficient, only n/2 mathblocks are utilized, saving n/2 mathblock resources.

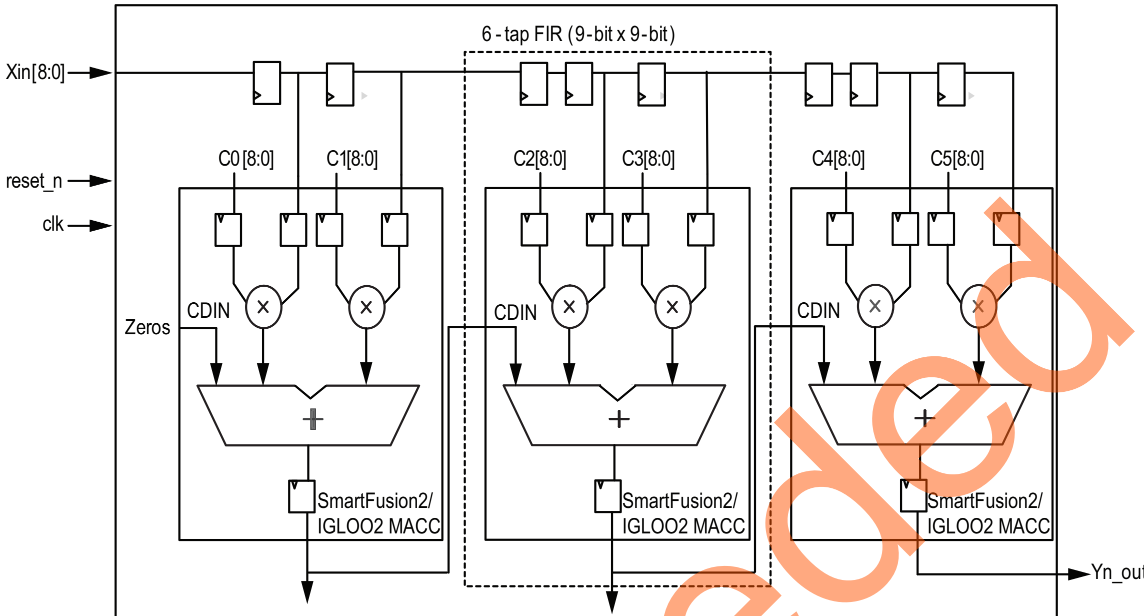

Figure 4 • **6-tap Systolic FIR Filter**

In this design, the FIR filter generates outputs for every clock cycle after an initial latency of 10 clock cycles.

Total initial latency = 8 clock cycles for 6 input samples + 2 clock cycles (MACC block input and output are registered)

= 10 clock cycles

Design Files

For information on the implementation of the 6-tap FIR filter design, refer to the `FIR_6_tap.vhd` design file provided in <*Design files 'FIR_6_TAP*>.

Hardware Configuration

For 6-tap systolic FIR filter, mathblock is configured as DOTP multiplier adder with inputs and outputs registered, refer to Figure 5.

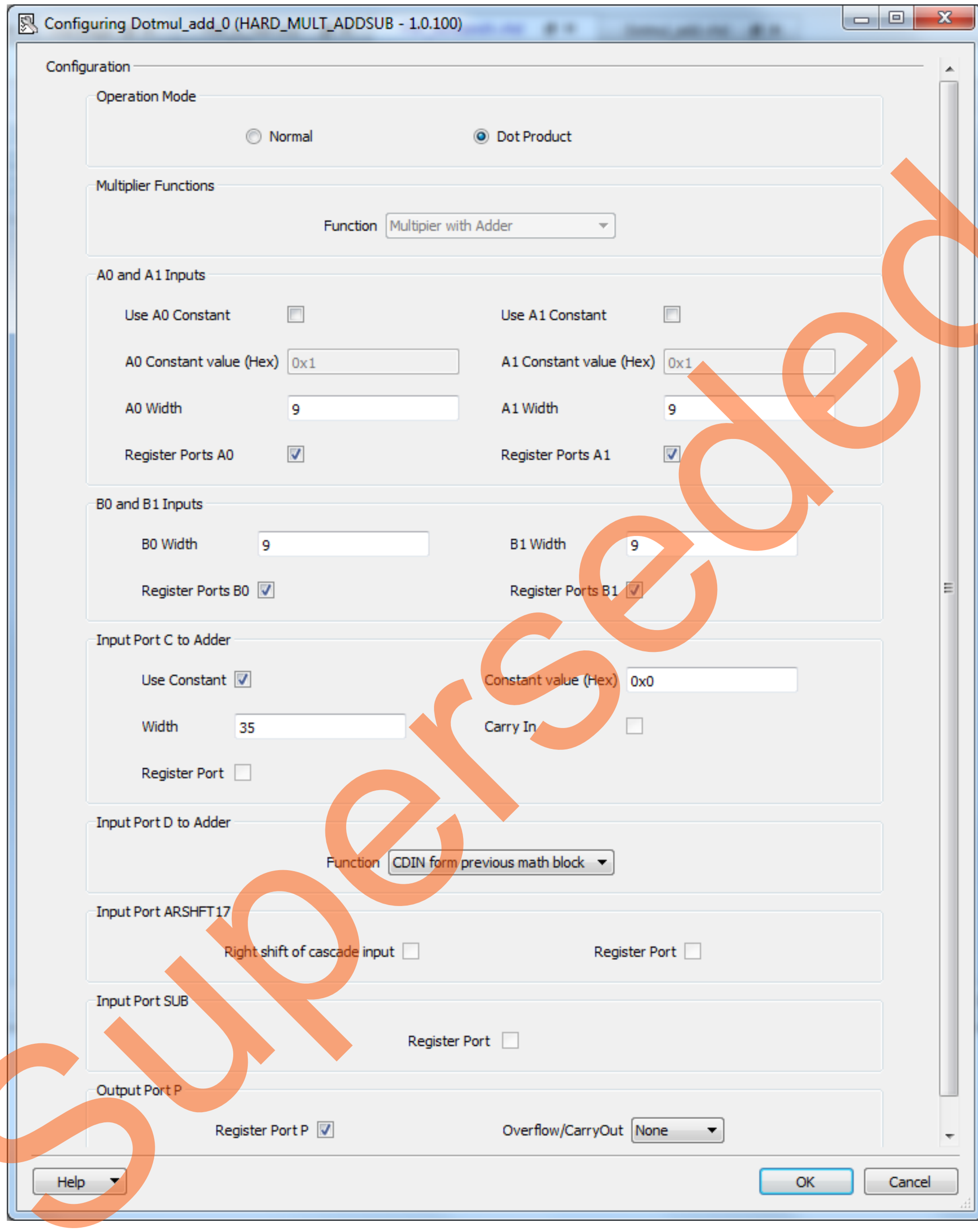

Figure 5 • **DOTP Multiplier Adder for 6-tap Systolic FIR**

Compile and Place-and-Route Results

Figure 6 shows the 6-tap systolic FIR filter resource utilization that uses multiple mathblocks.

Note: The results shown are specific to the IGLOO2 device. Similar results can be achieved using the SmartFusion2 device.

Resource Utilization

Resource Usage

Type	Used	Total	Percentage
4LUT	109	56340	0.19
DFF	180	56340	0.32
I/O Register	0	1125	0.00
User I/O	46	375	12.27
-- Single-ended I/O	46	375	12.27
-- Differential I/O Pairs	0	187	0.00
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	3	72	4.17
Chip Globals	2	16	12.50
CCC	0	6	0.00
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
FDDR	0	1	0.00
MSS	0	1	0.00

Figure 6 **• Resource Utilization for a 6-tap Systolic FIR Filter**

Place-and-Route Results

The frequency of operation is achieved with this implementation after place-and-route, refer to Figure 7.

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	5.000	200.000	1.670	0.200	4.329	7.637

Figure 7 **• Place-and-Route Results for 6-tap Systolic FIR Filter**

Simulation Results

Figure 8 shows the post layout simulation results. The coefficient values (c0-c5) are configured in design as C0 = 5, C1 = 3, C2 = 7, C3 = -4, C4 = 1, C5 = -2. The simulation results show that the 6-tap FIR filter outputs on every clock cycle. It has an initial latency of 10 clock cycles.

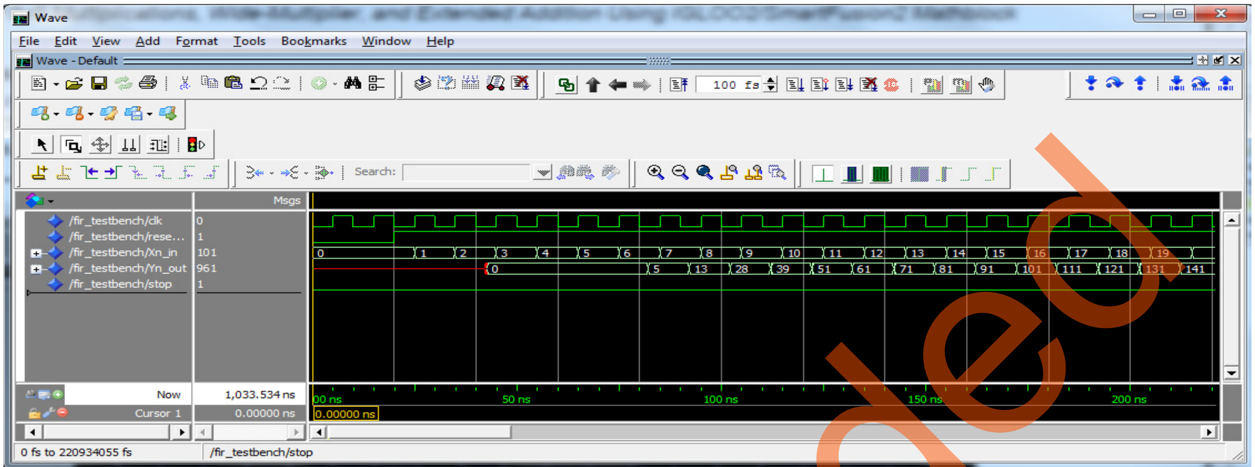

Figure 8 **• 6-tap FIR Filter Post Layout Simulation**

Example 2: 6-tap FIR Filter Using Single Mathblock

This design example shows the 6-tap FIR filter implementation with single-mathblock (MAC FIR filter) and also shows the performance result of the implementations, refer to Figure 9 on page 10.

Design Description

The 6-tap FIR filter can also be implemented with a single mathblock as shown in Figure 9 on page 10. This design uses coefficient memory where coefficients are stored and input memory that stores input samples. The control logic reads two consecutive coefficients from the coefficient memory and two consecutive input samples from the input memory and provides it to mathblock. Due to dual independent 9-bit×9-bit multipliers, the filter result is calculated in four clock cycles instead of six clock cycles that has a single multiplier and accumulator.

If a single multiplier and accumulator is used for sum of the products, the number of cycles taken for result is same as the number of coefficients or number of taps used in filter design. With this relationship, the performance of a single multiplier and accumulator is given as follows:

Maximum input sample rate = System Clock / (Number of taps + 1)

With IGLOO2 and SmartFusion2 mathblock, that is, for two products followed accumulator, the sample rate

= Clock /((1/2 × number of taps)+1)

For 6-tap FIR filter, sample rate = Clock/(6/2 + 1) = Clock/4

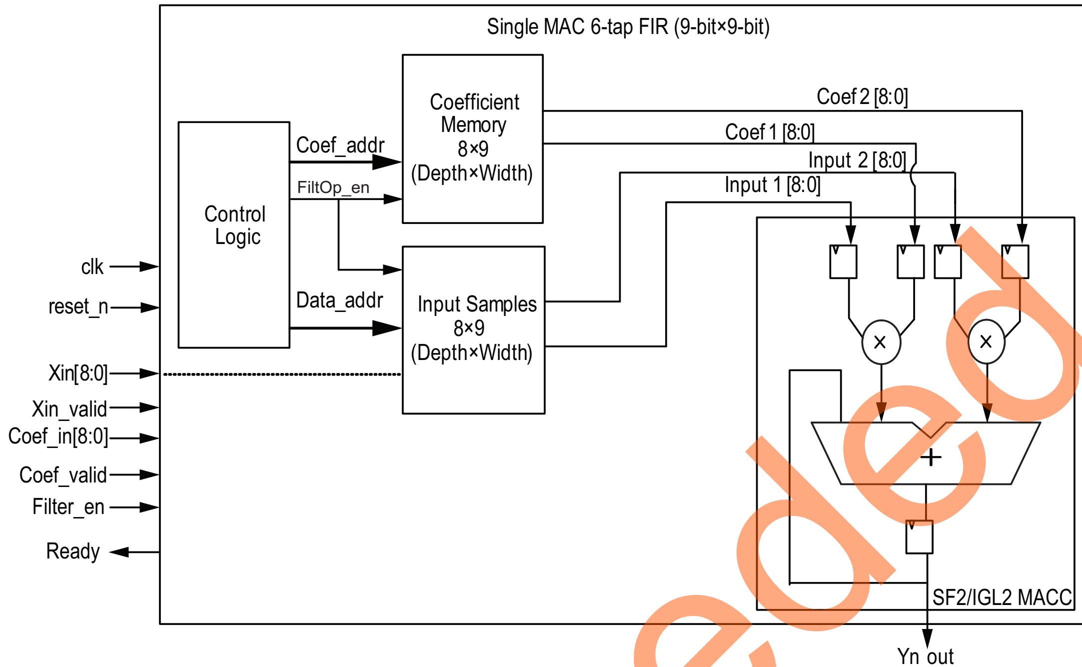

***Figure 9* • 6-tap FIR Filter With Single Mathblock**

Design Files

For information on the implementation of the 6-tap FIR filter design, refer to the `MAC_FIR_6_tap.vhd` design file provided in <*Design files' FIR_6_TAP_singleMACC*>.

Hardware Configuration

In this implementation, the mathblock used is DOTP multiplier accumulator as shown in Figure 10.

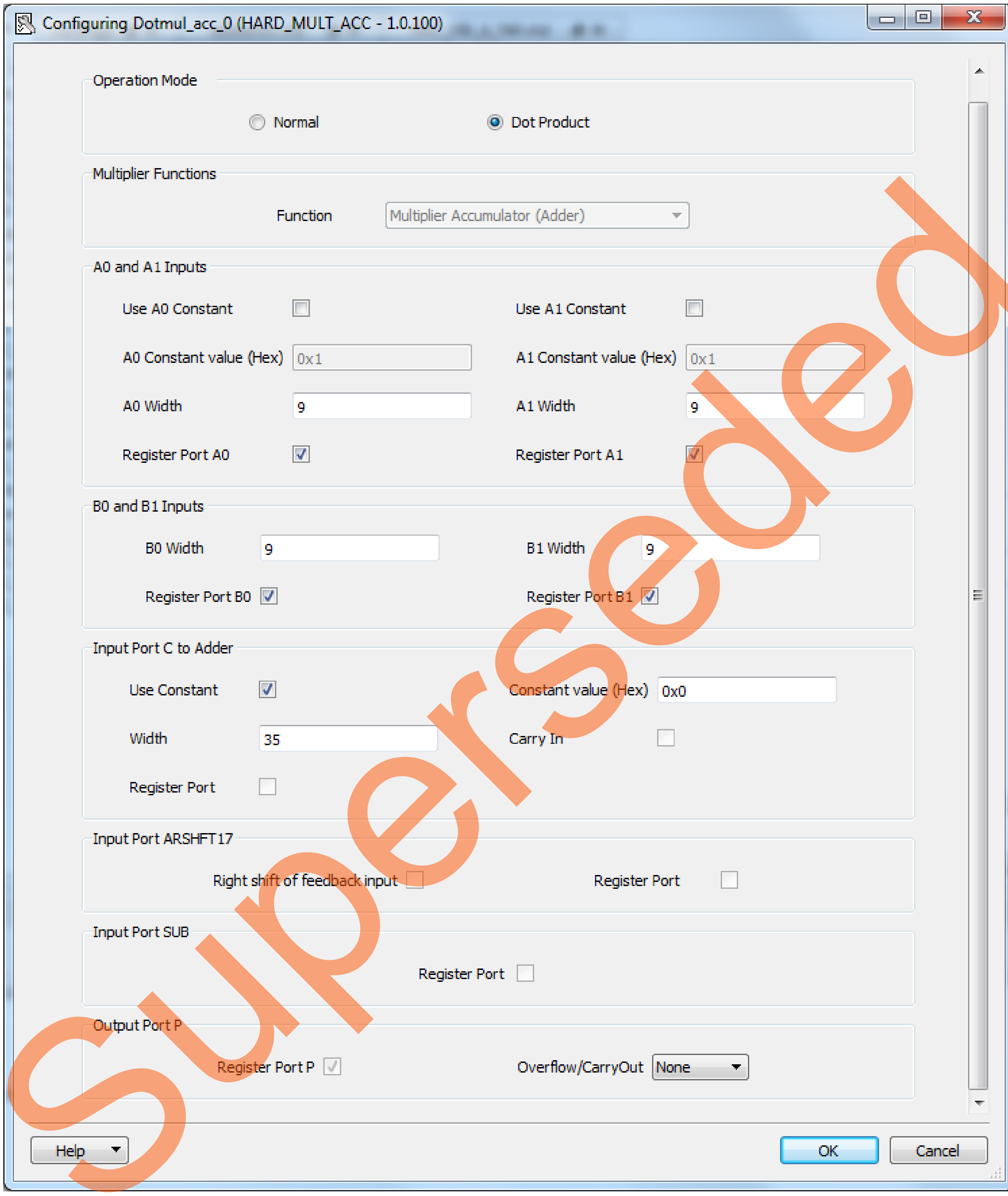

Figure 10 **• Dot Product Multiplier Accumulator**

Compile and Place-and-Route Results

Figure 11 shows the resource utilization results for the 6-tap FIR filter with a single mathblock.

Note: The results shown are specific to the IGLOO2 device. Similar results can be achieved using the SmartFusion2 device.

Resource Utilization

Resource Usage

Type	Used	Total	Percentage
4LUT	151	56340	0.27
DFF	197	56340	0.35
I/O Register	0	1125	0.00
User I/O	49	375	13.07
-- Single-ended I/O	49	375	13.07
-- Differential I/O Pairs	0	187	0.00
RAM64x18	2	72	2.78
RAM1K18	0	69	0.00
MACC	1	72	1.39
Chip Globals	2	16	12.50
CCC	0	6	0.00
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
FDDR	0	1	0.00
MSS	0	1	0.00

Figure 11 **• Resource Utilization Results for a Single MAC FIR**

Place-and-Route Results

The frequency of operation achieved with this implementation after place-and-route is shown in Figure 12.

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	4.273	-0.182	3.813	7.898

Figure 12 **• Place-and-Route Results for Single MAC FIR**

Example 3: Alpha Blending

The following example shows the implementation of Alpha blending used in image processing as shown in Figure 13. Alpha blending is the process of combining a translucent foreground color with a background color, thereby producing a new blended color.

Design Description

The Alpha blending for each R_{new}, G_{new}, B_{new} as shown in Figure 13 is implemented using the following equations:

$$R_{new} = (1\text{-alpha}) \times R0\,[7:0] + \text{alpha} \times R1[7:0] \quad \textit{EQ 2}$$

$$G_{new} = (1\text{-alpha}) \times G0\,[7:0] + \text{alpha} \times G1[7:0] \quad \textit{EQ 3}$$

$$B_{new} = (1\text{-alpha}) \times B0\,[7:0] + \text{alpha} \times B1[7:0] \quad \textit{EQ 4}$$

This implementation uses three mathblocks to output R', G', B' values simultaneously for blended image. Each mathblock is configured as dot product multiplier for performing 9-bit×9-bit multiplications.

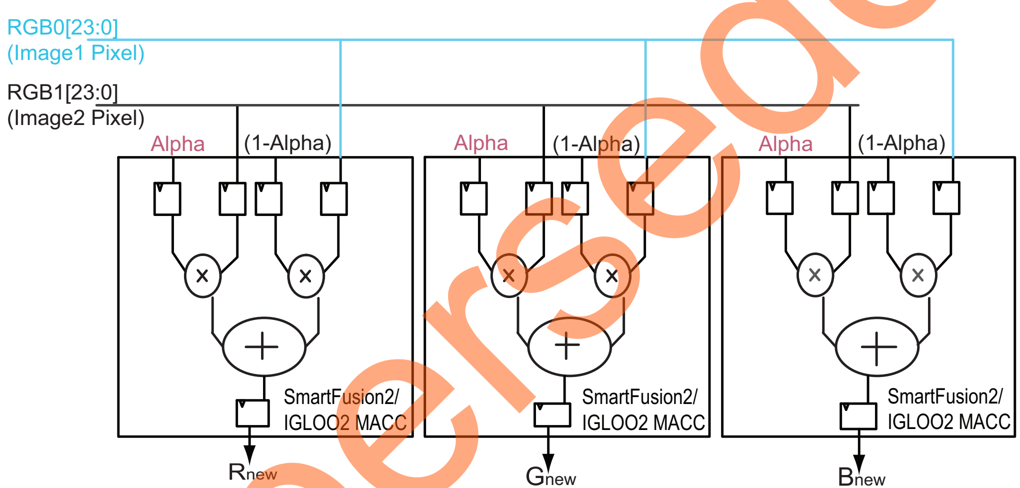

Figure 13 • **Alpha Blending Implementation Using IGLOO2 and SmartFusion2 Mathblocks**

Hardware Configuration

For Alpha blending, mathblock is configured as DOTP multiplier with inputs and outputs registered.

Compile and Place-and-Route Results

Figure 14 shows the Alpha blending resource utilization using three mathblocks.

Note: The results shown are specific to the IGLOO2 device. Similar results can be achieved using the SmartFusion2 device.

Resource Utilization

Resource Usage

Type	Used	Total	Percentage
4LUT	139	56340	0.25
DFF	135	56340	0.24
I/O Register	0	1125	0.00
User I/O	69	375	18.40
-- Single-ended I/O	69	375	18.40
-- Differential I/O Pairs	0	187	0.00
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	3	72	4.17
Chip Globals	2	16	12.50
CCC	0	6	0.00
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
FDDR	0	1	0.00
MSS	0	1	0.00

***Figure 14* • Resource Utilization Results for Alpha Blending**

Place-and-Route Results

The frequency of operation achieved with this implementation after place-and-route is shown in Figure 15.

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	2.376	420.875	5.000	200.000	1.493	0.441	4.684	9.475

***Figure 15* • Place-and-Route Results for Alpha Blending**

Wide-Multiplier

Overview

The wide-multipliers are extensively used in high precision (more than 18×18 multiplication) wireless and medical applications. These applications require high precision at every stage when implementing complex arithmetic functions used in FFT, filters and so on. Military, test, 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 DSP functions that require high precision, the IGLOO2 and SmartFusion2 devices offer implementing wide-multipliers (that is, operands width more than 18×18) with the IGLOO2 and SmartFusion2 mathblock. The wide-multipliers are implemented by cascading multiple IGLOO2 and SmartFusion2 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 example to achieve the best performance results.

Configuration

When implementing the wide-multipliers, the IGLOO2 and SmartFusion2 mathblocks are configured in Normal mode to function as normal multiplier (18×18), normal multiplier accumulator, and normal multiplier addsub.

Guidelines

Microsemi recommends to use the following for implementing wide-multiplier to achieve the best results.

- The inputs and output 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 inputs and outputs of mathblock, if arithmetic cores (mathblock) are used.
- CDOUT of one mathblock is connected to the CDIN of another mathblock.

Design Examples

This section shows the wide-multiplier with the following design examples:

- Multiplier 32×32 implementation using multiple mathblock
- Multiplier 32×32 implementation using single mathblock

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

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:

Design Description

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

$$A = (AH \times 2^{17}) + AL;$$

$$B = (BH \times 2^{17}) + BL;$$

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

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

The 32×32 multiplier is implemented efficiently using four mathblocks without using fabric resources to produce 64-bit result as shown in Figure 16 and Figure 17 on page 17. To achieve the best performance results, mathblock input and output registers are used.

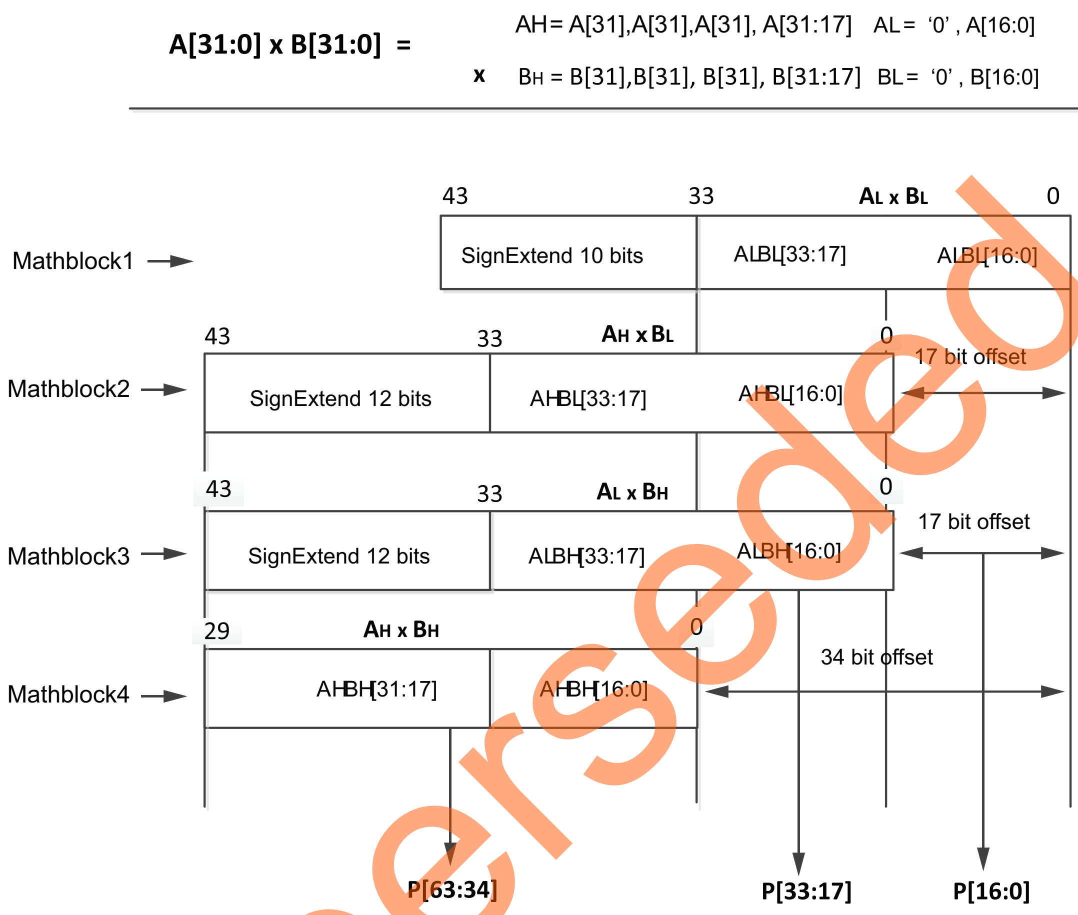

Figure 16 **• 32x32 Multiplication**

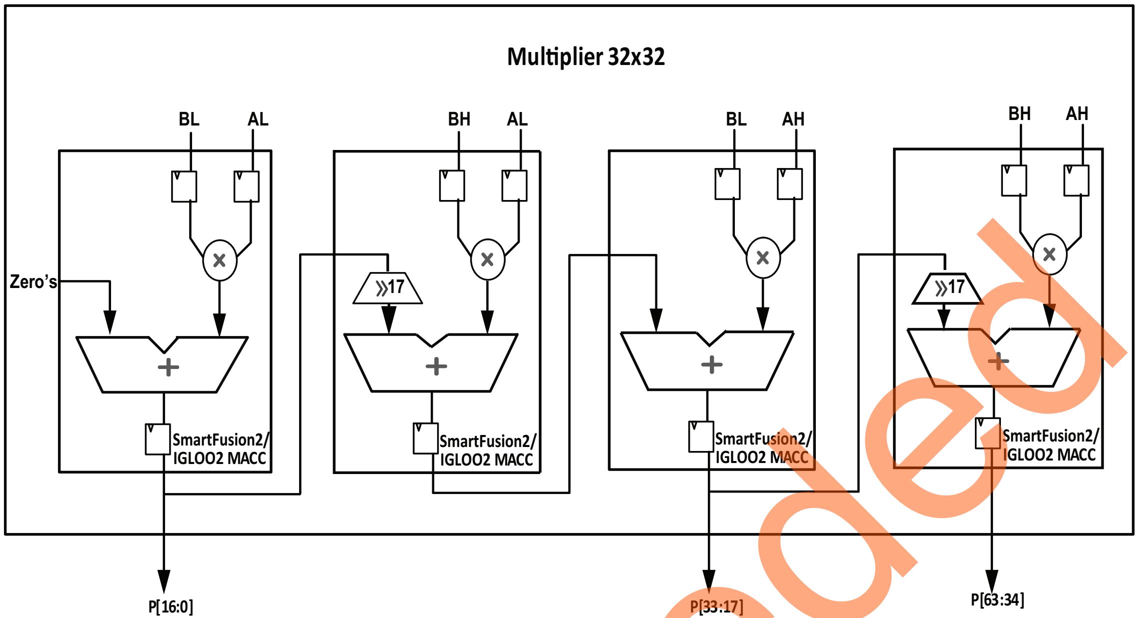

***Figure 17* • Implementation of 32x32 Multiplier**

When implementing using HDL, to infer mathblock input and output registers by synthesis tool, pipeline stages are added at output and input to achieve the 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 information on the implementation of the multiplier 32×32 design, refer to the `Mult32×32_multipleMACC.vhd` design file provided in <*Design files -> Mult32×32_multipleMACC*>.

Hardware Configuration

For 32×32 multiplier using single mathblock, mathblock is configured to function as normal multiplier, normal multiplier addsub with ARSHFT enabled, inputs and outputs registered.

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

Compile and Place-and-Route Results

Figure 18 shows the 32×32 multiplier resource utilization when using multiple mathblocks.

Note: The results shown are specific to the IGLOO2 device. Similar results can be achieved using the SmartFusion2 device.

Resource Utilization

Resource Usage

Type	Used	Total	Percentage
4LUT	145	56340	0.26
DFF	289	56340	0.51
I/O Register	0	1125	0.00
User I/O	130	375	34.67
-- Single-ended I/O	130	375	34.67
-- Differential I/O Pairs	0	187	0.00
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	4	72	5.56
Chip Globals	2	16	12.50
CCC	0	6	0.00
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
FDDR	0	1	0.00
MSS	0	1	0.00

***Figure 18* • Resource Utilization for Multiple Mathblocks**

Place-and-Route Results

The frequency of operation achieved with this implementation after place-and-route is shown in Figure 19.

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	5.000	200.000	3.910	0.537	4.772	8.724

***Figure 19* • Place-and-Route Results for 32×32 With Multiple Mathblock**

Example 2: 32×32 Multiplier Implementation Using Single Mathblock

The following section explains the 32×32 multiplier implementation with a single mathblock and also shows the performance results.

Design Description

The 32×32 multiplier is implemented using the same algorithm as shown in "Example 1: 6-tap FIR Filter Using Multiple Mathblocks" section on page 5.

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

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

In this implementation, the four multiplications are computed using a single mathblock in sequential manner. The control finite-state machine (FSM) in the design provides the inputs to the mathblock sequentially in four successive states as shown in Figure 20 and appropriately enables the shift operation in the corresponding state. The mathblock used in this design is configured as normal multiplier accumulator 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

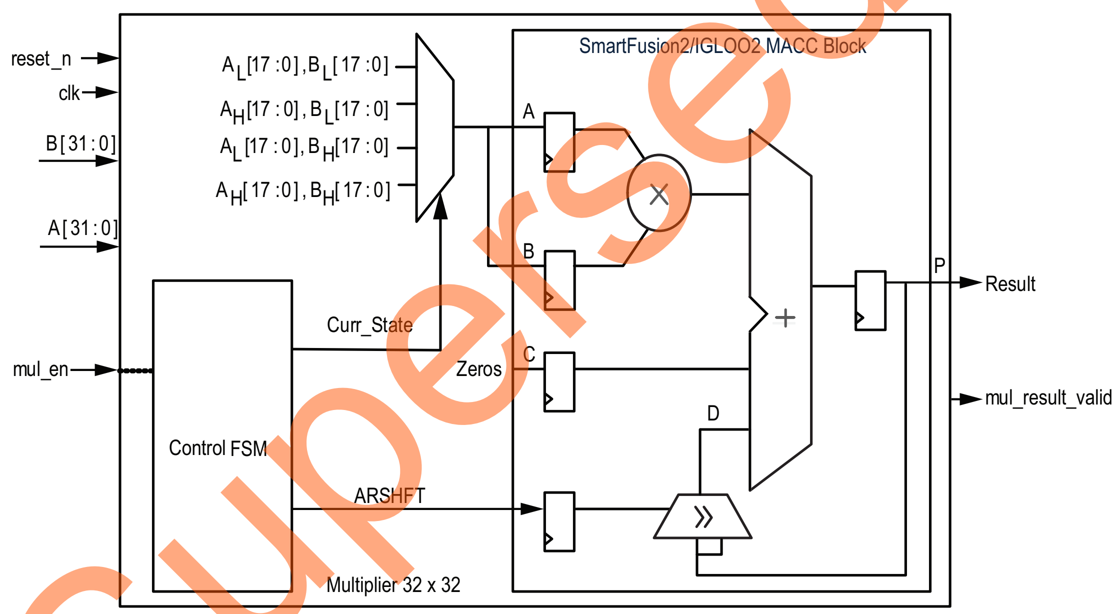

Figure 20 **• Multiplier 32×32 with One MACC Block**

Design Files

For more information on the implementation of the multiplier 32×32 design, refer to the `Mult32×32.vhd` design file provided in <*Design files'Mult32×32*>.

Hardware Configuration

For 32×32 multiplier using single mathblock, it is configured to function as normal multiplier accumulator with inputs and outputs registered.

Compile and Place-and-Route results

Figure 21 shows the 32×32 multiplier resource utilization when using a single mathblock.

Note: The results shown are specific to the IGLOO2 device. Similar results can be achieved using the SmartFusion2 device.

Resource Utilization

Resource Usage

Type	Used	Total	Percentage
4LUT	84	56340	0.15
DFF	141	56340	0.25
I/O Register	0	1125	0.00
User I/O	132	375	35.20
-- Single-ended I/O	132	375	35.20
-- Differential I/O Pairs	0	187	0.00
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	1	72	1.39
Chip Globals	2	16	12.50
CCC	0	6	0.00
RCOSC 25 50MHZ	0	1	0.00
RCOSC 1MHZ	0	1	0.00
XTLOSC	0	1	0.00
FDDR	0	1	0.00
MSS	0	1	0.00

***Figure 21* • Resource Utilization for a Single Mathblock**

Place-and-Route Results

The frequency of operation is achieved with this implementation after place-and-route is shown in Figure 22.

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	2.711	368.868	5.000	200.000	2.965	-0.048	4.101	8.493

***Figure 22* • Place-and-Route Results for 32×32 Multiplier with Single Mathblock**

Simulation Results

Figure 23 shows the post layout simulation results. The simulation result shows the multiplier outputs on 8 clock cycles after input is provided.

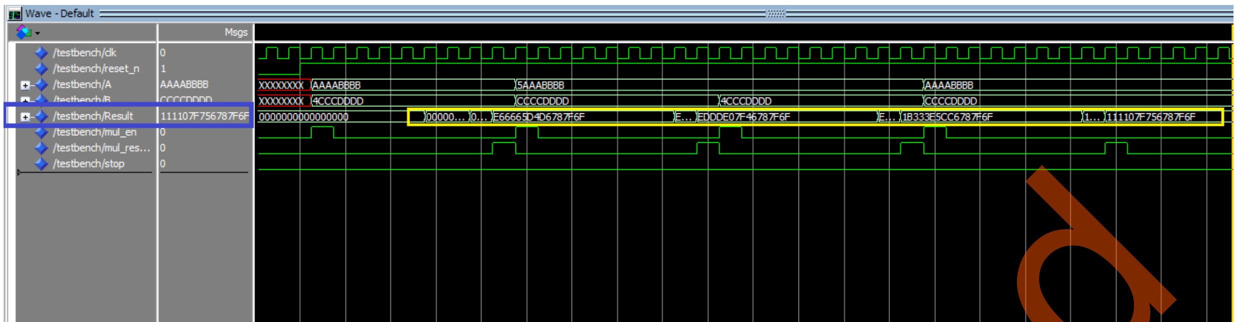

Figure 23 **• Multiplier 32×32 Post Layout Simulation Results**

Extended Addition

Overview

Mathblock has a 3-input adder and supports accumulation 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 IGLOO2 and SmartFusion2 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 IGLOO2 and SmartFusion2 mathblock and upper part (MSB) of addition is implemented with minimal fabric adder logic.

For a 2-input addition, the inputs can be from any one of the following:

1. CDIN and C input
2. Multiplier output and CDIN
3. Multiplier output and C input

For a 3-input addition, the inputs are from multiplier output, CDIN, and C-input. To perform arithmetic additions, the IGLOO2 and SmartFusion2 mathblock provides Carryin input and Carryout signal for propagating the carry from one mathblock to another mathblock or from mathblock to fabric logic.

Configuration

When implementing the extended addition, the IGLOO2 and SmartFusion2 mathblock is configured in Normal mode to function as normal multiplier addsub.

Guidelines

- Mathblock must be configured to function as multiplier adder or subtracter to perform 2-input extended signed addition.
- Add pipeline stages in RTL, so that the synthesis tool can automatically infer registers of mathblock or register the inputs and outputs of mathblock, if arithmetic cores (mathblock) are used.
- Ensure that the CDOUT of one mathblock is connected to the CDIN of another mathblock.

Design Examples

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

- 2-input extended signed addition
- 3-input extended signed addition

Example 1: 2-input Signed Extended Addition

The following section shows a 2-input extended signed addition—if one operand is more than 44-bit wide. In this section, it is also shown that the 2-input extended signed addition implementation logic with fabric resources are implemented with the multiplier adder.

Design Description

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 following logic must be implemented in fabric as shown in Figure 24.

$$
\begin{array}{cccccccccc}
 & U_{m-1} & U_{m-2} \ldots & U_{n+2} & U_{n+1} & U_n & U_{n-1} & U_{n-2} & \ldots & U_0 \\
+ & V_{n-1} & V_{n-1} \ldots & V_{n-1} & V_{n-1} & V_{n-1} & V_{n-1} & V_{n-2} & \ldots & V_0 \\
\hline
 & Z_{m-1} & Z_{m-2} \ldots & Z_{n+2} & Z_{n+1} & Z_n & Z_{n-1} & Z_{n-2} & \ldots & Z_0
\end{array}
$$

***Figure 24* • 2-input Extended Signed Addition**

Where U is an m-bit value (where m > 44), 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 is computed in the mathblock and the upper part is computed in the fabric.

Z = (Sumupper, Sumlower)

EQ 5

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

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

EQ 6

The Upper part of sum Z = U + V is calculated as shown below:

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

EQ 7

V [m: n] = {S, S….S, X},
S = P[n-1] AND X

Where,
P [n-1] is MSB of Sumlower
X is the overflow of the Sumlower (from the mathblock)
(m-n-1) number of S's must be appended in MSB bits of the V[m: n].

Hardware Implementation

Figure 25 shows the operand width of C as 52-bit wide and explains the implementation for 2-input extended signed addition. For 3-input addition, mathblock is configured as multiplier addsub in Normal mode. The upper part and lower part of the sum are shown as follows:

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}}

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

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

Where,

S = P[43] AND CARRYOUT

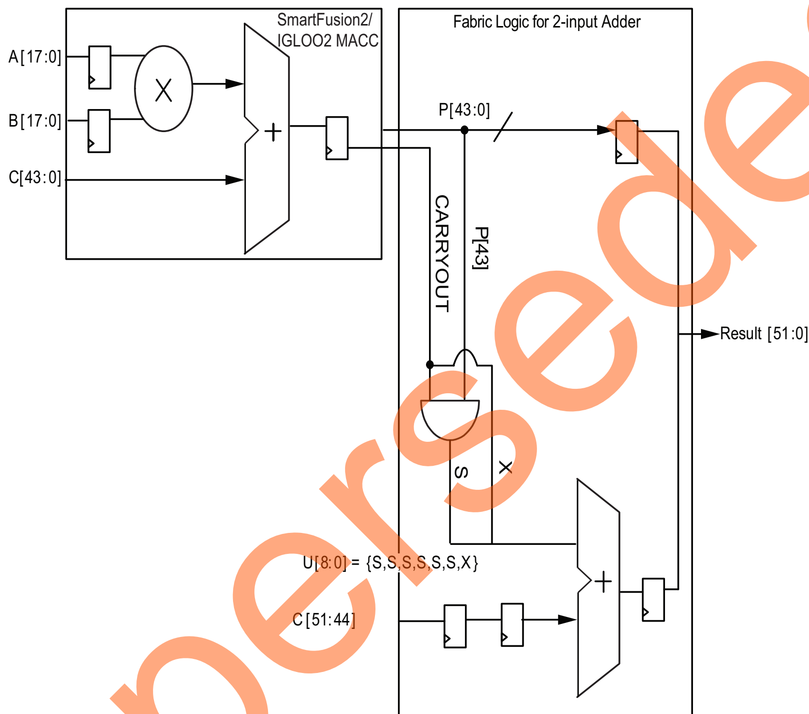

***Figure 25* • Fabric Logic for 2-input Extended Addition**

Design Files

For information on the implementation of the 2-input extended addition, refer to the `Extended_adder_2_input.vhd` design file provided in <*Design files'Extended_adder_2_input*>.

Compile and Place-and-Route Results

Figure 26 shows the 2-input extended addition resource utilization when using the mathblock and fabric logic.

Note: The results shown are specific to the IGLOO2 device. Similar results can be achieved using the SmartFusion2 device.

Resource Utilization with Fabric Adder Logic

Resource Usage

Type	Used	Total	Percentage
4LUT	62	56340	0.11
DFF	88	56340	0.16
I/O Register	0	1125	0.00
User I/O	142	375	37.87
-- Single-ended I/O	142	375	37.87
-- Differential I/O Pairs	0	187	0.00
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	1	72	1.39
Chip Globals	2	16	12.50
CCC	0	6	0.00
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
FDDR	0	1	0.00
MSS	0	1	0.00

***Figure 26* • Resource Utilization for 2-input Extended Addition with Fabric Resources**

Place-and-Route Results with Fabric Adder Logic

The frequency of operation achieved with this implementation after place-and-route is shown in Figure 27.

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	3.217	310.849	5.000	200.000	3.476	0.461	4.329	8.162

***Figure 27* • Place-and-Route Results for 2-input Extended Addition with Fabric Resources**

Simulation Results

Figure 28 show the post layout simulation results. The simulation result shows that the 2-input addition outputs on the next clock cycle after the input is provided.

Figure 28 • Post Layout Simulation Results for 2-Input Extended Addition with Fabric Adder

Example 1: 3-input Signed Extended Addition

The following section explains the 3-input extended signed addition, if one or more operands are more than 44-bit wide. In this section, it shows the 3-input extended signed addition implementation logic with fabric resources.

Design Description

3-input Extended Addition

For performing 3-input extended addition, Z = T + U + V, with two operands width more than the mathblock input width 44, the following logic must be implemented in fabric as shown in Figure 29.

	T_{m-1}	T_{m-2}	...	T_{n+2}	T_{n+1}	T_n	T_{n-1}	T_{n-2}	...	T_0
	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 29 • 3-input Extended Signed Addition

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

Z = {Sumupper, Sumlower}

EQ 8

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 Mathblock, where n = 44 is mathblock output width.

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

EQ 9

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 10

(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 S's should be appended in MSB bits of the V[m: n].

Hardware Implementation

Figure 30 shows the operand widths of C, D are 52-bit wide and explains implementation for 3-input extended signed addition. For 3-input addition, mathblock is configured as multiplier addsub in Normal mode. The lower part of the sum and upper part of the sum are shown as follows:

For 52-bit, 3-input extended signed addition,

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}}

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

Result [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

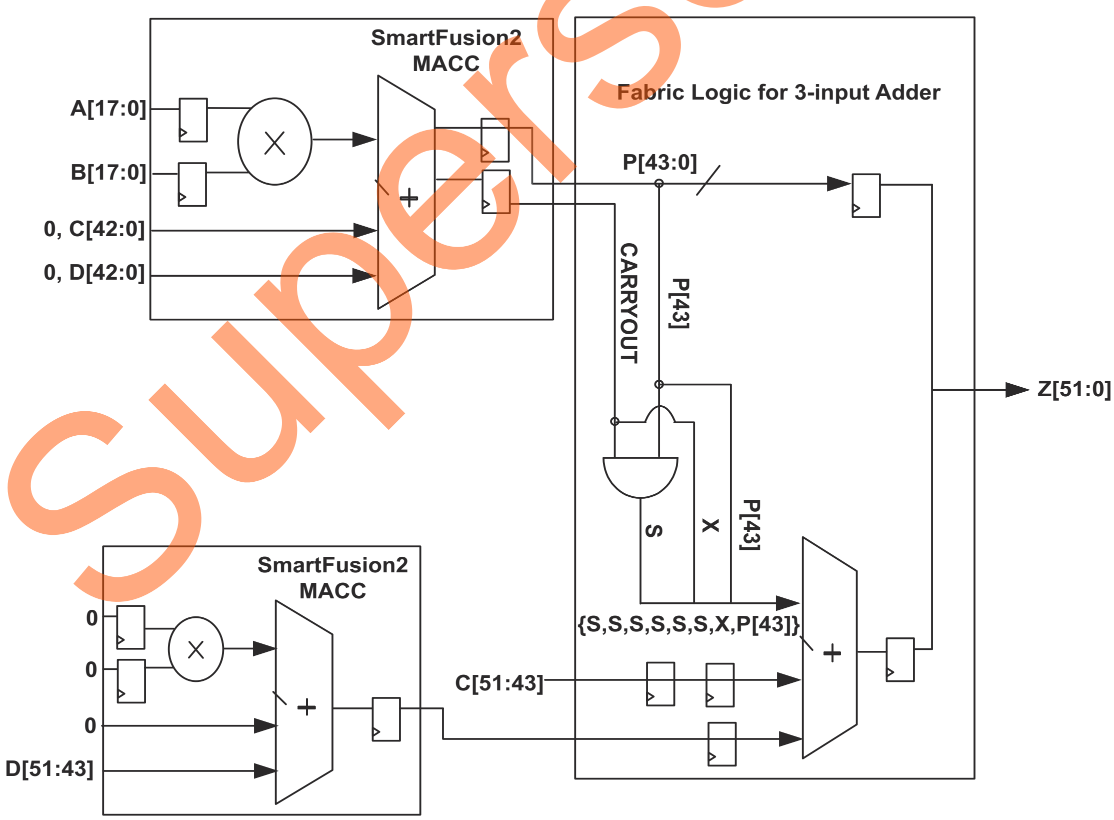

Figure 30 **• Fabric Logic for 3-input Extended Addition**

Design Files

For more information on how to implement the 3-input extended addition, refer to the `Extended_adder_3_input.vhd` design file provided in <*Design files'Extended_adder_3_input*>.

Compile and Place-and-Route Results

Figure 31 shows the 3-input extended addition resource utilization when using the fabric logic.

Note: The results shown are specific to the IGLOO2 device. Similar results can be achieved using the SmartFusion2 device.

Resource Utilization with Fabric Adder Logic Implemented with MACC Block

Resource Usage

Type	Used	Total	Percentage
4LUT	92	56340	0.16
DFF	120	56340	0.21
I/O Register	0	1125	0.00
User I/O	194	375	51.73
-- Single-ended I/O	194	375	51.73
-- Differential I/O Pairs	0	187	0.00
RAM64x18	0	72	0.00
RAM1K18	0	69	0.00
MACC	2	72	2.78
Chip Globals	2	16	12.50
CCC	0	6	0.00
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
FDDR	0	1	0.00
MSS	0	1	0.00

Figure 31 • **Resource Utilization for 3-input Extended Addition with Fabric Resources**

Place-and-Route Results with Fabric Adder Logic Implemented with MACC Block

The frequency of operation achieved with this implementation after place-and-route is shown in Figure 32.

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	2.796	357.654	5.000	200.000	2.826	0.635	4.778	9.209

Figure 32 • **Place-and-Route Results for 3-input Extended Addition with Fabric Resources**

Simulation Results

Figure 33 shows the post synthesis simulation results. The simulation result shows that the 3-input addition outputs on the three clock cycles after the input is provided.

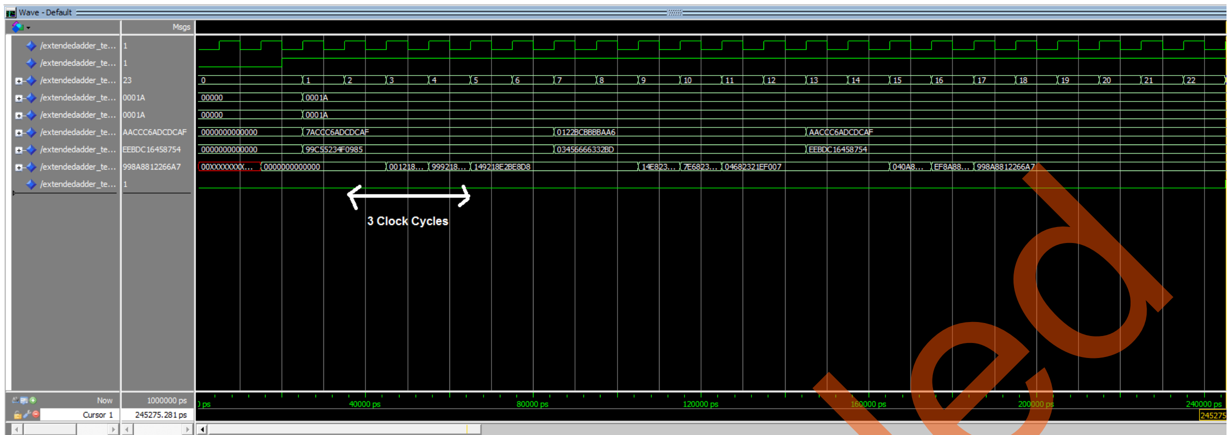

***Figure 33* • Post Synthesis Simulation Results for 3-input Extended Addition with Fabric Adder**

Conclusion

This application notes explains IGLOO2 and SmartFusion2 mathblock features such as 9x9 Multiplier mode, wide-multiplier, and extended addition. This document also provides implementation techniques and guidelines along with the design examples for the 9x9 multiplication, wide-multiplier, and extended addition for optimum performance.

Appendix: Design Files

Download the design files (VHDL) from the Microsemi website:
http://soc.microsemi.com/download/rsc/?f=m2s_m2gl_ac398_liberov11p6_df

Refer to the `Readme.txt` file included in the design file for the directory structure and description.

List of Changes

The following table shows important changes made in this document for each revision.

Date	Changes	Page
Revision 3 (October 2015)	Updated the document for Libero v11.6 software release (SAR 72381)	NA
Revision 2 (March 2015)	Updated the document for Libero v11.5 software release (SAR 64344).	NA
Revision 1 (September 2014)	Updated the document for Libero v11.4 software release (SAR 59686).	NA
Revision 0 (June 2013)	Initial release.	NA

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & 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; Ethernet Solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 3,600 employees globally. Learn more at **www.microsemi.com**.

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

51900274-3/10.15