Arithmetic Modules

User Guide

FPGA-IPUG-02032-1.0

February 2018

Contents

Figures

Tables

1. Introduction

The IP Catalog provides a variety of modules to assist your design work. These modules cover a variety of common functions and can be customized. They are optimized for Lattice Semiconductor device architectures. Use these modules to speed up your design work and to produce the most effective results.

This guide describes the Arithmetic modules that come with the IP Catalog. These modules can be integrated across Lattice device architectures and serve as the building block to realize a more complex or user-defined function. The descriptions mainly cover the ports, attributes and sample functional timing diagrams of the modules.

2. Arithmetic Modules

2.1. Adder

A two-input adder performs signed/unsigned addition of the data from inputs data_a and data_b with an optional cin carry input. The output result carries the Sum of the addition operation with an optional cout carry output.

Shaded portions of the Adder Schematic Diagram (shown in Figure 2.1) are optional and are removed/ignored in certain configurations.

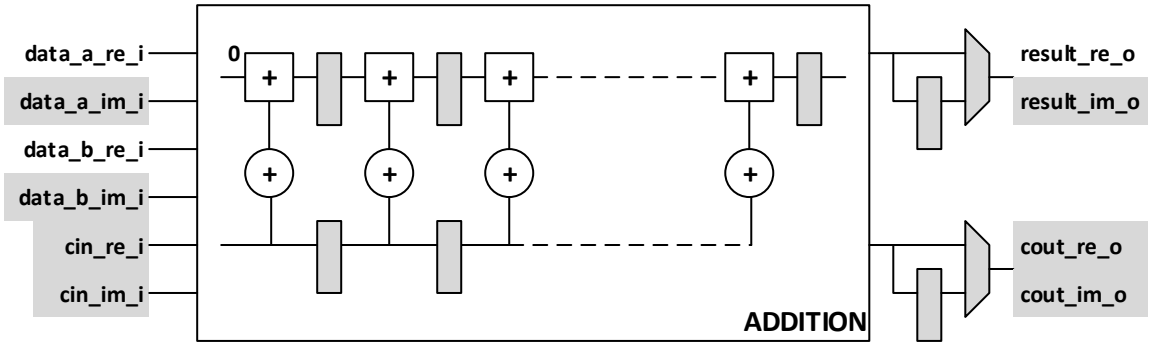

Figure 2.1. Adder Schematic Diagram

2.1.1. Ports

Table 2.1. Adder Ports

Signal Name	Direction	Width (bits)	Description
clk_i	IN	1	This signal connects to the clock pin of the input, output and pipeline registers. All flops toggle on the rising edge of this clock.
clk_en_i	IN	1	This signal is an active HIGH synchronous clock enable to the output flops in the design. 1'b0 – retain the previous state 1'b1 – toggle on the rising edge of the clock as per the logic
rst_i	IN	1	This signal initializes the flops in the design to a known state. It is an active HIGH asynchronous reset whose deassertion should be synchronized to the clock during integration.
data_a_re_i	IN	D_WIDTH	This signal contains a simple number or the Real part of a complex number which is the augend in the addition. USE_CNUM == 1'b0: Simple Number USE_CNUM == 1'b1: Real part of a complex number

Signal Name	Direction	Width (bits)	Description
data_a_im_i	IN	D_WIDTH	This signal contains the Imaginary part of a complex number which is the augend in the addition. USE_CNUM == 1'b0: tied to a constant in the integration USE_CNUM == 1'b1: used in the addition
data_b_re_i	IN	D_WIDTH	This signal contains a simple number or the Real part of a complex number which is the addend in the addition. USE_CNUM == 1'b0: Simple Number USE_CNUM == 1'b1: Real part of a complex number
data_b_im_i	IN	D_WIDTH	This signal contains the Imaginary part of a complex number which is the addend in the addition. USE_CNUM == 1'b0: tied to a constant in the integration USE_CNUM == 1'b1: used in the addition
cin_re_i	IN	D_WIDTH	This signal contains a simple number or the Real part of a complex number which is the Carry-In in the addition. USE_CIN == 1'b0: Carry-In is unused in the addition. Tied to a constant in the integration. USE_CIN == 1'b1: Carry-In is used in the addition
cin_im_i	IN	D_WIDTH	This signal contains the Imaginary part of a complex number which is the Carry-In in the addition. USE_CIN or USE_CNUM == 1'b0: Carry-In is unused in the addition. Tied to a constant in the integration. USE_CIN and USE_CNUM == 1'b1: Carry-In is used in the addition
result_re_o	OUT	D_WIDTH	This signal carries a simple number or the Real part of a complex number from the sum of the addition. Core functionality is represented below. {data_a_re_i + data_b_re_i + cin_re_i}
result_im_o	OUT	D_WIDTH	This signal carries the Imaginary part of a complex number from the sum of the addition. USE_CNUM == 1'b0: unconnected in the integration USE_CNUM == 1'b1: Core functionality is represented below {data_a_im_i + data_b_im_i + cin_im_i}
cout_re_o	OUT	D_WIDTH	This signal carries a simple number or the Real part of a complex number from the Carry-Out of the addition. USE_COUT == 1'b0: unconnected in the integration USE_COUT == 1'b1: Carry-Out is implemented SIGNED == Unsigned: treated as cout SIGNED == Signed: treated as Overflow This signal when HIGH indicates an overflow in the sum i.e. a value outside the range of the configured number system.

Signal Name	Direction	Width (bits)	Description
cout_im_o	OUT	D_WIDTH	This signal carries the Imaginary part of a complex number from the Carry-Out of the addition. USE_COUT or USE_CNUM == 1'b0: unconnected in the integration USE_COUT and USE_CNUM == 1'b1: Carry-Out is implemented SIGNED == Unsigned: treated as cout SIGNED == Signed: treated as Overflow This signal when HIGH indicates an overflow in the sum i.e. a value outside the range of the configured number system.

2.1.2. Attributes

Table 2.2. Adder Attributes

Configuration	Range	Default Value	Description
D_WIDTH	1–64	16	This configuration is the data width of the input/output ports.
SIGNED	Signed or Unsigned	Unsigned	This configuration controls the interpretation of the inputs whether a signed or an unsigned number.
USE_CNUM	0–1	0	This configuration controls the interpretation of the numbers being involved in the addition. 0 – Simple Numbers. data*_re_i, cin_re_i, result_re_o and cout_re_o are treated as simple numbers only while data*_im_i, cin_im_i, result_im_o and cout_im_o IOs are ignored. 1 – Complex Numbers. The logic is implemented on the basis of the IOs as complex numbers.
USE_CIN	0–1	0	This configuration controls the usage of Carry-In in the addition. 0 – Ignore Carry-In. cin_re_i and cin_im_i inputs are unused. 1 – Consider Carry-In. cin_re_i and cin_im _i inputs are used.
USE_COUT	0–1	0	This configuration controls the implementation of Carry-Out in the addition. 0 – Ignore Carry-Out. cout_re_o and cout_im_o outputs are always driven LOW. 1 – Consider Carry-Out. cout_re_o and cout_im_o outputs are driven with carry.
USE_IREG	“on”, “off”	“off”	This configuration controls the registering of the adder inputs. “off” – Unregistered. Input is directly routed to internal logic. “on” – Registered. Input is registered before being used by internal logic. This configuration improves static timing of the design when combinational delay from the adder is high.

Configuration	Range	Default Value	Description
USE_OREG	“on”, “off”	“off”	This configuration controls the registering of the adder result onto the outputs. “off” – Unregistered. Result is directly routed to the output. “on” – Registered. Result is registered at the output. This configuration improves static timing of the design when combinational delay from the adder is high.
PIPELINES	0-(D_WIDTH/8)–1	1	This configuration controls the insertion of pipeline stages into the addition operation. This configuration accepts integers only and any value less than 1 is treated as no pipelining by the design. Maximum pipelines permitted is dependent on the number of 8-bit chunks in the design computed IO Width configuration. For example: 1–8 bits: 0 9–16 bits: 1 17–24 bits: 2 25–32 bits: 3 and so on

2.1.3. Timing Diagram

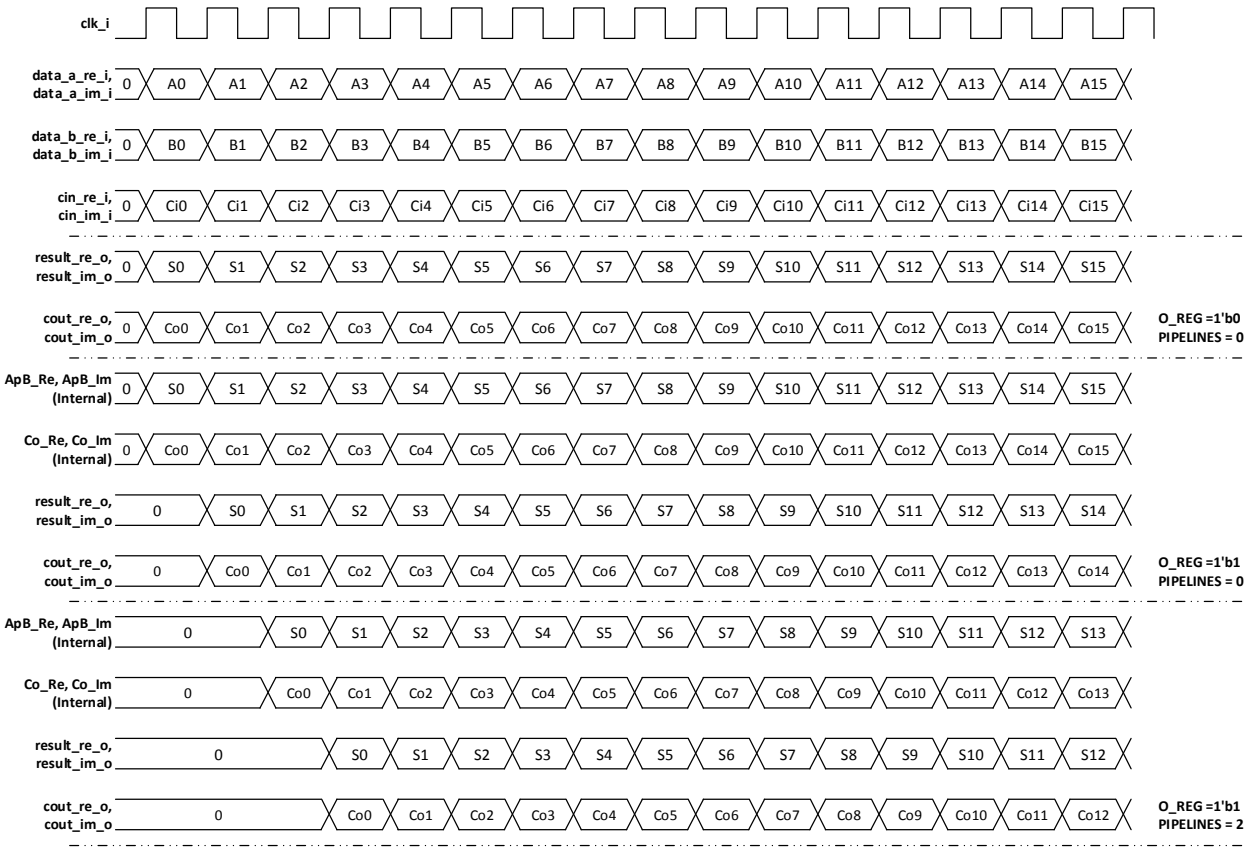

Figure 2.2. Adder Timing Diagram

2.2. Subtractor

A two-input subtractor performs signed/unsigned subtraction of the data from inputs data_a and data_b with an optional Cin borrow input. The output result carries the Difference of the subtraction operation with an optional cout borrow output.

Shaded portions of the Subtractor Schematic Diagram (shown in Figure 2.3) are optional and are removed/ignored in certain configurations.

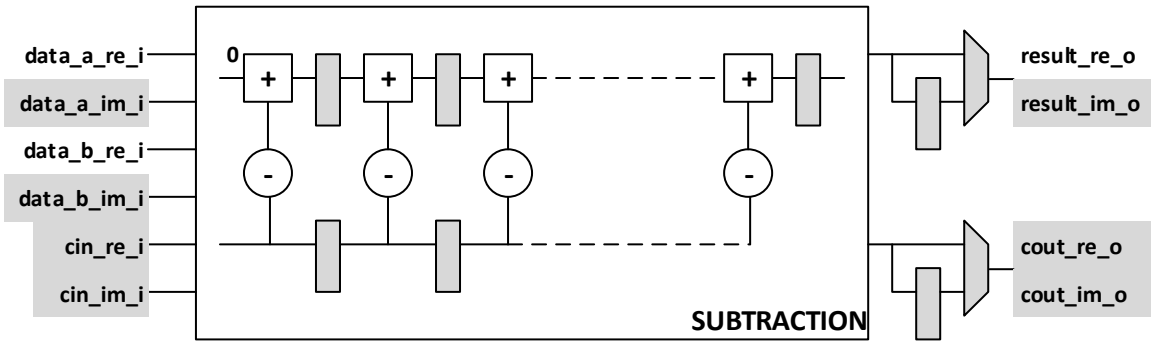

Figure 2.3. Subtractor Schematic Diagram

2.2.1. Ports

Table 2.3. Subtractor Ports

Signal Name	Direction	Width (bits)	Description
clk_i	IN	1	This signal connects to the clock pin of the input, output and pipeline registers. All flops toggle on the rising edge of this clock.
clk_en_i	IN	1	This signal is an active HIGH synchronous clock enable to the output flops in the design. 1'b0 – retain the previous state 1'b1 – toggle on the rising edge of the clock as per the logic
rst_i	IN	1	This signal initializes the flops in the design to a known state. It is an active HIGH asynchronous reset whose deassertion should be synchronized to the clock during integration.
data_a_re_i	IN	D_WIDTH	This signal contains a simple number or the Real part of a complex number which is the minuend in the subtraction. USE_CNUM == 1'b0: Simple Number USE_CNUM == 1'b1: Real part of a complex number
data_a_im_i	IN	D_WIDTH	This signal contains the Imaginary part of a complex number which is the minuend in the subtraction. USE_CNUM == 1'b0: tied to a constant in the integration USE_CNUM == 1'b1: used in the subtraction
data_b_re_i	IN	D_WIDTH	This signal contains a simple number or the Real part of a complex number which is the subtrahend in the subtraction. USE_CNUM == 1'b0: Simple Number USE_CNUM == 1'b1: Real part of a complex number
data_b_im_i	IN	D_WIDTH	This signal contains the Imaginary part of a complex number which is the subtrahend in the subtraction. USE_CNUM == 1'b0: tied to a constant in the integration USE_CNUM == 1'b1: used in the subtraction

Signal Name	Direction	Width (bits)	Description
cin_re_i	IN	D_WIDTH	This signal contains a simple number or the Real part of a complex number which is the Borrow-In in the subtraction. USE_CIN == 1'b0: Borrow-In is unused in the subtraction. Tied to a constant in the integration. USE_CIN == 1'b1: Borrow-In is used in the subtraction
cin_im_i	IN	D_WIDTH	This signal contains the Imaginary part of a complex number which is the Borrow-In in the subtraction. USE_CIN or USE_CNUM == 1'b0: Borrow-In is unused in the subtraction. Tied to a constant in the integration. USE_CIN and USE_CNUM == 1'b1: Borrow-In is used in the subtraction
result_re_o	OUT	D_WIDTH	This signal carries a simple number or the Real part of a complex number from the sum of the subtraction. Core functionality is represented below. {data_a_re_i - data_b_re_i - cin_re_i}
result_im_o	OUT	D_WIDTH	This signal carries the Imaginary part of a complex number from the sum of the subtraction. USE_CNUM == 1'b0: unconnected in the integration USE_CNUM == 1'b1: Core functionality is represented below. {data_a_im_i - data_b_im_i - cin_im_i}
cout_re_o	OUT	D_WIDTH	This signal carries a simple number or the Real part of a complex number from the Borrow-Out of the subtraction. USE_COUT == 1'b0: unconnected in the integration USE_COUT == 1'b1: Borrow-Out is implemented SIGNED == Unsigned: treated as borrow-out SIGNED == Signed: treated as underflow This signal when HIGH indicates an underflow in the difference i.e. a minuend (data_a_re_i) < subtrahend (data_b_re_i)
cout_im_o	OUT	D_WIDTH	This signal carries the Imaginary part of a complex number from the Borrow-Out of the subtraction. USE_COUT or USE_CNUM == 1'b0: unconnected in the integration USE_COUT and USE_CNUM == 1'b1: Borrow-out is implemented SIGNED == Unsigned: treated as borrow-out SIGNED == Signed: treated as underflow This signal when HIGH indicates an underflow in the difference i.e. a minuend (data_a_im_i) < subtrahend (data_b_im_i)

2.2.2. Attributes

Table 2.4. Subtractor Attributes

Configuration	Range	Default Value	Description
D_WIDTH	1–64	16	This configuration is the data width of the input/output ports.
SIGNED	Signed or Unsigned	Unsigned	This configuration controls the interpretation of the inputs whether a signed or an unsigned number.
USE_CNUM	0–1	0	This configuration controls the interpretation of the numbers being involved in the subtraction. 0 – Simple Numbers. data*_re_i, cin_re_i, result_re_o and cout_re_o are treated as simple numbers only while data*_im_i, cin_im_i, result_im_o and cout_im_o IOs are ignored. 1 – Complex Numbers. The logic is implemented on the basis of the IOs as complex numbers.
USE_CIN	0–1	0	This configuration controls the usage of Borrow-In in the subtraction. 0 – Ignore Borrow-In. cin_re_i and cin_im_i inputs are unused. 1 – Consider Borrow-In. cin_re_i and cin_im _i inputs are used.
USE_COUT	0–1	0	This configuration controls the implementation of Carry-Out from the subtraction. 0 – Ignore Carry-Out. cout_re_o and cout_im_o outputs are always driven LOW. 1 – Consider Carry-Out. cout_re_o and cout_im_o outputs are driven with carry.
USE_IREG	“on”, “off”	“off”	This configuration controls the registering of the adder inputs. “off” – Unregistered. Input is directly routed to internal logic. “on” – Registered. Input is registered before being used by internal logic. This configuration improves static timing of the design when combinational delay from the adder is high.
USE_OREG	“on”, “off”	“off”	This configuration controls the registering of the adder result onto the outputs. “off” – Unregistered. Result is directly routed to the output. “on” – Registered. Result is registered at the output. This configuration improves static timing of the design when combinational delay from the adder is high.
PIPELINES	0-(D_WIDTH/8)-1	1	This configuration controls the insertion of pipeline stages into the subtraction operation. This configuration accepts integers only and any value less than 1 is treated as no pipelining by the design. Maximum pipelines permitted is dependent on the number of 8-bit chunks in the design computed IO Width configuration. For example: 1–8 bits: 0 9–16 bits: 1 17–24 bits: 2 25–32 bits: 3 and so on

2.2.3. Timing Diagram

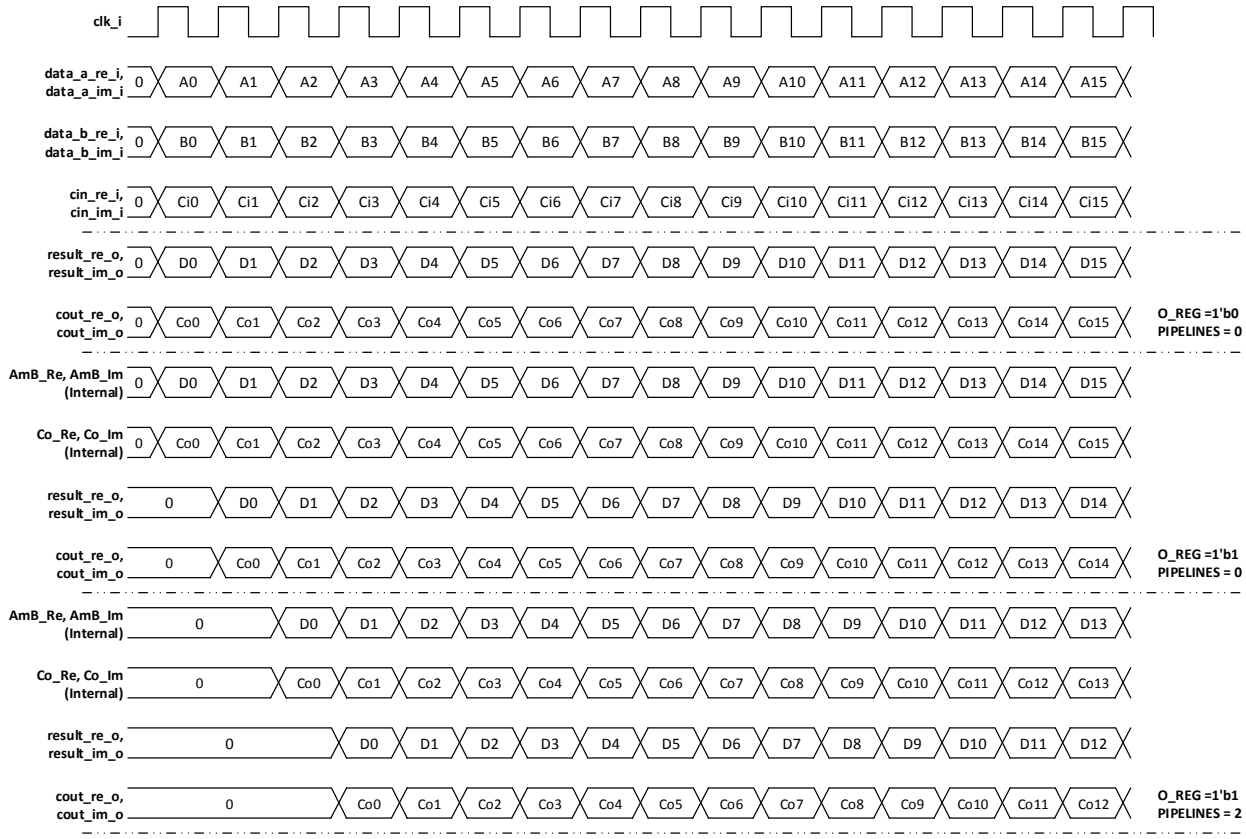

Figure 2.4. Subtractor Timing Diagram

2.3. Multiplier

A two-input multiplier performs signed/unsigned multiplication of the data from inputs data_a and data_b with an optional constant multiplier. The output result carries the Product of the multiplication operation.

The shaded portions of the Multiplier Schematic Diagram (shown in Figure 2.5) are optional and are removed/ignored in certain configurations.

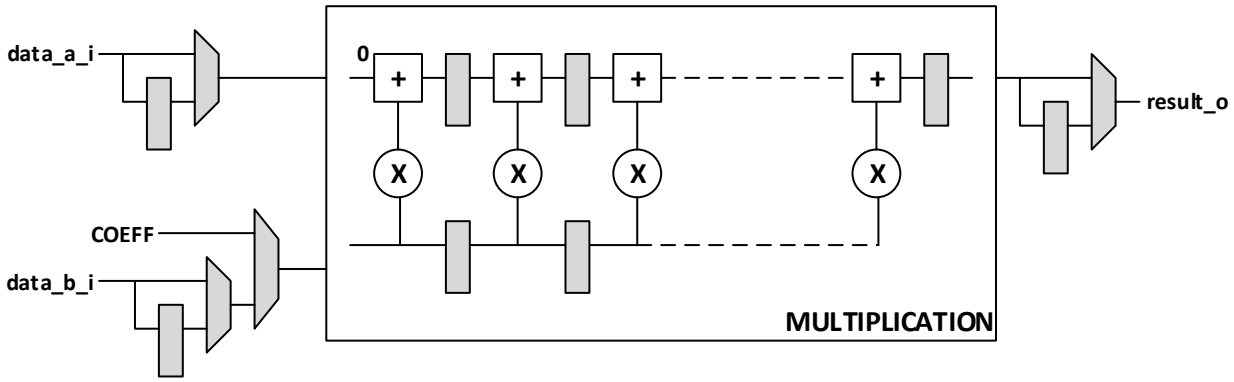

Figure 2.5. Multiplier Schematic

2.3.1. Ports

Table 2.5. Multiplier Ports

Signal Name	Direction	Width (bits)	Description
clk_i	IN	1	This signal connects to the clock pin of the input, output and pipeline registers. All flops toggle on the rising edge of this clock.
clk_en_i	IN	1	This signal is an active HIGH synchronous clock enable to the output flops in the design. 1'b0 - retain the previous state 1'b1 - toggle on the rising edge of the clock as per the logic
rst_i	IN	1	This signal initializes the flops in the design to a known state. It is an active HIGH asynchronous reset whose deassertion should be synchronized to the clock during integration.
data_a_i	IN	A_WIDTH	This signal contains a simple number or the Real part of a complex number which is the minuend in the subtraction. USE_CNUM == 1'b0: Simple Number USE_CNUM == 1'b1: Real part of a complex number
data_b_i	IN	B_WIDTH	This signal contains the Imaginary part of a complex number which is the minuend in the subtraction. USE_CNUM == 1’b0: tied to a constant in the integration USE_CNUM == 1’b1: used in the subtraction
result_o	OUT	M_WDT	This signal carries the product value of the multiplication. Core functionality is represented below. data_a * (USE_COEFF ? COEFF : data_b)

2.3.2. Attributes

Table 2.6. Multiplier Attributes

Configuration	Range	Default Value	Description
USE_COEFF	0–1	0	This configuration controls the multiplier input to the multiplication operation in the design. 1'b0 – data_b input of the design is the multiplier 1'b1 – COEFF configuration parameter is the multiplier
COEFF	-2^{31} to (2^{31}–1)	2	This configuration contains the co-efficient value used in the multiplication. It is considered valid only when the USE_COEFF parameter is configured HIGH. The configuration accepts signed integers. Width of the output, result is computed based on the bits required to represent the absolute value of this parameter.
A_WIDTH	2–64	9	This configuration controls the width of the input data_a.
B_WIDTH	2–64	9	This configuration controls the width of the input data_b.
A_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_a. 1'b0 – Unsigned number 1'b1 – Signed number
B_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_b. 1'b0 – Unsigned number 1'b1 – Signed number
USE_IREG	“on”, “off”	“on”	This configuration controls the registering of the inputs, data_a & data_b before routing them to the multiplication operation. off – Unregistered. Inputs are directly routed. on – Registered. Inputs are registered before routing. This configuration improves static timing of the design when insertion delay to the multiplier is high.
USE_OREG	“on”, “off”	“on”	This configuration controls the registering of the multiplier result onto the output, result. off – Unregistered. Result is directly routed to the output. on – Registered. Result is registered at the output. This configuration improves static timing of the design when combinational delay from the multiplier is high.
PIPELINES	0–N	1	This configuration controls the insertion of pipeline stages into the multiplier operation. This configuration accepts integers only and any value less than 1 is treated as no pipelining by the design. If this parameter is configured with a value greater than the value given below, the design automatically corrects it to the maximum limit. Otherwise, the configured value is honored. USE_COEFF == 1'b0: greater of the design computed parameters, A_WIDTH & B_WIDTH is the limiting factor USE_COEFF == 1'b1: design computed parameter, A_WIDTH is the limiting factor

Configuration	Range	Default Value	Description
IMPLEMENTATION	“LUT”, “DSP”	“LUT”	This configuration selects the resource that will be used for implementation. LUT – Use LUT DSP – Use DSP blocks
Internal Parameters			
COEFF_WDT	—	—	Actual bit width of the absolute (unsigned) COEFF value
X_WDT	—	—	USE_COEFF ? COEFF_WDT : B_WDT
M_WDT	—	—	A_WDT + X_WDT

2.3.3. Timing Diagram

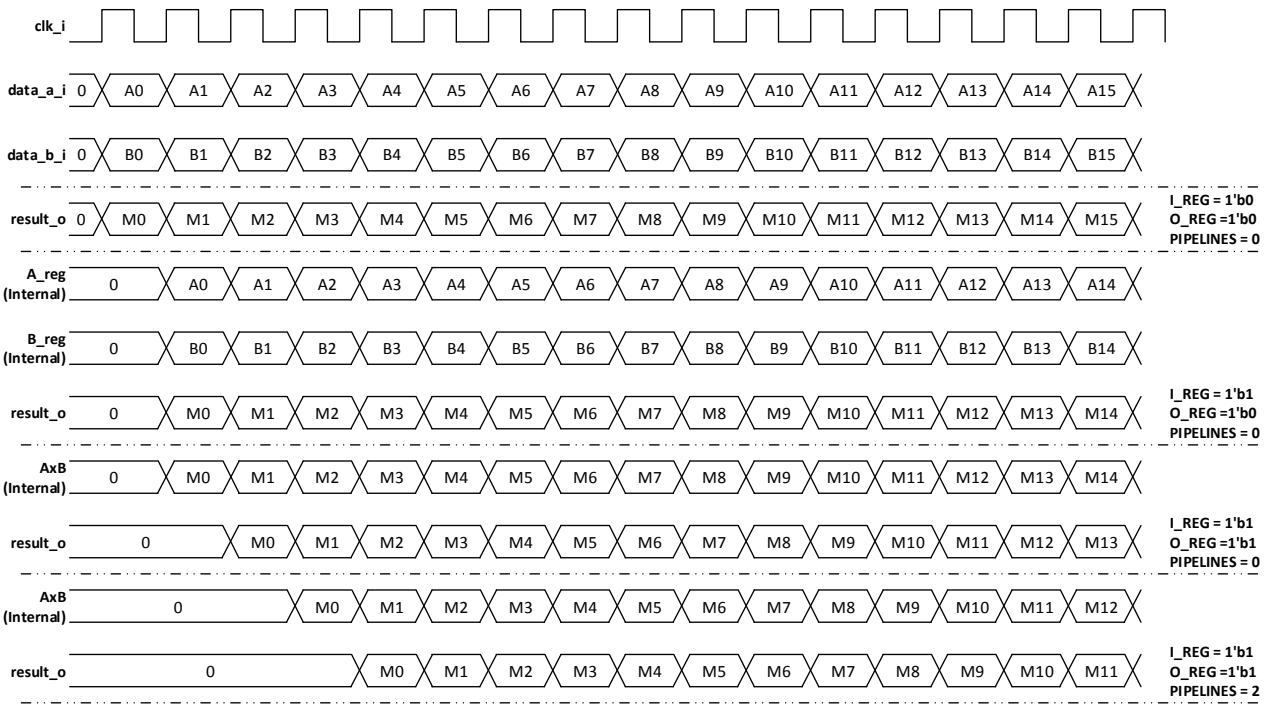

Figure 2.6. Multiplier Timing Diagram

2.4. Multiply_Accumulate

A two-input multiplier with accumulate performs signed/unsigned multiplication of the data from inputs data_a and data_b and accumulates the result. The output result carries the Accumulation of Products.

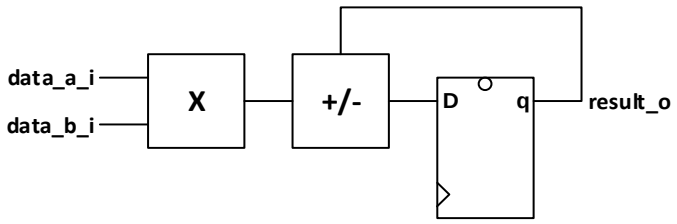

Figure 2.7. Multiply_Accumulate Schematic

2.4.1. Ports

Table 2.7. Multiply_Accumulate Ports

Signal Name	Direction	Width (bits)	Description
clk_i	IN	1	This signal connects to the clock pin of the input, output and pipeline registers. All flops toggle on the rising edge of this clock.
clk_en_i	IN	1	This signal is an active HIGH synchronous clock enable to the output flops in the design. 1'b0 - retain the previous state 1'b1 - toggle on the rising edge of the clock as per the logic
rst_i	IN	1	This signal initializes the flops in the design to a known state. It is an active HIGH asynchronous reset whose deassertion should be synchronized to the clock during integration.
data_a_i	IN	A_WIDTH	This signal contains a simple number or the Real part of a complex number which is the minuend in the subtraction. USE_CNUM == 1'b0: Simple Number USE_CNUM == 1'b1: Real part of a complex number
data_b_i	IN	B_WIDTH	This signal contains the Imaginary part of a complex number which is the minuend in the subtraction. USE_CNUM == 1'b0: tied to a constant in the integration USE_CNUM == 1'b1: used in the subtraction
result_o	OUT	M_WDT	This signal carries the sum of the product of inputs data_a and data_b. Core functionality is represented below. $\sum_{i=0}^{n} DataA_i * DataB_i$

2.4.2. Attributes

Table 2.8. Multiply_Accumulate Attributes

Configuration	Range	Default Value	Description
A_WIDTH	2–64	18	This configuration controls the width of the input data_a.
B_WIDTH	2–64	18	This configuration controls the width of the input data_b.
A_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_b. 1'b0 - Unsigned number 1'b1 - Signed number
B_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_b. 1'b0 - Unsigned number 1'b1 - Signed number
ADD_SUB	“add” or “sub”	“add”	This configuration controls the operation to be performed on the accumulated value and multiplication product. sub – Subtraction add – Addition
USE_IREG	“on”, “off”	“on”	This configuration controls the registering of the inputs, data_a & data_b before routing them to the multiplication operation. off - Unregistered. Inputs are directly routed. on - Registered. Inputs are registered before routing. This configuration improves static timing of the design when insertion delay to the multiplier is high.
USE_OREG	“on”, “off”	“on”	This configuration controls the registering of the multiplier result onto the output, result. off - Unregistered. Result is directly routed to the output. on - Registered. Result is registered at the output. This is fixed to “on” to improve static timing of the design.
PIPELINES	0–N	1	This configuration controls the insertion of pipeline stages into the multiplier operation. This configuration accepts integers only and any value less than 1 is treated as no pipelining by the design. If this parameter is configured with a value greater than the greatest of the design computed values, A_WIDTH & B_WIDTH, the design automatically corrects it to the greater of A_WIDTH & B_WIDTH.
IMPLEMENTATION	“LUT”, “DSP”	“LUT”	This configuration selects the resource that will be used for implementation. LUT – Use LUT DSP – Use DSP blocks
Internal Parameters			
M_WDT	—	—	A_WDT + X_WDT

2.4.3. Timing Diagram

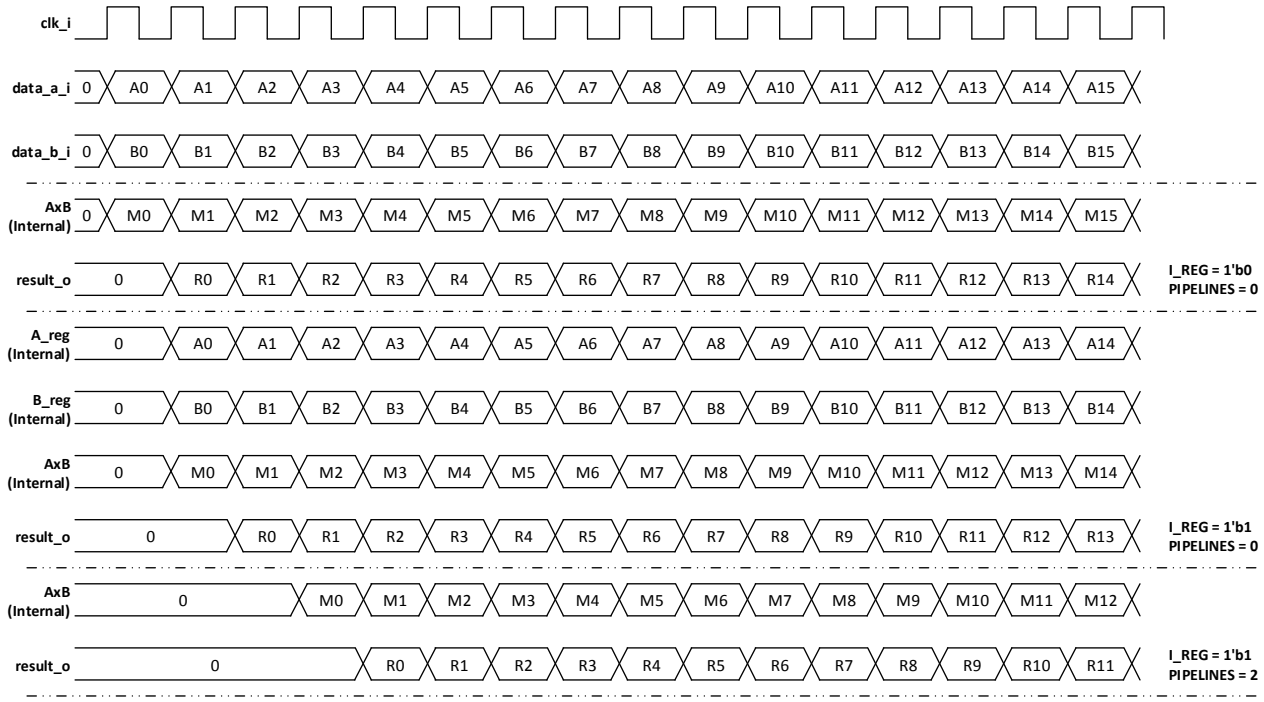

Figure 2.8. Multiply_Accumulate Timing Waveforms

2.5. Multiply_Add_Subtract

A pair of two-input multipliers that performs signed/unsigned multiplication of the data from inputs data_a and data_b with addition/subtraction on the product pair. The output result carries the Sum/Difference of Products.

The shaded portions of the Multiply_Add_Subtract Schematic Diagram (shown in Figure 2.9) are optional and are removed/ignored in certain configurations.

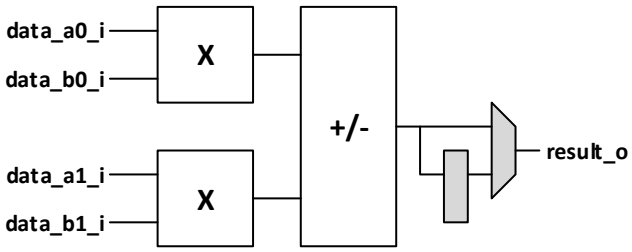

Figure 2.9. Multiply_Add_Subtract Schematic

2.5.1. Ports

Table 2.9. Multiply_Add_Subtract Ports

Signal Name	Direction	Width (bits)	Description
clk_i	IN	1	This signal connects to the clock pin of the input, output and pipeline registers. All flops toggle on the rising edge of this clock.
clk_en_i	IN	1	This signal is an active HIGH synchronous clock enable to the output flops in the design. 1'b0 – retain the previous state 1'b1 – toggle on the rising edge of the clock as per the logic
rst_i	IN	1	This signal initializes the flops in the design to a known state. It is an active HIGH asynchronous reset whose deassertion should be synchronized to the clock during integration.
data_a0_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication pair data_a0-data_b0.
data_a1_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication pair data_a1-data_b1.
data_b0_i	IN	B_WIDTH	This signal contains the multiplicand value of the multiplication pair data_a0-data_b0.
data_b1_i	IN	B_WIDTH	This signal contains the multiplicand value of the multiplication pair data_a1-data_b1.
result_o	OUT	M_WDT	This signal carries the result of the addition/subtraction of the multiplication products. Core functionality is represented below. M0 = (data_a0 * data_b0) M1 = (data_a1 * data_b1) ADD_SUB== add: {M0 + M1} ADD_SUB== sub: {M0 – M1}

2.5.2. Attributes

Table 2.10. Multiply_Add_Subtract Attributes

Configuration	Range	Default Value	Description
ADD_SUB	“add” or “sub”	“add”	This configuration controls the operation to be performed on the multiplication product pairs, data_a0-data_b0 & data_a1-data_b1. sub – Subtraction add – Addition
A_WIDTH	2–64	9	This configuration controls the width of the inputs data_a0 and data_a1.
B_WIDTH	2–64	9	This configuration controls the width of the inputs data_b0 and data_b1.
A_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_a0 & data_a1. 1'b0 – Unsigned number 1'b1 – Signed number

Configuration	Range	Default Value	Description
B_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_b0 & data_b1. 1'b0 – Unsigned number 1'b1 – Signed number
USE_IREG	“on”, “off”	“on”	This configuration controls the registering of the inputs, data_a* & data_b* before routing them to the multiplication operation. off – Unregistered. Inputs are directly routed. on – Registered. Inputs are registered before routing. This configuration improves static timing of the design when insertion delay to the multiplier is high.
USE_OREG	“on”, “off”	“on”	This configuration controls the registering of the output, result. off – Unregistered. Result is directly routed to the output. on – Registered. Result is registered at the output. This configuration improves static timing of the design when combinational delay from the multiplier is high.
PIPELINES	0–N	1	This configuration controls the insertion of pipeline stages into the multiplier operation. This configuration accepts integers only and any value less than 1 is treated as no pipelining by the design. If this parameter is configured with a value greater than the greatest of the design computed values, A_WIDTH & B_WIDTH, the design automatically corrects it to the greater of A_WIDTH & B_WIDTH.
IMPLEMENTATION	“LUT”, “DSP”	“LUT”	This configuration selects the resource that will be used for implementation. LUT – Use LUT DSP – Use DSP blocks
Internal Parameters			
M_WDT	—	—	A_WDT + X_WDT
O_WDT	—	—	M_WDT + 1

2.5.3. Timing Diagram

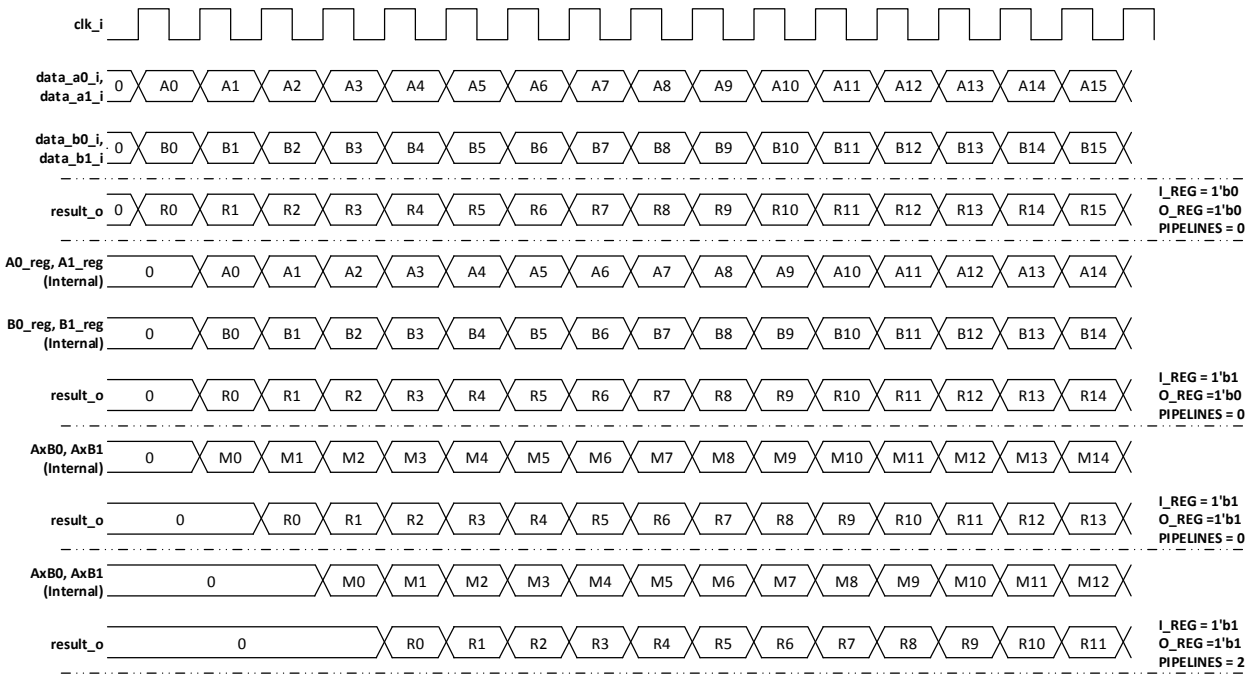

Figure 2.10. Multiply_Add_Subtract Timing Diagram

2.6. Complex_Multiplier

A two-input multiplier that performs signed/unsigned multiplication of complex numbers, input through data_a and data_b ports. The output result carries the Product of the multiplication operation.

The shaded portions of the Complex_Multiplier Schematic Diagram (shown in Figure 2.11) are optional and are removed/ignored in certain configurations.

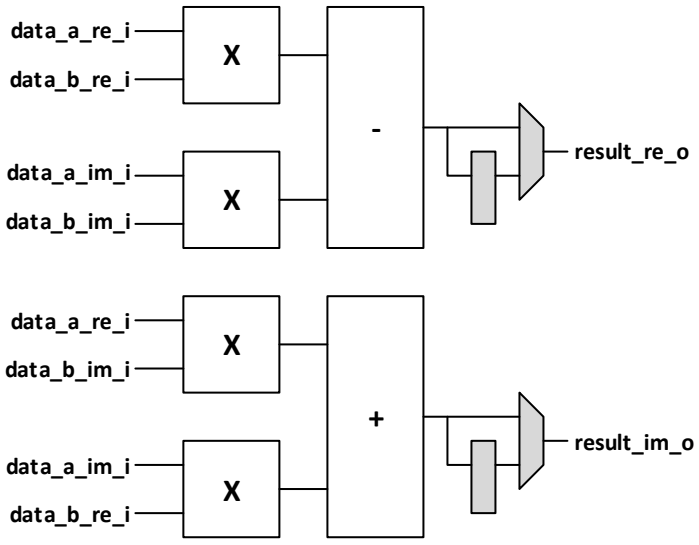

Figure 2.11. Complex_Multiplier Schematic Diagram

2.6.1. Ports

Table 2.11. Complex_Multiplier Ports

Signal Name	Direction	Width (bits)	Description
clk_i	IN	1	This signal connects to the clock pin of the input, output and pipeline registers. All flops toggle on the rising edge of this clock.
clk_en_i	IN	1	This signal is an active HIGH synchronous clock enable to the output flops in the design. 1'b0 – retain the previous state 1'b1 – toggle on the rising edge of the clock as per the logic
rst_i	IN	1	This signal initializes the flops in the design to a known state. It is an active HIGH asynchronous reset whose deassertion should be synchronized to the clock during integration.
data_a_re_i	IN	D_WIDTH	This signal contains the Real part of the complex number which is the multiplicand in the multiplication
data_a_im_i	IN	D_WIDTH	This signal contains the Imaginary part of the complex number which is the multiplicand in the multiplication.
data_b_re_i	IN	D_WIDTH	This signal contains the Real part of the complex number which is the multiplier in the multiplication.
data_b_im_i	IN	D_WIDTH	This signal contains the Imaginary part of the complex number which is the multiplier in the multiplication.
result_re_o	OUT	D_WIDTH	This signal carries the Real part of the complex number from the product of the multiplication. Core functionality is represented below. {(data_a_re_i * data_b_re_i) - (data_a_im_i * data_b_im_i)}
result_im_o	OUT	D_WIDTH	This signal carries the Imaginary part of the complex number from the product of the multiplication. Core functionality is represented below. {(data_a_re_i * data_b_im_i) + (data_a_im_i * data_b_re_i)}

2.6.2. Attributes

Table 2.12. Complex_Multiplier Attributes

Configuration	Range	Default Value	Description
A_WIDTH	2–64	8	This configuration controls the width of the inputs, data_a_re_i and data_a_im_i.
B_WIDTH	2–64	8	This configuration controls the width of the inputs, data_b_re_i and data_b_im_i.
SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the inputs data_a_re_i, data_a_im_i, data_b_re_i and data_b_im_i. off – Unsigned number on – Signed number

Configuration	Range	Default Value	Description
USE_IREG	“on”, “off”	“on”	This configuration controls the registering of the inputs, data_a_re_i, data_a_im_i, data_b_re_i and data_b_im_i before routing them to the multiplication operation. off – Unregistered. Inputs are directly routed. on – Registered. Inputs are registered before routing. This configuration improves static timing of the design when insertion delay to the multiplier is high.
USE_OREG	“on”, “off”	“on”	This configuration controls the registering of the outputs, result_re_o and result_im_o. off – Unregistered. Result is directly routed to the output. on – Registered. Result is registered at the output. This configuration improves static timing of the design when combinational delay from the multiplier is high.
PIPELINES	0–N	1	This configuration controls the insertion of pipeline stages into the multiplier operation. This configuration accepts integers only and any value less than 1 is treated as no pipelining by the design. If this parameter is configured with a value greater than the greatest of the design computed values, A_WIDTH and B_WIDTH, the design automatically corrects it to the greater of A_WIDTH and B_WIDTH.
IMPLEMENTATION	“LUT”, “DSP”	“LUT”	This configuration selects the resource that will be used for implementation. LUT – Use LUT DSP – Use DSP blocks
Internal Parameters			
M_WDT	—	—	A_WDT + X_WDT
O_WDT	—	—	M_WDT + 1

2.6.3. Timing Diagram

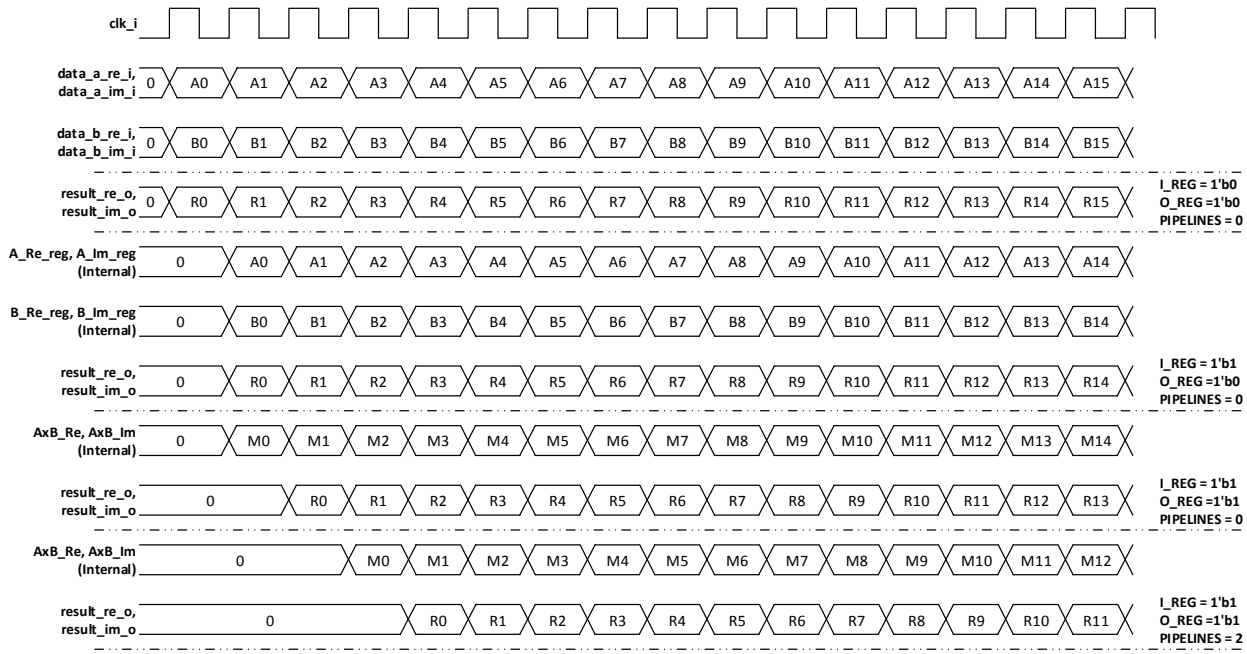

Figure 2.12. Complex_Multiplier Timing Diagram

3. IP Generation

Module/IP Block Wizard in Lattice Radiant Software allows you to generate, create, or open modules for the target device. From the Lattice Radiant Software, select the **IP Catalog** tab as shown in Figure 3.1.

The left pane of the IP Catalog window displays the module/IP tree. You can utilize this to specify a variety of arithmetic modules in your designs.

The right pane of the window shows the description of the selected module and provides link(s) to the documentation.

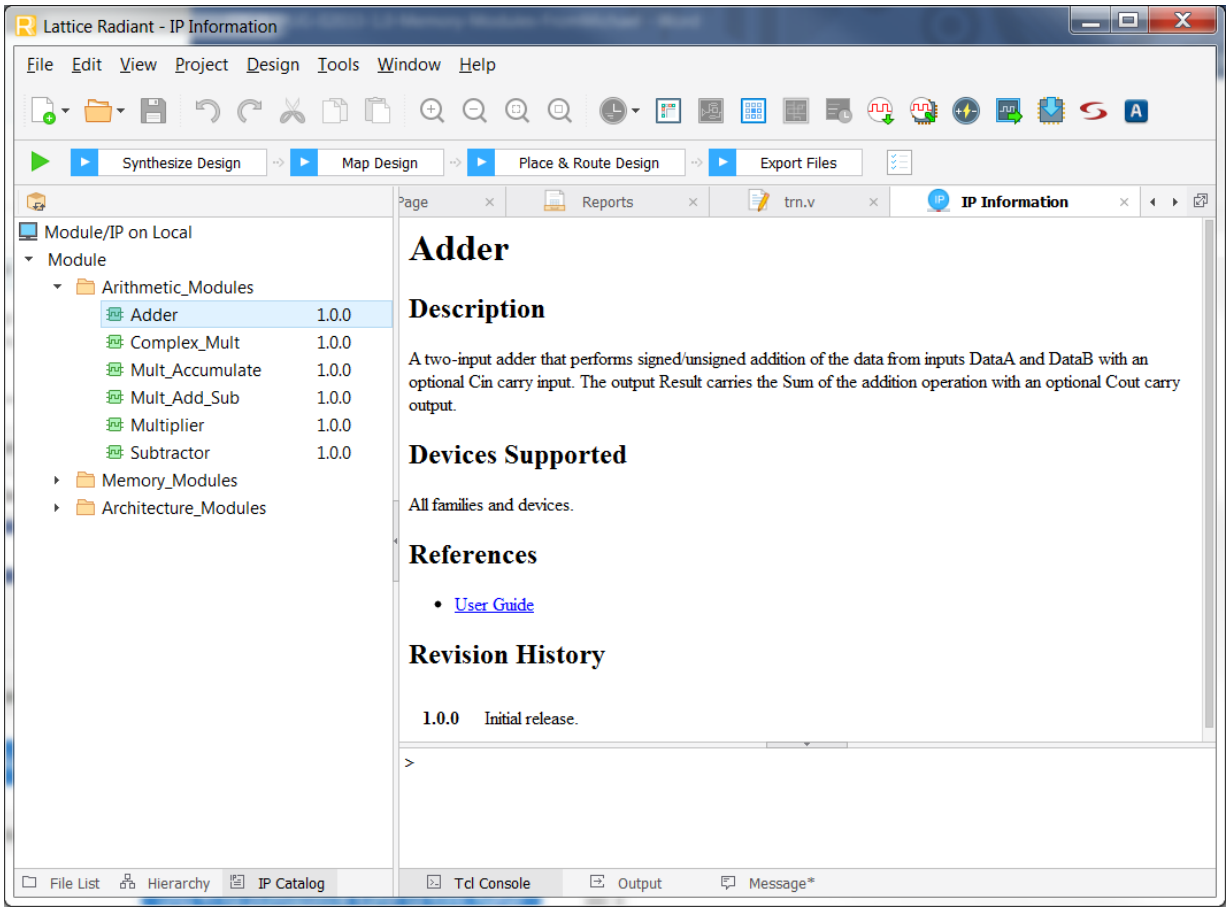

Figure 3.1. Arithmetic Modules under Module/IP on Local in the Lattice Radiant Software

The following section shows an example of generating a 32-bit Multiplier module. This process is similar for all Arithmetic Modules.

1. Double-click **Multiplier** under **Arithmetic_Modules**. This opens the Module/IP Block Wizard.
2. Fill out the following information then click **Next**. An example is shown in Figure 3.2.
   - **Language**
   - **Instance name**
   - **Create in**

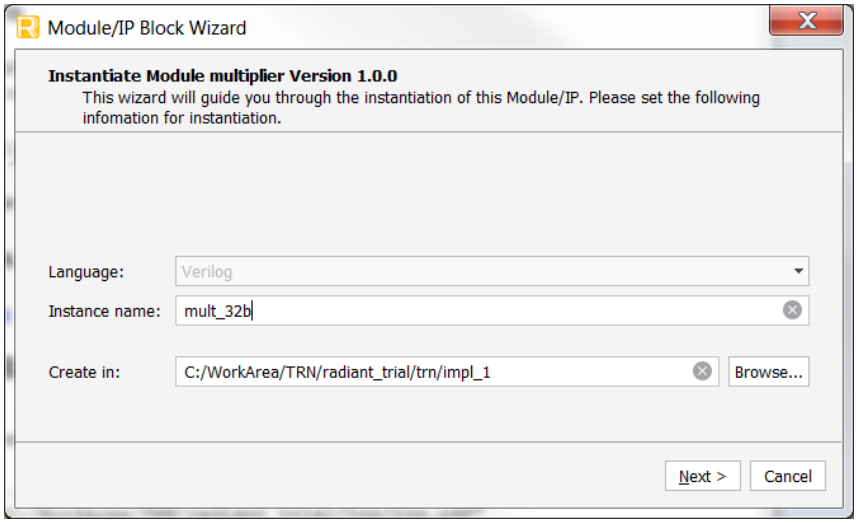

Figure 3.2. Example: Generating 32-bit Multiplier Using Module/IP Block Wizard

3. In the **IP Configuration** page, customize the Multiplier by selecting options as shown in Figure 3.3.

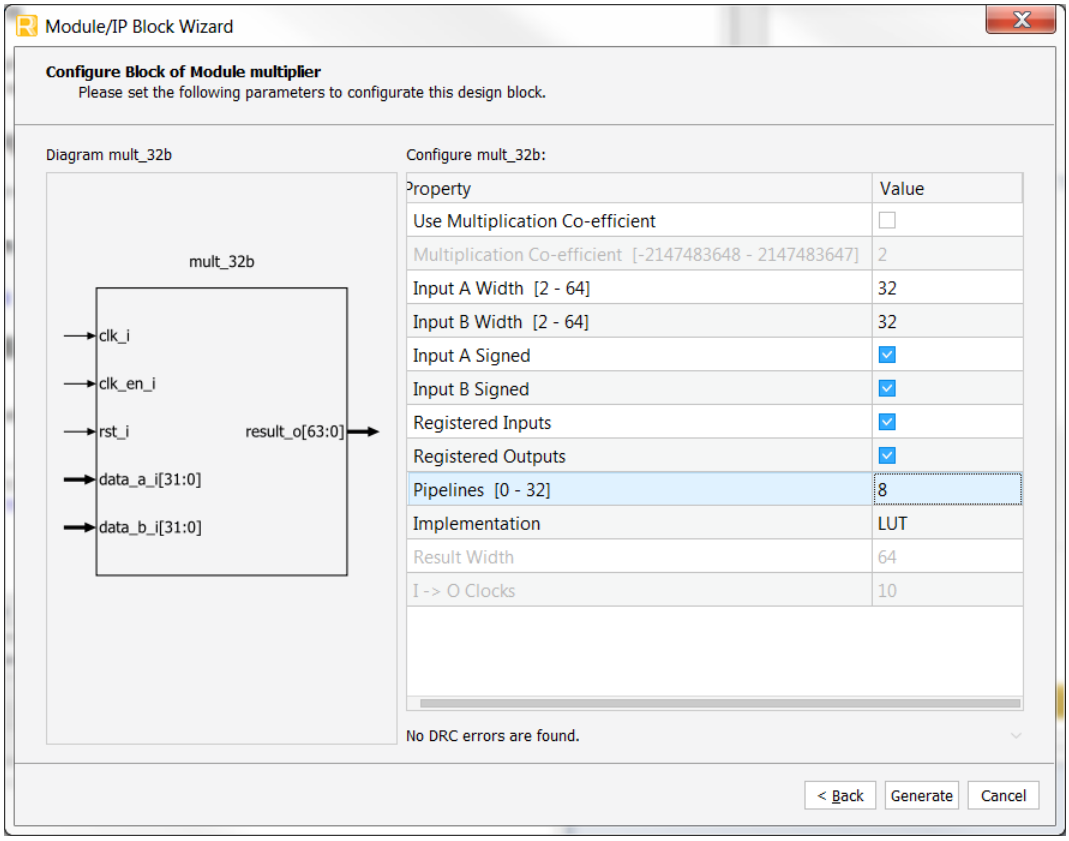

Figure 3.3. Example: Generating 32-bit Multiplier in IP Configuration

4. When all the options are set, click **Generate**.

Once this module is in the Lattice Radiant Software project, it can be instantiated in other modules within the project.

4. PMI Support

Using PMI (Parameterized Module Instantiation), you can set the parameters and control signals directly in Verilog/VHDL code. Refer to the Using PMI section of Lattice Radiant Help for more information on PMI.

The following sections discuss the different PMI modules which can be utilized for entering custom parameters for Arithmetic Modules. If parameters are left unspecified during module instantiation, default values will be used.

4.1. pmi_add

The following are port descriptions, attribute descriptions, Verilog definition and VHDL definition for pmi_add.

Table 4.1. Port Definitions for pmi_add

Direction	Port Name	Type	Size (buses only)
I	DataA	Bus	(pmi_data_width - 1):0
I	DataB	Bus	(pmi_data_width - 1):0
I	Cin	Bit	N/A
O	Result	Bus	(pmi_data_width - 1):0
O	Cout	Bit	N/A
O	Overflow	Bit	N/A

Table 4.2. Attribute Definitions for pmi_add

Attributes	Description	Values	Default Value
pmi_data_width	Defines Bit width of DataA, DataB and Result ports	2–64	16
pmi_sign	This configuration controls the interpretation of the inputs whether a signed or an unsigned number.	“on”, "off"	"on"
pmi_family	Defines the FPGA family being used in the module	"common"	"common"
module_type	Refers to the type of pmi module	“pmi_add”	“pmi_add”

Verilog Definition for `pmi_add`

```
module pmi_add
#(
    parameter pmi_data_width = 16,
    parameter pmi_sign       = “on”,
    parameter pmi_family     = “common”,
    parameter module_type    = “pmi_add”
)
(
    input [pmi_data_width-1:0]   DataA,
    input [pmi_data_width-1:0]   DataB,
    input                        Cin,

    output [pmi_data_width-1:0]  Result,
    output                       Cout,
    output wire                  Overflow
```

```
)/*synthesis syn_black_box */;
endmodule // pmi_add
```

VHDL Definition `pmi_add`

```
component pmi_add is
    generic(
        pmi_data_width : integer := 16;
        pmi_sign       : string  := "on";
        pmi_family     : string  := "common";
        module_type    : string  := "pmi_add"
    )
    (
        DataA    : in  std_logic_vector(pmi_data_width-1 downto 0);
        DataB    : in  std_logic_vector(pmi_data_width-1 downto 0);
        Cin      : in  std_logic;
        Result   : out std_logic_vector(pmi_data_width-1 downto 0);
        Cout     : out std_logic;
        Overflow : out std_logic
    );
end component pmi_add;
```

4.2. pmi_sub

The following are port descriptions, attribute descriptions, Verilog definition and VHDL definition for pmi_sub.

Table 4.3. Port Definitions for pmi_sub

Direction	Port Name	Type	Size (buses only)
I	DataA	Bus	(pmi_data_width - 1):0
I	DataB	Bus	(pmi_data_width - 1):0
I	Cin	Bit	N/A
O	Result	Bus	(pmi_data_width - 1):0
O	Cout	Bit	N/A
O	Overflow	Bit	N/A

Table 4.4. Attribute Definitions for pmi_sub

Attributes	Description	Values	Default Value
pmi_data_width	Defines Bit width of DataA, DataB and Result ports	2–64	16
pmi_sign	Controls the interpretation of the inputs whether a signed or an unsigned number.	“on”, "off"	"on"
pmi_family	Defines the FPGA family being used in the module	"common"	"common"
module_type	Refers to the type of pmi module	“pmi_sub”	“pmi_sub”

Verilog Definition for `pmi_sub`

```
module pmi_sub
#(
    parameter pmi_data_width = 16,
    parameter pmi_sign       = "on",
    parameter pmi_family     = "common",
    parameter module_type    = "pmi_sub"
)
(
    input [pmi_data_width-1:0]   DataA,
    input [pmi_data_width-1:0]   DataB,
    input                        Cin,
    output [pmi_data_width-1:0]  Result,
    output                       Cout,
    output wire                  Overflow
) /*synthesis syn_black_box */;
endmodule // pmi_sub
```

VHDL Definition `pmi_sub`

```
component pmi_sub is
    generic(
        pmi_data_width : integer := 16;
        pmi_sign       : string  := "on";
        pmi_family     : string  := "common";
        module_type    : string  := "pmi_sub"
    )
    (
        DataA    : in  std_logic_vector(pmi_data_width-1 downto 0);
        DataB    : in  std_logic_vector(pmi_data_width-1 downto 0);
        Cin      : in  std_logic;
        Result   : out std_logic_vector(pmi_data_width-1 downto 0);
        Cout     : out std_logic;
        Overflow : out std_logic
    );
end component pmi_sub;
```

4.3. pmi_mult

The following are port descriptions, attribute descriptions, Verilog definition and VHDL definition for pmi_mult.

Table 4.5. Port Definitions for pmi_mult

Direction	Port Name	Type	Size (buses only)
I	DataA	Bus	(pmi_dataa_width - 1):0
I	DataB	Bus	(pmi_datab_width - 1):0
I	Clock	Bit	N/A
I	ClkEn	Bit	N/A
I	Aclr	Bit	N/A
O	Result	Bus	(pmi_accum_width - 1):0

Table 4.6. Attribute Definitions for pmi_mult

Attributes	Description	Values	Default Value
pmi_dataa_width	Defines Bit width of DataA port	2–36	9
pmi_datab_width	Defines Bit width of DataB port	2–36	9
pmi_additional_pipeline	Specifies the number of pipeline stages. If pmi_implementation == LUT; N = 9 If pmi_implementation == DSP; N = 3	0–N	1
pmi_input_reg	Enables input register	“on”, “off”	“on”
pmi_output_reg	Enables output register	“on”, “off”	“on”
pmi_sign	Controls the interpretation of the inputs whether a signed or an unsigned number.	“on”, "off"	"on"
pmi_family	Defines the FPGA family being used in the module	"common"	"common"
pmi_implementation	Selects the resource that will be used for implemention.	“DSP”, ”LUT”	“LUT”
module_type	Refers to the type of pmi module	“pmi_mult”	“pmi_mult”

Verilog Definition for `pmi_mult`

```
module pmi_mult
#(
    parameter pmi_dataa_width         = 9,
    parameter pmi_datab_width         = 9,
    parameter pmi_sign                = "on",
    parameter pmi_additional_pipeline = 1,
    parameter pmi_input_reg           = "on",
    parameter pmi_output_reg          = "on",
    parameter pmi_family              = "common",
    parameter pmi_implementation      = "LUT",
    parameter module_type             = "pmi_mult"
)
(
    input                             Clock,
    input                             ClkEn,
    input                             Aclr,
    input   [(pmi_dataa_width-1):0]   DataA,
    input   [(pmi_datab_width-1):0]   DataB,
    output  [(pmi_dataa_width + pmi_datab_width - 1):0]  Result
)/*synthesis syn_black_box*/;
endmodule // pmi_mult
```

VHDL Definition for `pmi_mult`

```
component pmi_mult is
    generic (
        pmi_dataa_width         : integer := 9;
        pmi_datab_width         : integer := 9;
        pmi_sign                : string := "on";
        pmi_additional_pipeline : integer := 1;
        pmi_input_reg           : string := "on";
        pmi_output_reg          : string := "on";
        pmi_family              : string := "common";
        pmi_implementation      : string := "LUT";
        module_type             : string := "pmi_mult"
    );
    port (
        Clock  : in std_logic;
        ClkEn  : in std_logic;
        Aclr   : in std_logic;
        DataA  : in std_logic_vector((pmi_dataa_width-1) downto 0);
        DataB  : in std_logic_vector((pmi_datab_width-1) downto 0);
        Result : out std_logic_vector((pmi_dataa_width + pmi_datab_width - 1)
downto 0)
    );
end component pmi_mult;
```

4.4. pmi_mac

The following are port descriptions, attribute descriptions, Verilog definition and VHDL definition for pmi_mac.

Table 4.7. Port Definitions for pmi_mac

Direction	Port Name	Type	Size (buses only)
I	DataA	Bus	(pmi_dataa_width - 1):0
I	DataB	Bus	(pmi_datab_width - 1):0
I	Clock	Bit	N/A
I	ClkEn	Bit	N/A
I	Aclr	Bit	N/A
O	Result	Bus	(pmi_accum_width - 1):0

Table 4.8. Attribute Definitions for pmi_mac

Attributes	Description	Values	Default Value
pmi_dataa_width	Defines Bit width of DataA port	2–36	18
pmi_datab_width	Defines Bit width of DataB port	2–36	18
pmi_accum_width	Defines Bit width of Result port	1–32	16
pmi_additional_pipeline	Specifies the number of pipeline stages. If pmi_implementation == LUT; N = 9 If pmi_implementation == DSP; N = 3	0–N	1
pmi_add_sub	Controls the operation to be performed on the accumulated value and multiplication product.	“add”, “sub”	“add”
pmi_input_reg	Enables input register	“on”, “off”	“off”
pmi_sign	Controls the interpretation of the inputs whether a signed or an unsigned number.	“on”, "off"	"on"
pmi_family	Defines the FPGA family being used in the module	"common"	"common"
pmi_implementation	Selects the resource that will be used for implemention.	“DSP”, ”LUT”	“LUT”
module_type	Refers to the type of pmi module	“pmi_mac”	“pmi_mac”

Verilog Definition for `pmi_mac`

```
module pmi_mac
#(
    parameter pmi_dataa_width         = 18,
    parameter pmi_datab_width         = 18,
    parameter pmi_accum_width         = 16,
    parameter pmi_sign                = "on",
    parameter pmi_additional_pipeline = 1,
    parameter pmi_add_sub             = “add”,
    parameter pmi_input_reg           = "on",
    parameter pmi_family              = "common",
    parameter pmi_implementation      = "LUT",
    parameter module_type             = "pmi_mac"
)
(
    input                            Clock,
    input                            ClkEn,
    input                            Aclr,
    input   [(pmi_dataa_width-1):0]  DataA,
    input   [(pmi_datab_width-1):0]  DataB,
    output  [(pmi_accum_width-1):0]  Result
)/*synthesis syn_black_box*/;
endmodule // pmi_mac
```

VHDL Definition for `pmi_mac`

```
component pmi_mac is
    generic (
        pmi_dataa_width         : integer := 18;
        pmi_datab_width         : integer := 18;
        pmi_accum_width         : integer := 18;
        pmi_sign                : string  := "on";
        pmi_additional_pipeline : integer := 1;
        pmi_add_sub             : string  := "add";
        pmi_input_reg           : string  := "on";
        pmi_family              : string  := "common";
        pmi_implementation      : string  := "LUT";
        module_type             : string  := "pmi_mac"
    );
    port (
        Clock  : in std_logic;
        ClkEn  : in std_logic;
        Aclr   : in std_logic;
        DataA  : in std_logic_vector((pmi_dataa_width-1) downto 0);
        DataB  : in std_logic_vector((pmi_datab_width-1) downto 0);
        Result : out std_logic_vector((pmi_accum_width-1) downto 0)
    );
end component pmi_mac;
```

4.5. pmi_multaddsub

The following are port descriptions, attribute descriptions, Verilog definition and VHDL definition for pmi_multaddsub.

Table 4.9. Port Definitions for pmi_multaddsub

Direction	Port Name	Type	Size (buses only)
I	DataA0	Bus	(pmi_dataa_width - 1):0
I	DataA1	Bus	(pmi_dataa_width - 1):0
I	DataB0	Bus	(pmi_datab_width - 1):0
I	DataB1	Bus	(pmi_datab_width - 1):0
I	Clock	Bit	N/A
I	ClkEn	Bit	N/A
I	Aclr	Bit	N/A
O	Result	Bus	(pmi_dataa_width + pmi_datab_width):0

Table 4.10. Attribute Definitions for pmi_multaddsub

Attributes	Description	Values	Default Value
pmi_dataa_width	Defines Bit width of DataA0 and DataA1 ports	2–36	9
pmi_datab_width	Defines Bit width of DataB0 and DataB1 ports	2–36	9
pmi_additional_pipeline	Specifies the number of pipeline stages. If pmi_implementation == LUT; N = 9 If pmi_implementation == DSP; N = 3	0–N	1
pmi_add_sub	Controls the operation to be performed on the accumulated value and multiplication product.	“add”, “sub”	“add”
pmi_input_reg	Enables input register	“on”, “off”	“on”
pmi_output_reg	Enables output register	“on”, “off”	“on”
pmi_sign	Controls the interpretation of the inputs whether a signed or an unsigned number.	“on”, "off"	"on"
pmi_family	Defines the FPGA family being used in the module	"common"	"common"
pmi_implementation	Selects the resource that will be used for implemention.	“DSP”, ”LUT”	“LUT”
module_type	Refers to the type of pmi module	“pmi_multaddsub”	“pmi_multaddsub”

Verilog Definition for `pmi_multaddsub`

```
module pmi_multaddsub
#(
    parameter pmi_dataa_width         = 9,
    parameter pmi_datab_width         = 9,
    parameter pmi_sign                = "on",
    parameter pmi_additional_pipeline = 1,
    parameter pmi_add_sub             = “add”,
    parameter pmi_input_reg           = "on",
    parameter pmi_output_reg          = "on",
    parameter pmi_family              = "common",
    parameter pmi_implementation      = "LUT",
    parameter module_type             = "pmi_multaddsub"
)
(
    input                              Clock,
    input                              ClkEn,
    input                              Aclr,
    input   [(pmi_dataa_width-1):0]    DataA0,
    input   [(pmi_dataa_width-1):0]    DataA1,
    input   [(pmi_datab_width-1):0]    DataB0,
    input   [(pmi_datab_width-1):0]    DataB1,
    output  [(pmi_dataa_width+pmi_datab_width):0]  Result
)/*synthesis syn_black_box*/;
endmodule // pmi_multaddsub
```

VHDL Definition for `pmi_multaddsub`

```
component pmi_multaddsub is
    generic (
        pmi_dataa_width         : integer := 9;
        pmi_datab_width         : integer := 9;
        pmi_sign                : string  := "on";
        pmi_additional_pipeline : integer := 1;
        pmi_add_sub             : string  := "add";
        pmi_input_reg           : string  := "on";
        pmi_output_reg          : string  := "on";
        pmi_family              : string  := "common";
        pmi_implementation      : string  := "LUT";
        module_type             : string  := "pmi_multaddsub"
    );
    port (
        Clock  : in std_logic;
        ClkEn  : in std_logic;
        Aclr   : in std_logic;
        DataA0 : in std_logic_vector((pmi_dataa_width-1) downto 0);
        DataA1 : in std_logic_vector((pmi_dataa_width-1) downto 0);
        DataB0 : in std_logic_vector((pmi_datab_width-1) downto 0);
        DataB1 : in std_logic_vector((pmi_datab_width-1) downto 0);
        Result : out std_logic_vector((pmi_dataa_width+pmi_datab_width) downto
0)
    );
end component pmi_multaddsub;
```

4.6. pmi_complex_mult

The following are port descriptions, attribute descriptions, Verilog definition and VHDL definition for pmi_complex_mult.

Table 4.11. Port Definitions for pmi_complex_mult

Direction	Port Name	Type	Size (buses only)
I	DataA_Re	Bus	(pmi_dataa_width - 1):0
I	DataA_Im	Bus	(pmi_dataa_width - 1):0
I	DataB_Re	Bus	(pmi_datab_width - 1):0
I	DataB_Im	Bus	(pmi_datab_width - 1):0
I	Clock	Bit	N/A
I	ClkEn	Bit	N/A
I	Aclr	Bit	N/A
O	Result_Re	Bus	(pmi_dataa_width + pmi_datab_width):0
O	Result_Im	Bus	(pmi_dataa_width + pmi_datab_width):0

Table 4.12. Attribute Definitions for pmi_complex_mult

Attributes	Description	Values	Default Value
pmi_dataa_width	Defines Bit width of DataA_Re and DataA_Im ports	2–36	8
pmi_datab_width	Defines Bit width of DataB_Re and DataB_Im ports	2–36	8
pmi_additional_pipeline	Specifies the number of pipeline stages. If pmi_implementation == LUT; N = 9 If pmi_implementation == DSP; N = 3	0–N	1
pmi_input_reg	Enables input register	“on”, “off”	“on”
pmi_output_reg	Enables output register	“on”, “off”	“on”
pmi_sign	Controls the interpretation of the inputs whether a signed or an unsigned number.	“on”, "off"	"on"
pmi_family	Defines the FPGA family being used in the module	"common"	"common"
pmi_mult_mode	Specifies the number of multiplier instances for implementation	3, 4	3
pmi_implementation	Selects the resource that will be used for implemention.	“DSP”, ”LUT”	“LUT”
module_type	Refers to the type of pmi module	“pmi_complex_mult”	“pmi_complex_mult”

Verilog Definition for `pmi_complex_mult`

```
module pmi_complex_mult
#(
    parameter pmi_dataa_width         = 8,
    parameter pmi_datab_width         = 8,
    parameter pmi_sign                = "on",
    parameter pmi_additional_pipeline = 1,
    parameter pmi_input_reg           = "on",
    parameter pmi_output_reg          = "on",
    parameter pmi_family              = "common",
    parameter pmi_mult_mode           = 3,
    parameter pmi_implementation      = "LUT",
    parameter module_type             = "pmi_complex_mult"
)
(
    input                             Clock,
    input                             ClkEn,
    input                             Aclr,
    input   [(pmi_dataa_width-1):0]   DataA0,
    input   [(pmi_dataa_width-1):0]   DataA1,
    input   [(pmi_datab_width-1):0]   DataB0,
    input   [(pmi_datab_width-1):0]   DataB1,
    output  [(pmi_dataa_width+pmi_datab_width):0]   Result
```

```
)/*synthesis syn_black_box*/;
endmodule // pmi_complex_mult
```

VHDL Definition for `pmi_complex_mult`

```
component pmi_complex_mult is
    generic (
        pmi_dataa_width         : integer := 9;
        pmi_datab_width         : integer := 9;
        pmi_sign                : string  := "on";
        pmi_additional_pipeline : integer := 1;
        pmi_input_reg           : string  := "on";
        pmi_output_reg          : string  := "on";
        pmi_family              : string  := "common";
        pmi_mult_mode           : integer := 1;
        pmi_implementation      : string  := "LUT";
        module_type             : string  := "pmi_complex_mult"
    );
    port (
        Clock  : in std_logic;
        ClkEn  : in std_logic;
        Aclr   : in std_logic;
        DataA0 : in std_logic_vector((pmi_dataa_width-1) downto 0);
        DataA1 : in std_logic_vector((pmi_dataa_width-1) downto 0);
        DataB0 : in std_logic_vector((pmi_datab_width-1) downto 0);
        DataB1 : in std_logic_vector((pmi_datab_width-1) downto 0);
        Result : out std_logic_vector((pmi_dataa_width+pmi_datab_width) downto
0)
    );
end component pmi_complex_mult;
```

4.7. pmi_dsp

The pmi_dsp is used to easily instantiate a MAC16 DSP block which can be configured as a multiplier, adder, subtractor, accumulator, multiply-adder or multiply-subtractor. This can only be used for iCE40UP devices. The following are port descriptions, attribute descriptions, Verilog definition and VHDL definition for pmi_dsp.

Table 4.13. Port Definitions for pmi_complex_mult

Direction	Port Name	Type	Size (buses only)
I	CLK	Bit	N/A
I	CE	Bit	N/A
I	C	Bus	16
I	A	Bus	16
I	B	Bus	16
I	D	Bus	16
I	AHOLD	Bit	N/A
I	BHOLD	Bit	N/A
I	CHOLD	Bit	N/A

Direction	Port Name	Type	Size (buses only)
I	DHOLD	Bit	N/A
I	IRSTTOP	Bit	N/A
I	IRSTBOT	Bit	N/A
I	ORSTTOP	Bit	N/A
I	ORSTBOT	Bit	N/A
I	OLOADTOP	Bit	N/A
I	OLOADBOT	Bit	N/A
I	ADDSUBTOP	Bit	N/A
I	ADDSUBBOT	Bit	N/A
I	OHOLDTOP	Bit	N/A
I	OHOLDBOT	Bit	N/A
I	CI	Bit	N/A
I	ACCUMCI	Bit	N/A
I	SIGNEXTIN	Bit	N/A
O	O	Bus	32
O	CO	Bit	N/A
O	ACCUMCO	Bit	N/A
O	SIGNEXTOUT	Bit	N/A

Table 4.14. Attribute Definitions for pmi_complex_mult

Attributes	Description	Values	Default Value
NEG_TRIGGER	Controls input clock polarity	"0b0", “0b1”	"0b0"
A_REG	Enables Input A register	"0b0", “0b1”	"0b0"
B_REG	Enables Input B register	"0b0", “0b1”	"0b0"
C_REG	Enables Input C register	"0b0", “0b1”	"0b0"
D_REG	Enables Input D register	"0b0", “0b1”	"0b0"
TOP_8x8_MULT_REG	Enables output register of top 8x8 multiplier	"0b0", “0b1”	"0b0"
BOT_8x8_MULT_REG	Enables output register of bottom 8x8 multiplier	"0b0", “0b1”	"0b0"
PIPELINE_16x16_MULT_REG1	Enables intermediate register of 16x16 multiplier	"0b0", “0b1”	"0b0"

Attributes	Description	Values	Default Value
PIPELINE_16x16_MULT_REG2	Enables pipeline register of 16x16 multiplier	"0b0", “0b1”	"0b0"
TOPOUTPUT_SELECT	Selects top output O[31:16] 0b00 = top accumulator 0b01 = top accumulator (registered) 0b10 = top mult8x8 0b11 = mult16x16	"0b00"	"0b00"
TOPADDSUB_LOWERINPUT	Selects lower input for the upper adder/subtractor, 0b00 = input A 0b01 = top mult8x8 0b10 = mult16x16 0b11 = sign extend from lower adder/subtractor	"0b00"	"0b00"
TOPADDSUB_UPPERINPUT	Selects upper input for the upper adder/subtractor 0b0 = accumulate 0b1 = input C	"0b0", “0b1”	"0b0"
TOPADDSUB_CARRYSELECT	Selects Carry/borrow input to upper adder/subtractor 0b00 = constant 0 0b01 = constant 1 0b10 = carry from lower adder/subtractor 0b11 = carry from lower adder/subtractor	"0b00"	"0b00"
BOTOUTPUT_SELECT	Selects lower output O[15:0] 0b00 = bottom adder/subtractor 0b01 = bottom adder/subtractor (registered), 0b10 = bottom mult8x8 0b11 = mult16x16	"0b00"	"0b00"
BOTADDSUB_LOWERINPUT	Selects lower input for the lower adder/subtractor 0b00 = input B 0b01 = bottom mult8x8 0b10 = mult16x16 0b11 = sign extend from previous MAC16	"0b00"	"0b00"
BOTADDSUB_UPPERINPUT	Selects upper input for the lower adder/subtractor 0b0 = accumulate 0b1 = input C	"0b0", “0b1”	"0b0"
BOTADDSUB_CARRYSELECT	Selects Carry/borrow input to lower adder/subtractor 0b00 = constant 0 0b01 = constant 1 0b10 = ACCUMCI 0b11 = CI	"0b00"	"0b00"
MODE_8x8	Selects 8x8 Multiplier mode and 8x8 Low-Power Multiplier Blocking Option	"0b0", “0b1”	"0b0"

Attributes	Description	Values	Default Value
A_SIGNED	Indicates whether multiplier input A is signed or unsigned	"0b0", “0b1”	"0b0"
B_SIGNED	Indicates whether multiplier input B is signed or unsigned	"0b0", “0b1”	"0b0"

Verilog Definition for `pmi_dsp`

```
module pmi_dsp
#(
    parameter NEG_TRIGGER              = "0b0";
    parameter A_REG                    = "0b0";
    parameter B_REG                    = "0b0";
    parameter C_REG                    = "0b0";
    parameter D_REG                    = "0b0";
    parameter TOP_8x8_MULT_REG         = "0b0";
    parameter BOT_8x8_MULT_REG         = "0b0";
    parameter PIPELINE_16x16_MULT_REG1 = "0b0";
    parameter PIPELINE_16x16_MULT_REG2 = "0b0";
    parameter TOPOUTPUT_SELECT         = "0b00";
    parameter TOPADDSUB_LOWERINPUT     = "0b00";
    parameter TOPADDSUB_UPPERINPUT     = "0b0";
    parameter TOPADDSUB_CARRYSELECT    = "0b00";
    parameter BOTOUTPUT_SELECT         = "0b00";
    parameter BOTADDSUB_LOWERINPUT     = "0b00";
    parameter BOTADDSUB_UPPERINPUT     = "0b0";
    parameter BOTADDSUB_CARRYSELECT    = "0b00";
    parameter MODE_8x8                 = "0b0";
    parameter A_SIGNED                 = "0b0";
    parameter B_SIGNED                 = "0b0";
)
(
    input         CLK,
    input         CE,
    input  [15:0] C,
    input  [15:0] A,
    input  [15:0] B,
    input  [15:0] D,
    input         AHOLD,
    input         BHOLD,
    input         CHOLD,
    input         DHOLD,
    input         IRSTTOP,
    input         IRSTBOT,
    input         ORSTTOP,
    input         ORSTBOT,
    input         OLOADTOP,
    input         OLOADBOT,
    input         ADDSUBTOP,
    input         ADDSUBBOT,
    input         OHOLDTOP,
```

```
    input          OHOLDBOT,
    input          CI,
    input          ACCUMCI,
    input          SIGNEXTIN,

    output [31:0] O,
    output         CO,
    output         ACCUMCO,
    output         SIGNEXTOUT
);
endmodule // pmi_dsp
```

VHDL Definition for `pmi_dsp`

```
component pmi_dsp is
    generic (
        NEG_TRIGGER              : string := "0b0";
        A_REG                    : string := "0b0";
        B_REG                    : string := "0b0";
        C_REG                    : string := "0b0";
        D_REG                    : string := "0b0";
        TOP_8x8_MULT_REG         : string := "0b0";
        BOT_8x8_MULT_REG         : string := "0b0";
        PIPELINE_16x16_MULT_REG1 : string := "0b0";
        PIPELINE_16x16_MULT_REG2 : string := "0b0";
        TOPOUTPUT_SELECT         : string := "0b00";
        TOPADDSUB_LOWERINPUT     : string := "0b00";
        TOPADDSUB_UPPERINPUT     : string := "0b0";
        TOPADDSUB_CARRYSELECT    : string := "0b00";
        BOTOUTPUT_SELECT         : string := "0b00";
        BOTADDSUB_LOWERINPUT     : string := "0b00";
        BOTADDSUB_UPPERINPUT     : string := "0b0";
        BOTADDSUB_CARRYSELECT    : string := "0b00";
        MODE_8x8                 : string := "0b0";
        A_SIGNED                 : string := "0b0";
        B_SIGNED                 : string := "0b0"
    );
    port (
        CLK        : in  std_logic;
        CE         : in  std_logic;
        C          : in  std_logic_vector(15 downto 0);
        A          : in  std_logic_vector(15 downto 0);
        B          : in  std_logic_vector(15 downto 0);
        D          : in  std_logic_vector(15 downto 0);
        AHOLD      : in  std_logic;
        BHOLD      : in  std_logic;
        CHOLD      : in  std_logic;
        DHOLD      : in  std_logic;
        IRSTTOP    : in  std_logic;
        IRSTBOT    : in  std_logic;
        ORSTTOP    : in  std_logic;
        ORSTBOT    : in  std_logic;
        OLOADTOP   : in  std_logic;
```

```
        OLOADBOT   : in  std_logic;
        ADDSUBTOP  : in  std_logic;
        ADDSUBBOT  : in  std_logic;
        OHOLDTOP   : in  std_logic;
        OHOLDBOT   : in  std_logic;
        CI         : in  std_logic;
        ACCUMCI    : in  std_logic;
        SIGNEXTIN  : in  std_logic;
        O          : out std_logic_vector(31 downto 0);
        CO         : out std_logic;
        ACCUMCO    : out std_logic;
        SIGNEXTOUT : out std_logic
    );
end component pmi_dsp;
```

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Date	Document Version	IP Core Version	Change Summary
February 2018	1.0	1.0	Initial release.

7th Floor, 111 SW 5th Avenue
Portland, OR 97204, USA
T 503.268.8000
www.latticesemi.com