Arithmetic Modules - Lattice Radiant Software

User Guide

FPGA-IPUG-02032-2.5

November 2023

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Contents....3
Figures....5
Tables....6
Acronyms in This Document....8
1. Introduction....9
2. Arithmetic Modules....10
2.1. Adder....10
2.1.1. Ports....10
2.1.2. Attributes....12
2.1.3. Timing Diagram....13
2.2. Subtractor....13
2.2.1. Ports....14
2.2.2. Attributes....15
2.2.3. Timing Diagram....16
2.3. Adder-Subtractor....17
2.3.1. Ports....17
2.3.2. Attributes....19
2.3.3. Timing Diagram....21
2.4. Comparator....22
2.4.1. Ports....22
2.4.2. Attributes....23
2.4.3. Timing Diagram....25
2.5. Counter....26
2.5.1. Ports....26
2.5.2. Attributes....27
2.5.3. Timing Diagram....28
2.6. Multiplier....29
2.6.1. Ports....29
2.6.2. Attributes....29
2.6.3. Timing Diagram....31
2.7. Multiply_Accumulate....31
2.7.1. Ports....32
2.7.2. Attributes....32
2.7.3. Timing Diagram....33
2.8. Multiply_Add_Subtract....34
2.8.1. Ports....34
2.8.2. Attributes....35
2.8.3. Timing Diagram....36
2.9. Multiply-Add-Subtract-Sum....36
2.9.1. Ports....37
2.9.2. Attributes....38
2.9.3. Timing Diagram....39
2.10. Complex_Multiplier....40
2.10.1. Ports....40
2.10.2. Attributes....41
2.10.3. Timing Diagram....42
2.11. LFSR....42
2.11.1. Ports....43
2.11.2. Attributes....44
2.11.3. Timing Diagram....45
2.12. Convert....45
2.12.1. Ports....45

2.12.2. Attributes....46
2.12.3. Timing Diagram....47
2.13. Sin-Cos Table....48
2.13.1. Ports....48
2.13.2. Attributes....49
3. IP Generation....51
3.1. Generating a 32-bit Multiplier Module....52
4. Simulation Flow....53
4.1. Generating a Multiplier on Default Settings....53
5. Constraining the IP....56
6. PMI Support....57
6.1. pmi_add....57
6.2. pmi_sub....58
6.3. pmi_addsub....59
6.4. pmi_counter....61
6.5. pmi_mult....62
6.6. pmi_mac....64
6.7. pmi_multaddsub....66
6.8. pmi_multaddsubsum....68
6.9. pmi_complex_mult....71
6.10. pmi_dsp....73
Appendix A. Resource Utilization....78
A.1. Adder....78
A.2. Subtractor....78
A.3. Adder-Subtractor....79
A.4. Comparator....80
A.5. Counter....81
A.6. Multiplier....81
A.7. Multiply Accumulate....82
A.8. Mult-Add-Sub....82
A.9. Mult-Add-Sub-Sum....83
A.10. Complex Multiplier....84
A.11. LFSR....84
A.12. Convert....85
A.13. Sin-Cos Table....85
References....87
Technical Support Assistance....88
Revision History....89

Figures

Figure 2.1. Adder Schematic Diagram 10
Figure 2.2. Adder Timing Diagram 13
Figure 2.3. Subtractor Schematic Diagram 13
Figure 2.4. Subtractor Timing Diagram 16
Figure 2.5. Adder-Subtractor Schematic Diagram 17
Figure 2.6. Adder-Subtractor Timing Diagram 21
Figure 2.7. Comparator Schematic Diagram 22
Figure 2.8. Comparator Timing Diagram 25
Figure 2.9. Counter Schematic Diagram 26
Figure 2.10. Counter Timing Diagram 28
Figure 2.11. Multiplier Schematic Diagram 29
Figure 2.12. Multiplier Timing Diagram 31
Figure 2.13. Multiply_Accumulate Schematic 31
Figure 2.14. Multiply_Accumulate Timing Waveforms 33
Figure 2.15. Multiply_Add_Subtract Schematic 34
Figure 2.16. Multiply_Add_Subtract Timing Diagram 36
Figure 2.17. Multiply_Add_Subtract_Sum Schematic Diagram 36
Figure 2.18. Multiply_Add_Subtract_Sum Timing Diagram 39
Figure 2.19. Complex_Multiplier Schematic Diagram 40
Figure 2.20. Complex_Multiplier Timing Diagram 42
Figure 2.21. Fibonacci LFSR Illustrative Shifter Implementation Schematic 42
Figure 2.22. Galois LFSR Illustrative Shifter Implementation Schematic 42
Figure 2.23. LFSR Timing Diagram 45
Figure 2.24 Convert Schematic Diagram 45
Figure 2.25. Convert Timing Diagram 47
Figure 2.26. Sin-Cos Table Schematic Diagram 48
Figure 3.1. Arithmetic Modules under Module/IP on Local in the Lattice Radiant Software 51
Figure 3.2. Example: Generating 32-bit Multiplier Using Module/IP Block Wizard 52
Figure 3.3. Example: Generating 32-bit Multiplier in IP Configuration 52
Figure 4.1. Project with an Instance of Multiplier 53
Figure 4.2. Simulation Wizard 54
Figure 4.3. Final Simulation Files 54
Figure 4.4. File Parsing and Top Module Selection 55
Figure 4.5. Modelsim Initial Simulation 55

Tables

Table 2.1. Adder Ports 10
Table 2.2. Adder Attributes 12
Table 2.3. Subtractor Ports 14
Table 2.4. Subtractor Attributes 15
Table 2.5. Adder-Subtractor Ports 17
Table 2.6. Adder-Subtractor Attributes 19
Table 2.7. Comparator Ports 22
Table 2.8. Comparator Attributes 23
Table 2.9. Counter Ports 26
Table 2.10. Counter Attributes 27
Table 2.11. Multiplier Ports 29
Table 2.12. Multiplier Attributes 29
Table 2.13. Multiply_Accumulate Ports 32
Table 2.14. Multiply_Accumulate Attributes 32
Table 2.15. Multiply_Add_Subtract Ports 34
Table 2.16. Multiply_Add_Subtract Attributes 35
Table 2.17. Multiply_Add_Subtract_Sum Ports 37
Table 2.18. Multiply_Add_Subtract_Sum Attributes 38
Table 2.19. Complex_Multiplier Ports 40
Table 2.20. Complex_Multiplier Attributes 41
Table 2.21. LFSR Ports 43
Table 2.22. LFSR Attributes 44
Table 2.23. Convert Ports 45
Table 2.24. Convert Attributes 46
Table 2.25. Sin-Cos Table Ports 48
Table 2.26. Sin-Cos Table Attributes 49
Table 4.1. File List 53
Table 6.1. Port Definitions for pmi_add 57
Table 6.2. Attribute Definitions for pmi_add 57
Table 6.3. Port Definitions for pmi_sub 58
Table 6.4. Attribute Definitions for pmi_sub 58
Table 6.5. Port Definitions for pmi_addsub 59
Table 6.6. Attribute Definitions for pmi_addsub 60
Table 6.7. Port Definitions for pmi_counter 61
Table 6.8. Attribute Definitions for pmi_counter 61
Table 6.9. Port Definitions for pmi_mult 62
Table 6.10. Attribute Definitions for pmi_mult 63
Table 6.11. Port Definitions for pmi_mac 64
Table 6.12. Attribute Definitions for pmi_mac 65
Table 6.13. Port Definitions for pmi_multaddsub 66
Table 6.14. Attribute Definitions for pmi_multaddsub 67
Table 6.15. Port Definitions for pmi_multaddsubsum 68
Table 6.16. Attribute Definitions for pmi_multaddsubsum 69
Table 6.17. Port Definitions for pmi_complex_mult 71
Table 6.18. Attribute Definitions for pmi_complex_mult 71
Table 6.19. Port Definitions for pmi_dsp 73
Table 6.20. Attribute Definitions for pmi_dsp 74
Table A.1. Adder Resource Utilization (LIFCL) 78
Table A.2. Adder Resource Utilization (LFCPNX) 78
Table A.3. Adder Resource Utilization (LAV-AT) 78
Table A.4. Subtractor Resource Utilization (LIFCL) 78
Table A.5. Subtractor Resource Utilization (LFCPNX) 79

Table A.6. Subtractor Resource Utilization (LAV-AT) 79
Table A.7. Adder-Subtractor Resource Utilization (LIFCL) 79
Table A.8. Adder-Subtractor Resource Utilization (LFCPNX) 79
Table A.9. Adder - Subtractor Resource Utilization (LAV-AT) 80
Table A.10. Comparator Resource Utilization (LIFCL) 80
Table A.11. Comparator Resource Utilization (LFCPNX) 80
Table A.12. Comparator Resource Utilization (LAV-AT) 80
Table A.13. Counter Resource Utilization (LIFCL) 81
Table A.14. Counter Resource Utilization (LFCPNX) 81
Table A.15. Counter Resource Utilization (LAV-AT) 81
Table A.16. Multiplier Resource Utilization (LIFCL) 81
Table A.17. Multiplier Resource Utilization (LFCPNX) 81
Table A.18. Multiplier Resource Utilization (LAV-AT) 82
Table A.19. Multipliy Accumulate Resource Utilization (LIFCL) 82
Table A.20. Multipliy Accumulate Resource Utilization (LFCPNX) 82
Table A.21. Multiply Accumulate Resource Utilization (LAV-AT) 82
Table A.22. Mult Add Sub Resource Utilization (LIFCL) 82
Table A.23. Mult Add Sub Resource Utilization (LFCPNX) 83
Table A.24. Mult Add Sub Resource Utilization (LAV-AT) 83
Table A.25. Mult Add Sub Sum Resource Utilization (LIFCL) 83
Table A.26. Mult Add Sub Sum Resource Utilization (LFCPNX) 83
Table A.27. Mult Add Sub Sum Resource Utilization (LAV-AT) 83
Table A.28. Complex Multiplier Resource Utilization (LIFCL) 84
Table A.29. Complex Multiplier Resource Utilization (LFCPNX) 84
Table A.30. Complex Multiplier Resource Utilization (LAV-AT) 84
Table A.31. LFSR Resource Utilization (LIFCL) 84
Table A.32. LFSR Resource Utilization (LFCPNX) 84
Table A.33. LFSR Resource Utilization (LAV-AT) 85
Table A.34. Convert Resource Utilization (LIFCL) 85
Table A.35. Convert Resource Utilization (LFCPNX) 85
Table A.36. Convert Resource Utilization (LAV-AT) 85
Table A.37. Sin-Cos Table Resource Utilization (LIFCL) 85
Table A.38. Sin-Cos Table Resource Utilization (LFCPNX) 85
Table A.39. Sin-Cos Table Resource Utilization (LAV-AT) 86

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
FPGA	Field-Programmable Gate Array
IP	Intellectual Property
LUT	Lookup-Table
PMI	Parameterized Module Instantiation
RTL	Register Transfer Level

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, as shown in Figure 2.1, are optional and are removed/ignored in certain configurations.

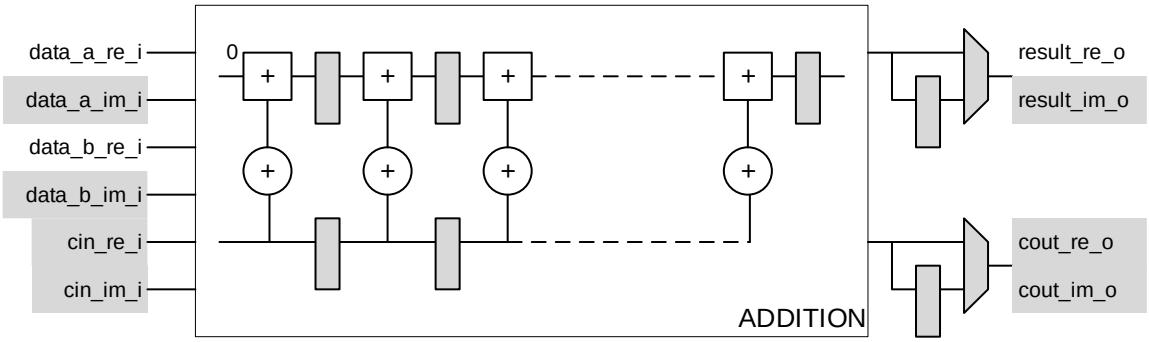

Figure 2.1. Adder Schematic Diagram

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 – Retains the previous state 1'b1 – Toggles 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 with 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
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

Signal Name	Direction	Width (bits)	Description
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	1	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	1	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	1	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, that is, a value outside the range of the configured number system.
cout_im_o	OUT	1	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, that is, 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	2–64	16	This configuration is the data width of the input/output ports.
SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the inputs whether a signed or an unsigned number.
USE_CNUM	on, off	off	This configuration controls the interpretation of the numbers being involved in the addition. off – 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 I/O are ignored. on – Complex numbers. The logic is implemented on the basis of the I/O as complex numbers.
USE_CIN	on, off	off	This configuration controls the usage of Carry-In in the addition. off – Ignore Carry-In. cin_re_i and cin_im_i inputs are unused. on – Consider Carry-In. cin_re_i and cin_im _i inputs are used.
USE_COUT	None, Overflow	None	This configuration controls the implementation of Carry-Out in the addition. None – Ignore Carry-Out. cout_re_o and cout_im_o outputs are always driven LOW. Overflow – Consider Carry-Out. cout_re_o and cout_im_o outputs are driven with carry.
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–N	0	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 I/O Width configuration. For example: 1-8 bits – 0 9-16 bits – 1 17-24 bits – 2 25-32 bits – 3 and so on This parameter is editable if D_WIDTH > 8

2.1.3. Timing Diagram

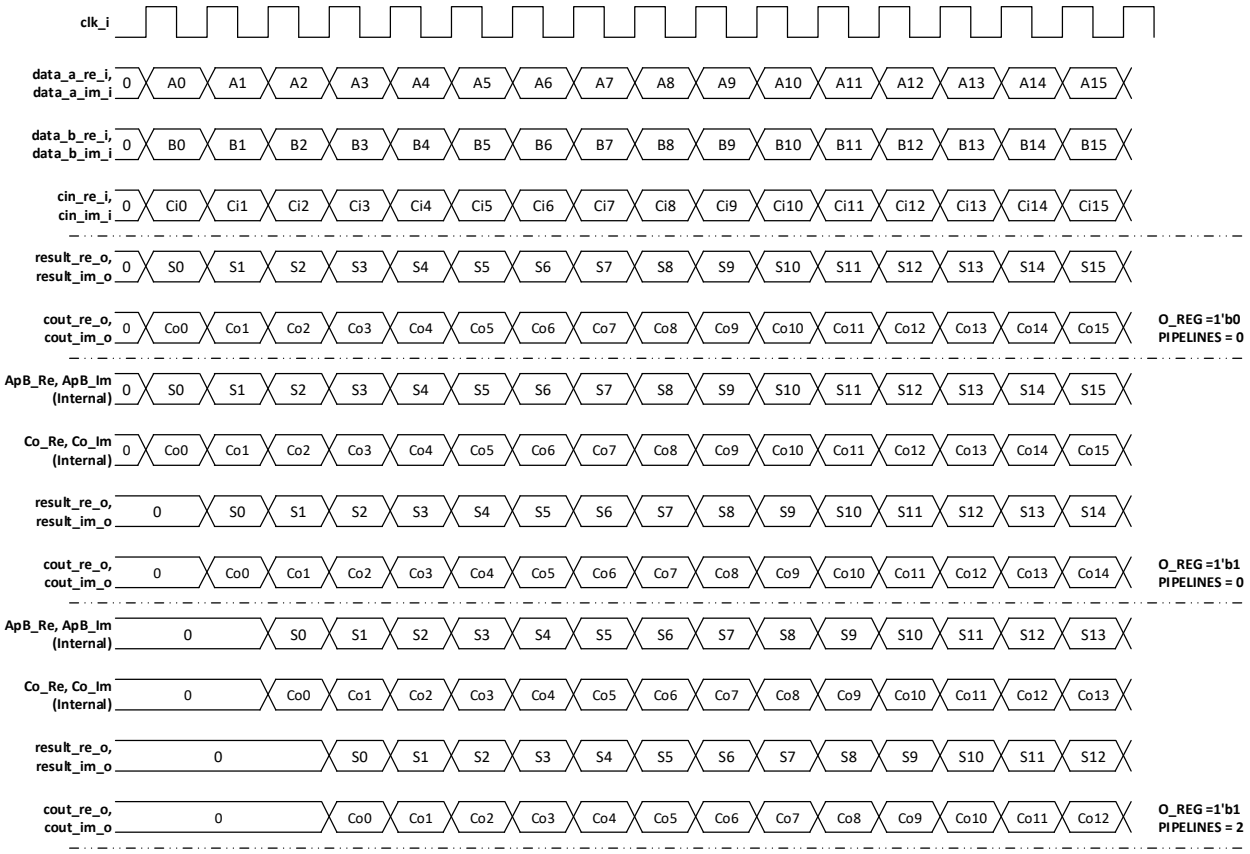

Figure 2.2. Adder Timing Diagram

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, as shown in Figure 2.3, are optional and are removed/ignored in certain configurations.

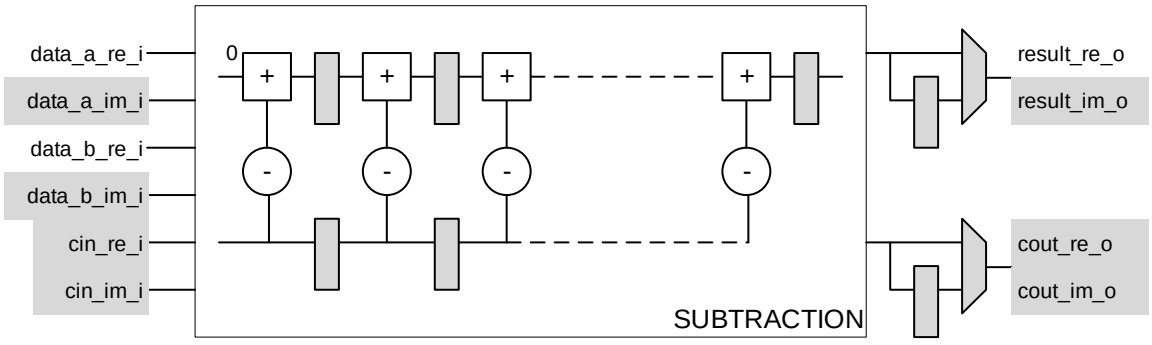

Figure 2.3. Subtractor Schematic Diagram

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 – Retains the previous state 1'b1 – Toggles 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 with 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
cin_re_i	IN	1	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	1	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}

Signal Name	Direction	Width (bits)	Description
cout_re_o	OUT	1	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 Overflow/Underflow This signal when HIGH indicates an Overflow/Underflow in the difference, that is, a value outside the range of the configured number system.
cout_im_o	OUT	1	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 Overflow/Underflow This signal when HIGH indicates an Overflow/Underflow in the difference, that is, a value outside the range of the configured number system.

2.2.2. Attributes

Table 2.4. Subtractor Attributes

Configuration	Range	Default Value	Description
D_WIDTH	2–64	16	This configuration is the data width of the input/output ports.
SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the inputs whether a signed or an unsigned number.
USE_CNUM	on, off	off	This configuration controls the interpretation of the numbers being involved in the subtraction. off – 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 I/O are ignored. on – Complex numbers. The logic is implemented on the basis of the I/O as complex numbers.
USE_CIN	on, off	off	This configuration controls the usage of Borrow-In in the subtraction. off – Ignore Borrow-In. cin_re_i and cin_im_i inputs are unused. on – Consider Borrow-In. cin_re_i and cin_im _i inputs are used.
USE_COUT	None, Overflow	None	This configuration controls the implementation of Borrow-Out from the subtraction. None – Ignore Borrow-Out. cout_re_o and cout_im_o outputs are always driven LOW. Overflow – Consider Borrow-Out. cout_re_o and cout_im_o outputs are driven with borrow.
USE_OREG	on, off	off	This configuration controls the registering of the subtractor 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 subtractor is high.

Configuration	Range	Default Value	Description
PIPELINES	0–N	0	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 I/O Width configuration. For example: 1-8 bits – 0 9-16 bits – 1 17-24 bits – 2 25-32 bits – 3 and so on This parameter is editable if D_WIDTH > 8

2.2.3. Timing Diagram

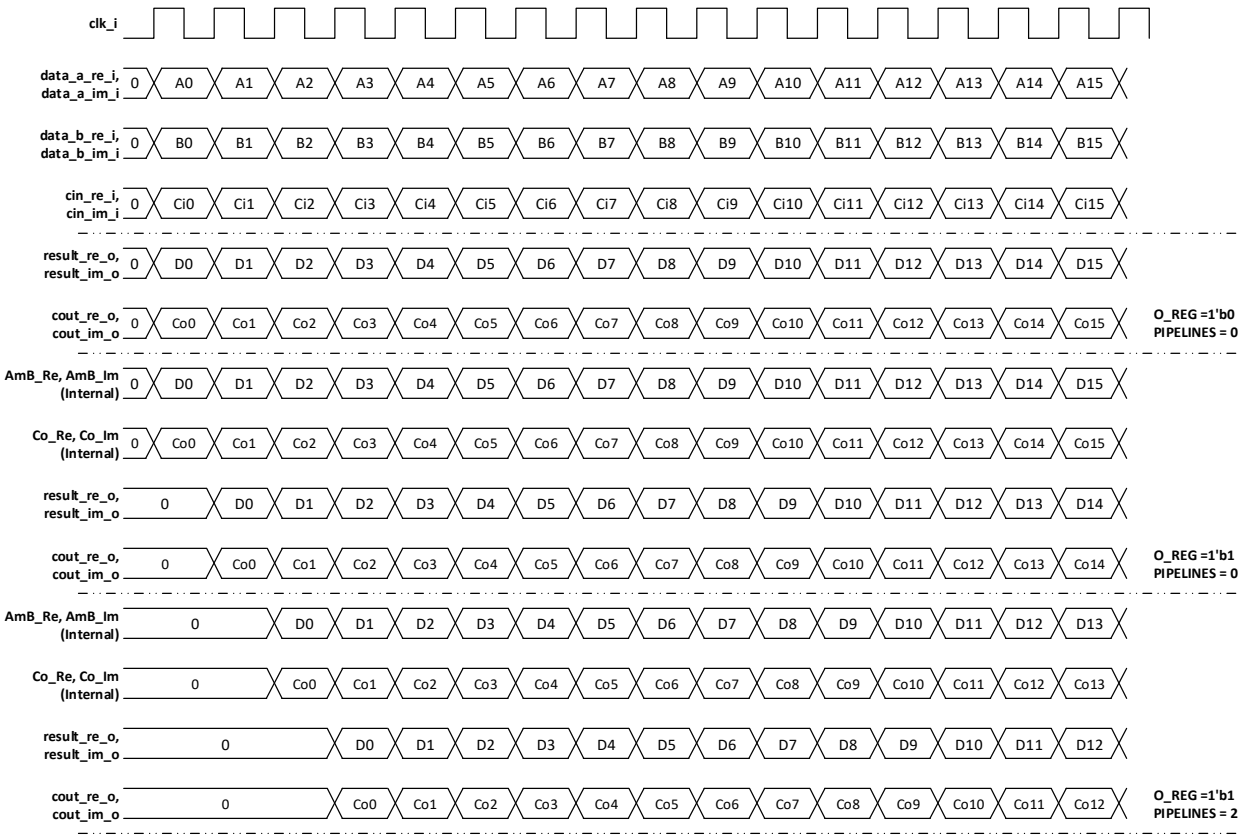

Figure 2.4. Subtractor Timing Diagram

2.3. Adder-Subtractor

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

Shaded portions of the Adder-Subtractor Schematic Diagram, shown in Figure 2.5, are optional and are removed/ignored in certain configurations.

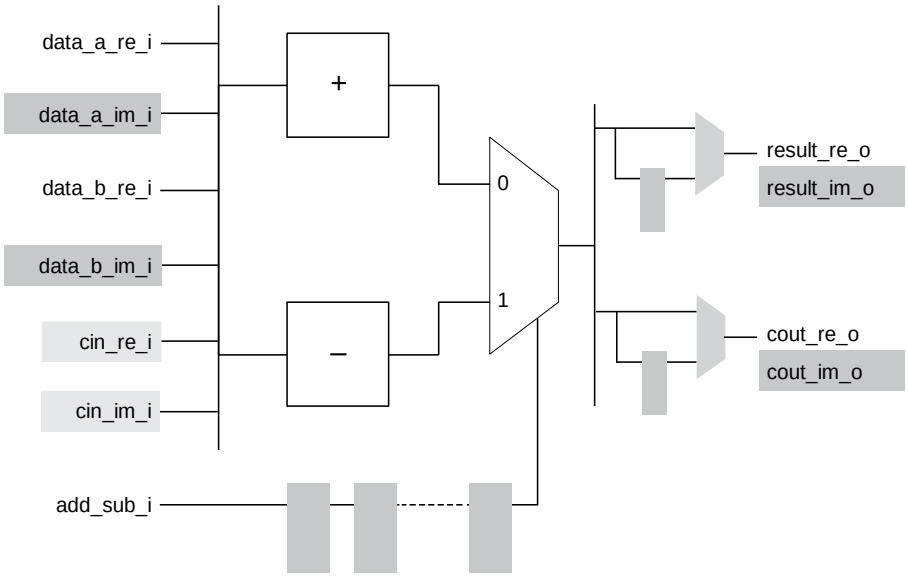

Figure 2.5. Adder-Subtractor Schematic Diagram

2.3.1. Ports

Table 2.5. Adder-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 – Retains the previous state 1'b1 – Toggles 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 with the clock during integration.
add_sub_i	IN	1	This signal controls the operation to be performed on the inputs. 1'b0 – Subtraction 1'b1 – Addition This signal can toggle dynamically per cycle and the result of the operation is dependent on the value for that input cycle.
data_a_re_i	IN	I_WDT	This signal contains a simple number or the real part of a complex number, which is the augend/minuend in the addition/subtraction respectively. 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	I_WDT	This signal contains the imaginary part of a complex number, which is the augend/minuend in the addition/subtraction respectively. USE_CNUM == 1'b0 – Tied to a constant in the integration USE_CNUM == 1'b1 – Used in the addition/subtraction
data_b_re_i	IN	I_WDT	This signal contains a simple number or the real part of a complex number, which is the addend/subtrahend in the addition/subtraction respectively. USE_CNUM == 1'b0 – Simple number USE_CNUM == 1'b1 – Real part of a complex number
data_b_im_i	IN	I_WDT	This signal contains the imaginary part of a complex number, which is the addend/subtrahend in the addition/subtraction respectively. USE_CNUM == 1'b0 – Tied to a constant in the integration USE_CNUM == 1'b1 – Used in the addition/subtraction
cin_re_i	IN	1	This signal contains a simple number or the real part of a complex number, which is the Carry/Borrow-In in the addition/subtraction respectively. USE_CIN == 1'b0 – Carry/Borrow-In is unused in the addition/subtraction. Tied to a constant in the integration. USE_CIN == 1'b1 – Carry-In/Borrow-In is used in the addition/subtraction. add_sub_i == 1'b1 – Treated as Active HIGH add_sub_i == 1'b0 – Implemented as below SIGNED == on – Treated as Active HIGH SIGNED == off – Treated as Active LOW
cin_im_i	IN	1	This signal contains the imaginary part of a complex number, which is the Carry/Borrow-In in the addition/subtraction respectively. USE_CIN or USE_CNUM == 1'b0 – Carry/Borrow-In is unused in the addition/subtraction. Tied to a constant in the integration. USE_CIN and USE_CNUM == 1'b1 – Carry/Borrow-In is used in the addition/subtraction. add_sub_i == 1'b1 – Treated as Active HIGH add_sub_i == 1'b0 – Implemented as below SIGNED == on – Treated as Active HIGH SIGNED == off – Treated as Active LOW
result_re_o	OUT	I_WDT	This signal carries a simple number or the real part of a complex number from the sum/difference of the addition/subtraction respectively. Core functionality is represented below. add_sub_i ? { data_a_re_i + data_b_re_i + cin_re_i } : { data_a_re_i - data_b_re_i - cin_re_i }
result_im_o	OUT	I_WDT	This signal carries the imaginary part of a complex number from the sum/difference of the addition/subtraction respectively. USE_CNUM == 1'b0 – Unconnected in the integration USE_CNUM == 1'b1 – Core functionality is represented below. add_sub_i? { data_a_im_i + data_b_im_i + cin_im_i } : { data_a_im_i - data_b_im_i - cin_im_i }

Signal Name	Direction	Width (bits)	Description
cout_re_o	OUT	1	This signal carries a simple number or the real part of a complex number from the Carry/Borrow-Out of the addition/subtraction respectively. USE_COUT == 1'b0 – Unconnected in the integration USE_COUT == 1'b1 – Carry/Borrow-Out is implemented. This signal when HIGH indicates an overflow/underflow in the sum/difference respectively, that is, a value outside the range of the configured number system. add_sub_i == 1'b1 – Treated as Active HIGH add_sub_i == 1'b0 – Implemented as below: SIGNED == on – Treated as Active HIGH SIGNED == off – Treated as Active LOW
cout_im_o	OUT	1	This signal carries the imaginary part of a complex number from the Carry/Borrow-Out of the addition/subtraction respectively. USE_COUT or USE_CNUM == 1'b0 – Unconnected in the integration USE_COUT and USE_CNUM == 1'b1 – Carry/Borrow-Out is implemented. This signal when HIGH indicates an overflow/underflow in the sum/difference, that is, a value outside the range of the configured number system. add_sub_i == 1'b1 – Treated as Active HIGH add_sub_i == 1'b0 – Implemented as below SIGNED == on – Treated as Active HIGH SIGNED == off – Treated as Active LOW

2.3.2. Attributes

Table 2.6. Adder-Subtractor Attributes

Configuration	Range	Default Value	Description
D_WIDTH	2-64	16	This configuration controls the width of the I/O, data_a_re_i, data_a_im_i, data_b_re_i, data_b_im_i, result_re_o and result_im_o. This configuration accepts integers only and any value less than 1 is automatically corrected to 1 by the design.
SIGNED	Signed or Unsigned	Unsigned	This configuration controls the interpretation of the inputs, data_a_re_i, data_a_im_i, data_b_re_i, data_b_im_i.
USE_CNUM	on or off	off	This configuration controls the interpretation of the numbers being involved in the operation. off – Simple numbers, that is, data_a_re_i, data_b_re_i, cin_re_i, result_re_o and cout_re_o are treated as simple numbers only while data_a_im_i, data_b_im_i, cin_im_i, result_im_o and cout_im_o I/O are ignored. on – Complex numbers, that is, logic is implemented on the basis of the I/O as complex numbers.
USE_CIN	on or off	off	This configuration controls the usage of Carry/Borrow-In in the operation. off – Ignore Carry/Borrow-In, that is, cin_re_i and cin_im_i inputs are unused. on – Consider Carry/Borrow-In, that is, cin_re_i and cin_im_i inputs are used.

Configuration	Range	Default Value	Description
USE_COUT	None or Overflow	None	This configuration controls the implementation of Carry/Borrow-Out from the addition/subtraction respectively. None – Ignore Carry/Borrow-Out, that is, cout_re_o and cout_im_o outputs are always driven LOW Overflow – Consider Carry/Borrow-Out, that is, cout_re_o and cout_im_o outputs are driven with carry.
USE_OREG	on or off	off	This configuration controls the registering of the operation result onto the outputs, result_re_o, result_im_o, cout_re_o, and cout_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 operation is high.
PIPELINES	0-N	0	This configuration controls the insertion of pipeline stages into the addition/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 I/O Width configuration. For example: 1-8 bits – 0 9-16 bits – 1 17-24 bits – 2 25-32 bits – 3 and so on This parameter is editable if D_WIDTH > 8
Internal Parameters			
I_WDT	—	—	I_WDT = (I_WIDTH < 1) ? 1 : I_WIDTH

2.3.3. Timing Diagram

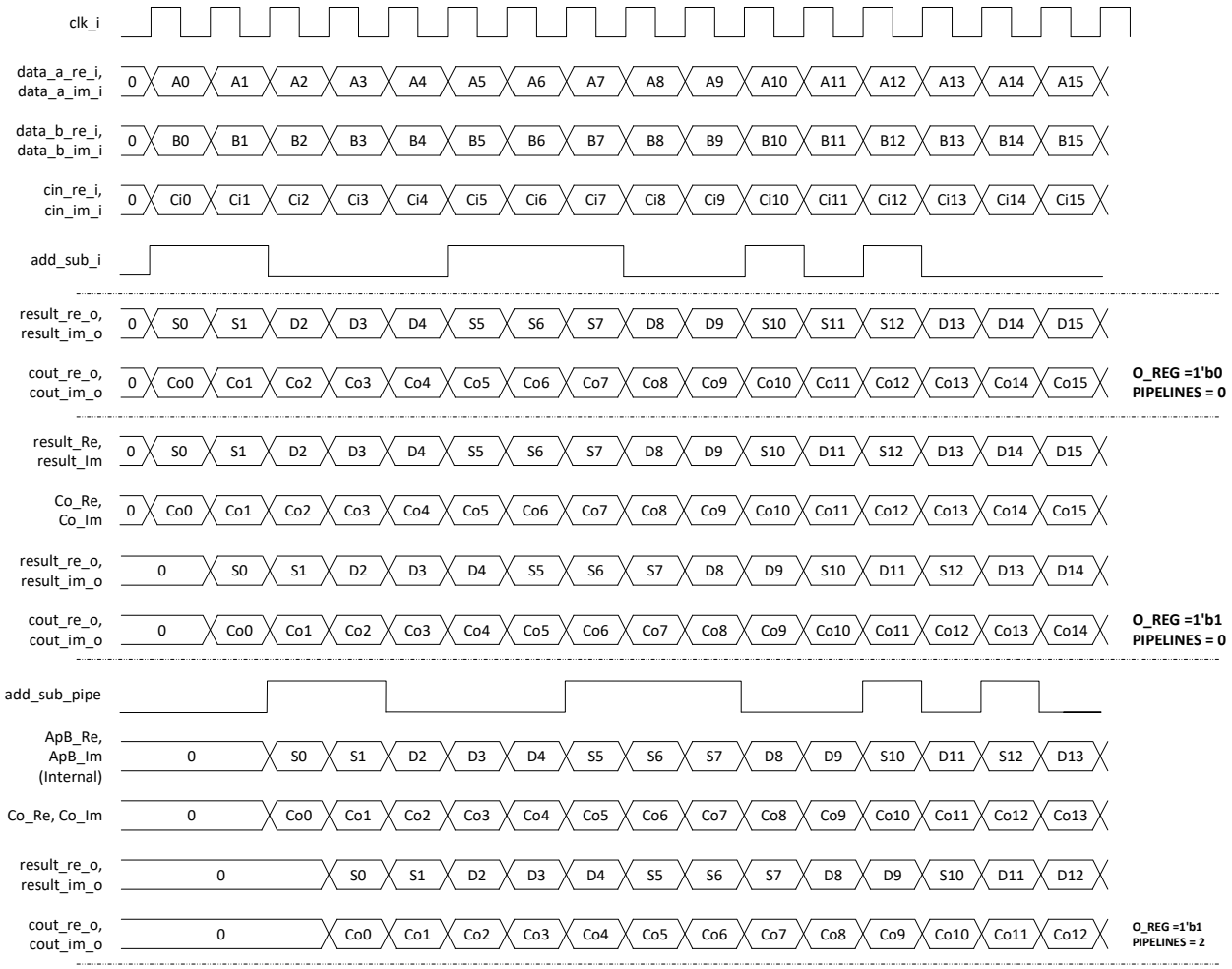

Figure 2.6. Adder-Subtractor Timing Diagram

2.4. Comparator

A two-input comparator that performs signed/unsigned comparison of the value represented by input data_a_i versus value represented by input data_b_i. The output result carries the comparison in terms of equals to, not equals to, less than, less than equals to, greater than and greater than equals to.

Shaded portions of the Comparator Schematic Diagram, as shown in Figure 2.7, are optional and are removed/ignored in certain configurations.

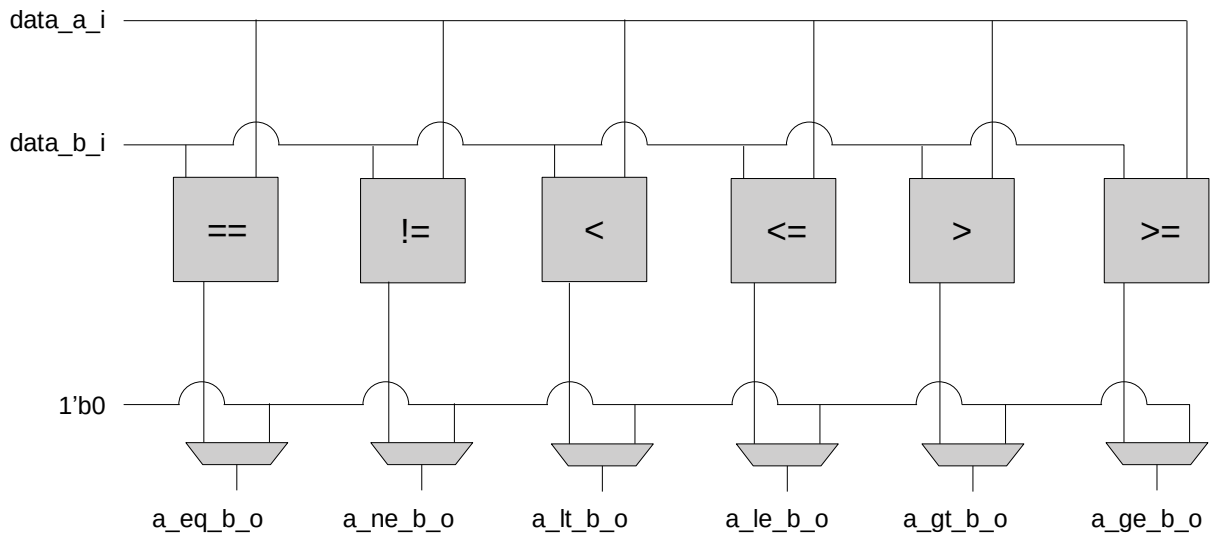

Figure 2.7. Comparator Schematic Diagram

2.4.1. Ports

Table 2.7. Comparator 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 - Retains the previous state 1'b1 - Toggles on the rising edge of the clock as per the logic
aclr_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 with the clock during integration.
data_a_i	IN	I_WDT	This signal contains one of the inputs for comparison.
data_b_i	IN	I_WDT	This signal contains one of the inputs for comparison.
a_eq_b_o	OUT	1	This signal carries the result of the Equals to compare function. This signal toggles only when the design computed parameter, CMP_EQ is HIGH else unconnected in the integration. Core functionality is represented below. CMP_EQ and (data_a_i == data_b_i)

Signal Name	Direction	Width (bits)	Description
a_ne_b_o	OUT	1	This signal carries the result of the Not Equals to compare function. This signal toggles only when the design computed parameter, CMP_NE is HIGH else unconnected in the integration. Core functionality is represented below. CMP_NE and (data_a_i != data_b_i)
a_lt_b_o	OUT	1	This signal carries the result of the Less than compare function. This signal toggles only when the design computed parameter, CMP_LT is HIGH else unconnected in the integration. Core functionality is represented below. CMP_LT and (data_a_i < data_b_i)
a_le_b_o	OUT	1	This signal carries the result of the Less than or Equals to compare function. This signal toggles only when the design computed parameter, CMP_LE is HIGH else unconnected in the integration. Core functionality is represented below. CMP_LE and (data_a_i <= data_b_i)
a_gt_b_o	OUT	1	This signal carries the result of the Greater than compare function. This signal toggles only when the design computed parameter, CMP_GT is HIGH else unconnected in the integration. Core functionality is represented below. CMP_GT and (data_a_i > data_b_i)
a_ge_b_o	OUT	1	This signal carries the result of the Greater than or Equals to compare function. This signal toggles only when the design computed parameter, CMP_GE is HIGH else unconnected in the integration. Core functionality is represented below. CMP_GE and (data_a_i >= data_b_i)

2.4.2. Attributes

Table 2.8. Comparator Attributes

Configuration	Range	Default Value	Description
I_WIDTH	2-64	8	This configuration controls the width of the I/O, data_a_i and data_b_i. This configuration accepts integers only and any value less than 2 is automatically corrected to 2 by the design.
I_SIGNED	Signed or Unsigned	Unsigned	This configuration controls the interpretation of the inputs, data_a_i and data_b_i. Unsigned – Data are interpreted. Signed – MSB is the sign bit; remaining bit is the value of the data.

Configuration	Range	Default Value	Description
CMP_SEL	A != B A < B A <= B A = B A > B A >= B	A ! = B	This configuration controls the enabling of a compare function. 3'd0 – Equal to 3'd1 – Not Equal to 3'd2 – Less than 3'd3 – Less than or Equal to 3'd4 – Greater than 3'd5 – Greater than or Equal to others – All compare functions disabled This configuration is considered only when CMP_ALL parameter is configured LOW.
USE_OREG	on or off	off	This configuration controls the registering of the compare functions onto the outputs, a_eq_b_o, a_ne_b_o, a_lt_b_o, a_le_b_o, a_gt_b_o and a_ge_b_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 adder is high.
PIPELINES	0-N	0	This configuration controls the insertion of pipeline stages into the comparison operations. Editable only if I_WIDTH > 8. 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 I/O Width configuration. For example: 1-8 bits – 0 9-16 bits – 1 17-24 bits – 2 25-32 bits – 3 and so on This parameter is editable if D_WIDTH > 8
Internal Parameters			
I_WDT	—	—	(I_WIDTH < 2) ? 2 : I_WIDTH
CMP_EQ	—	—	CMP_SEL == 3’d0
CMP_NE	—	—	CMP_SEL == 3’d1
CMP_LT	—	—	CMP_SEL == 3’d2
CMP_LE	—	—	CMP_SEL == 3’d3
CMP_GT	—	—	CMP_SEL == 3’d4
CMP_GE	—	—	CMP_SEL == 3’d5

2.4.3. Timing Diagram

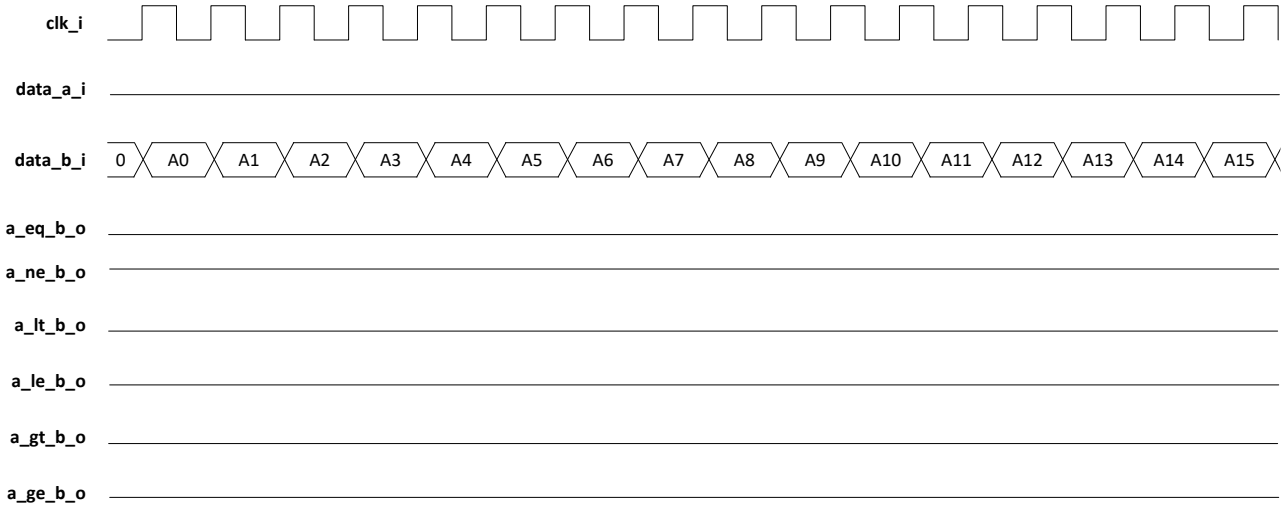

Figure 2.8. Comparator Timing Diagram

2.5. Counter

An Up, Down and Up-Down counter that counts a step of 1 per clock.

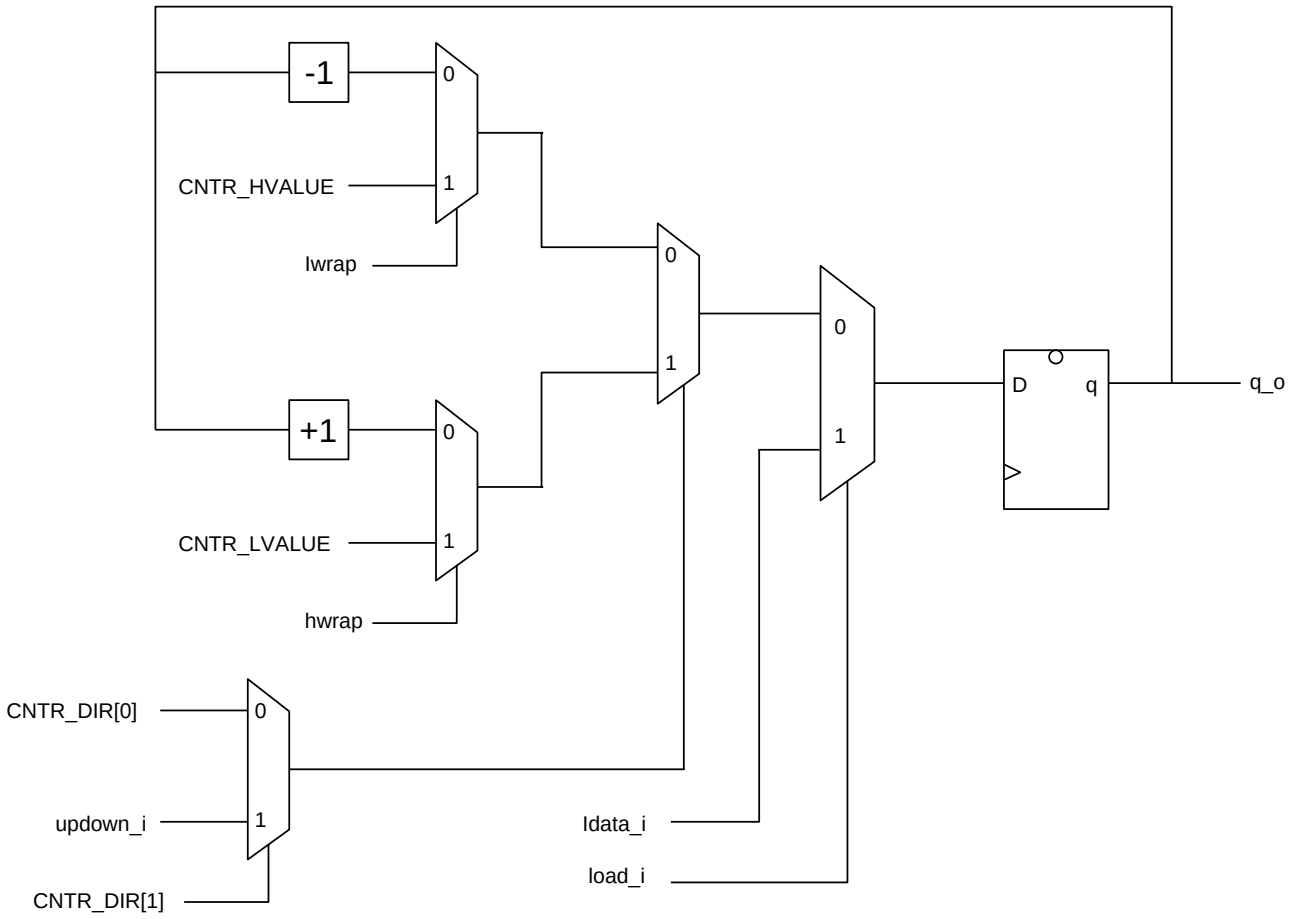

Figure 2.9. Counter Schematic Diagram

2.5.1. Ports

Table 2.9. Counter Ports

Signal Name	Direction	Width (bits)	Description
clk_i	IN	1	This signal connects to the clock pin of the flops in the design. 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
aclr_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 with the clock during integration.

Signal Name	Direction	Width (bits)	Description
updown_i	IN	1	This signal controls the direction of the counter. 1'b0 – Down Counter, that is, counter decrements 1'b1 – Up Counter, that is, counter increments CNTR_DIR[1] == 1'b0 – Ignored. Tied to a constant. CNTR_DIR[1] == 1'b1 – Considered in the design The resultant function is represented below. UpDown_w = CNTR_DIR[1] ? updown_i: CNTR_DIR[0]
load_i	IN	1	This signal controls the loading of the counter with a value. 1'b0 – Counter keeps counting 1'b1 – counter is loaded with the value of the input, ldata_i
ldata_i	IN	CNTR_WDT	This signal contains the load value of the counter. The value of this signal is loaded into the counter when load_i input is HIGH. Avoid changing the value of this signal when load_i goes LOW.
q_o	OUT	CNTR_WDT	This signal carries the counter value. Core functionality is represented below. If (aclr_i) q_o = CNTR_LVAL else if (load_i) q_o = ldata_i else if (UpDown_w) Q = hwrap ? CNTR_LVAL : {q_o + CNTR_STEP} else q_o = lwrap ? CNTR_HVAL : {q_o - CNTR_STEP} CNTR_STEP = 1;

2.5.2. Attributes

Table 2.10. Counter Attributes

Configuration	Range	Default Value	Description
CNTR_WIDTH	1 - 64	8	This configuration controls the width of the counter. This configuration accepts integers only and any value less than 2 is automatically corrected to 2 by the design.
CNTR_DIR	Down Up Up-Down	Up	This configuration controls the direction of the counter. 2'b1x – Input updown_i controls the direction of the counter 2'b00 – Down Counter, that is, counter decrements per clock 2'b01 – Up Counter, that is, counter increments per clock
CNTR_LVALUE	0 to (2^(CNTR_WIDTH)-1)	0	This configuration controls the Lowest counter value below which it does not Decrement (down count). The counter initializes to this value on reset assertion. Up Counter: The counter wraps to this value when it reaches the design computed parameter value, CNTR_HVAL. Down Counter: The counter wraps at this value and moves to the design computed parameter value, CNTR_HVAL. This parameter accepts integers only and if with a value greater than or equal to the design computed parameter, CNTR_HVAL, the design automatically corrects it to a value of CNTR_HVAL-1. This parameter is uneditable and fixed to 0 if CNTR_WIDTH>31

Configuration	Range	Default Value	Description
CNTR_HVALUE	0 to (2^(CNTR_WIDTH)-1)	255	This configuration controls the Highest counter value above which it does not Increment (Up count). Up Counter – The counter wraps at this value and moves to the design computed parameter value, CNTR_LVAL. Down Counter – The counter wraps to this value when it reaches the design computed parameter value, CNTR_LVAL. This parameter accepts integers only and if configured to 0, the design automatically corrects it to 1. This parameter is uneditable and fixed to 255 if CNTR_WIDTH>31
CNTR_LOAD	on or off	off	This configuration controls the loading of ldata_i to result q_o. off – counter increments/decrements per clock on – counter loads the value of ldata_i This parameter is uneditable and fixed to off if CNTR_WIDTH>31
Internal Parameters			
CNTR_WDT	—	—	(CNTR_WIDTH < 2) ? 2 : CNTR_WIDTH

2.5.3. Timing Diagram

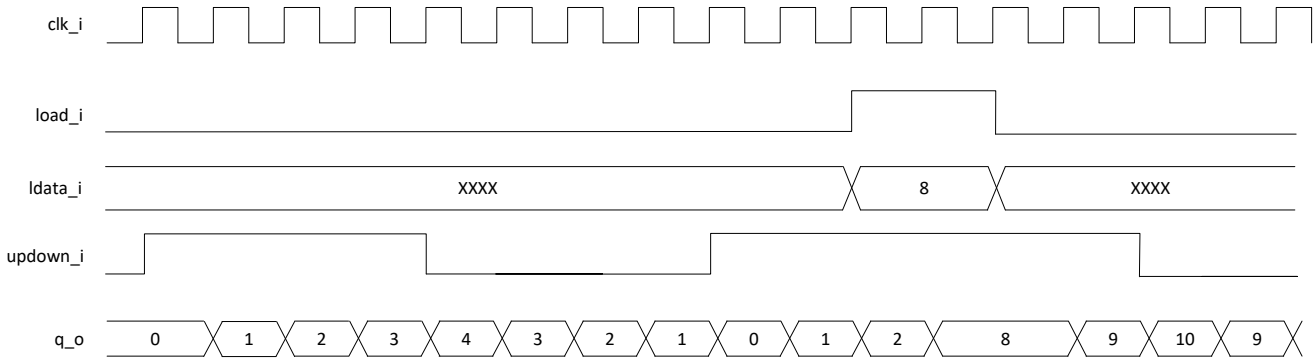

Figure 2.10. Counter Timing Diagram

2.6. Multiplier

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

The shaded portions of the Multiplier Schematic Diagram, as shown in Figure 2.11, are optional and are removed/ignored in certain configurations.

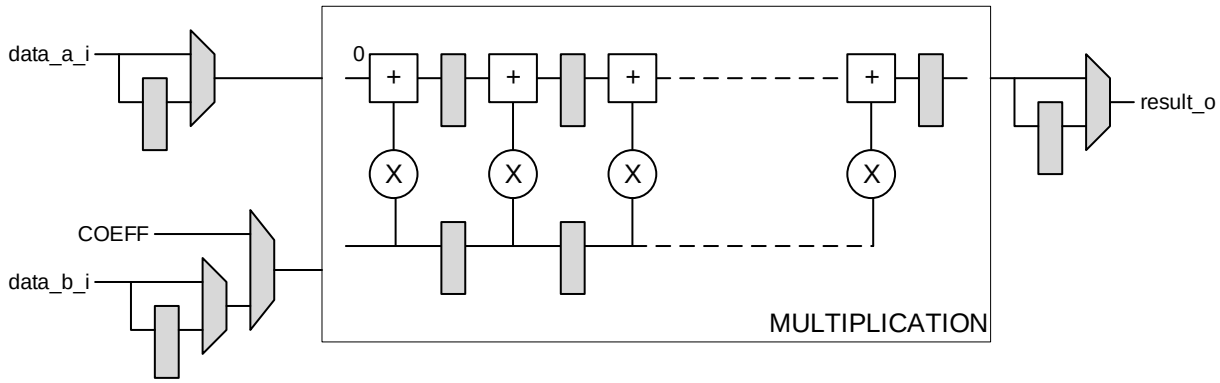

Figure 2.11. Multiplier Schematic Diagram

2.6.1. Ports

Table 2.11. 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 with the clock during integration.
data_a_i	IN	A_WIDTH	This signal contains an input for multiplication operation.
data_b_i	IN	B_WIDTH	This signal contains an input for multiplication operation.
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)

2.6.2. Attributes

Table 2.12. Multiplier Attributes

Configuration	Range	Default Value	Description
USE_COEFF	on, off	off	This configuration controls the multiplier input to the multiplication operation in the design. off – data_b_i input of the design is the multiplier on – COEFF configuration parameter is the multiplier

Configuration	Range	Default Value	Description
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_o, 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_i.
B_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_b_i.
USE_IREG	on, off	on	This configuration controls the registering of the inputs, data_a_i and data_b_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 multiplier result onto the output, result_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, the design automatically corrects it to the maximum limit. Otherwise, the configured value is honored: IMPLEMENTATION == 'LUT', 9 is the limiting factor IMPLEMENTATION == 'DSP', 3 is the limiting factor
IMPLEMENTATION	LUT, DSP	LUT	This configuration selects the resource to be used for implementation. LUT – Use LUT DSP – Use DSP blocks
O_WDT	4–128	18	This parameter is uneditable. It only gets the sum of A_WIDTH and B_WIDTH. InputA Width + InputB Width = Result Width
Internal Parameters			
COEFF_WDT	—	—	Actual bit width of the absolute (unsigned) COEFF value
X_WDT	—	—	COEFF_EN ? C_WDT : B_WIDTH;
M_WDT	—	—	A_WDT + X_WDT

2.6.3. Timing Diagram

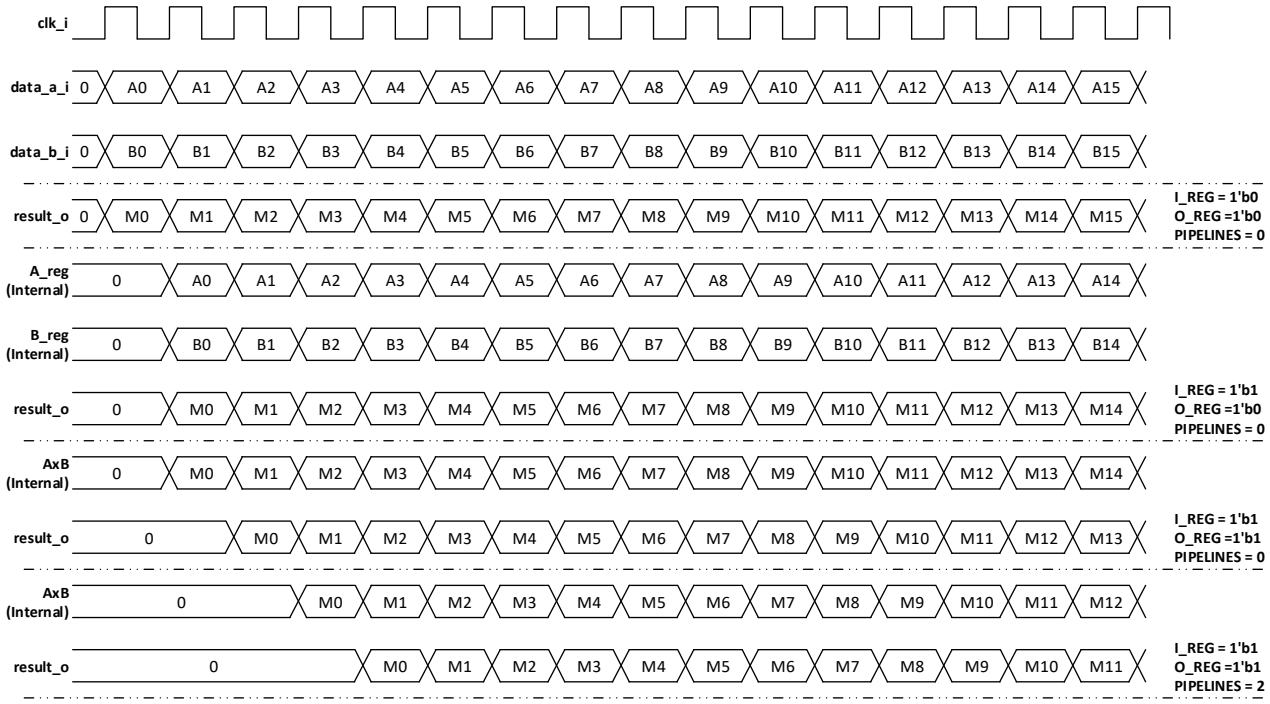

Figure 2.12. Multiplier Timing Diagram

2.7. Multiply_Accumulate

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

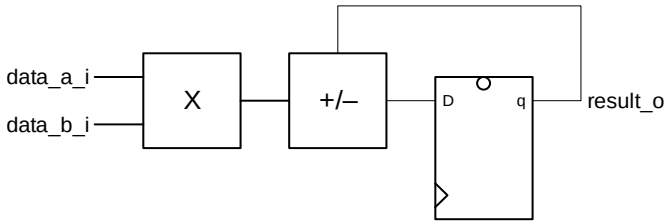

Figure 2.13. Multiply_Accumulate Schematic

2.7.1. Ports

Table 2.13. 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 with the clock during integration.
data_a_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication.
data_b_i	IN	B_WIDTH	This signal contains the multiplier value of the multiplication.
result_o	OUT	ACC_WIDTH	This signal carries the accumulated value of the product of inputs data_a_i and data_b_i. Core functionality is represented below. $\sum_{i=0}^{n} DataA_i * DataB_i$

2.7.2. Attributes

Table 2.14. Multiply_Accumulate Attributes

Configuration	Range	Default Value	Description
A_WIDTH	2–64	18	This configuration controls the width of the input data_a_i.
B_WIDTH	2–64	18	This configuration controls the width of the input data_b_i.
ACC_WIDTH	[(A_WIDTH+ B_WIDTH+1), (A_WIDTH+ B_WIDTH+32)]	37	This configuration controls the accumulator width wherein it must be wider than the sum of A_WIDTH and B_WIDTH by 1 to 32 bits.
A_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_a_i.
B_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the input data_b_i.
ADD_SUB	Addition or Subtraction	Addition	This configuration controls the operation to be performed on the accumulated value and multiplication product.
USE_IREG	on, off	off	This configuration controls the registering of the inputs, data_a_i and data_b_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.

Configuration	Range	Default Value	Description
USE_OREG	on, off	on	This configuration controls the registering of the multiplier result onto the output, result_o. 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, the design automatically corrects it to the maximum limit. Otherwise, the configured value is honored. IMPLEMENTATION == 'LUT', 9 is the limiting factor IMPLEMENTATION == 'DSP', 3 is the limiting factor
IMPLEMENTATION	LUT, DSP	LUT	This configuration selects the resource to be used for implementation. LUT – Use LUT DSP – Use DSP blocks
O_WDT	[(A_WIDTH+ B_WIDTH+1), (A_WIDTH+ B_WIDTH+32)]	37	This parameter is uneditable. It only gets the value of ACC_WIDTH. Input A Width + Input B Width + Accumulate Width = Result Width
Internal Parameters			
M_WDT	—	—	A_WDT + X_WDT

2.7.3. Timing Diagram

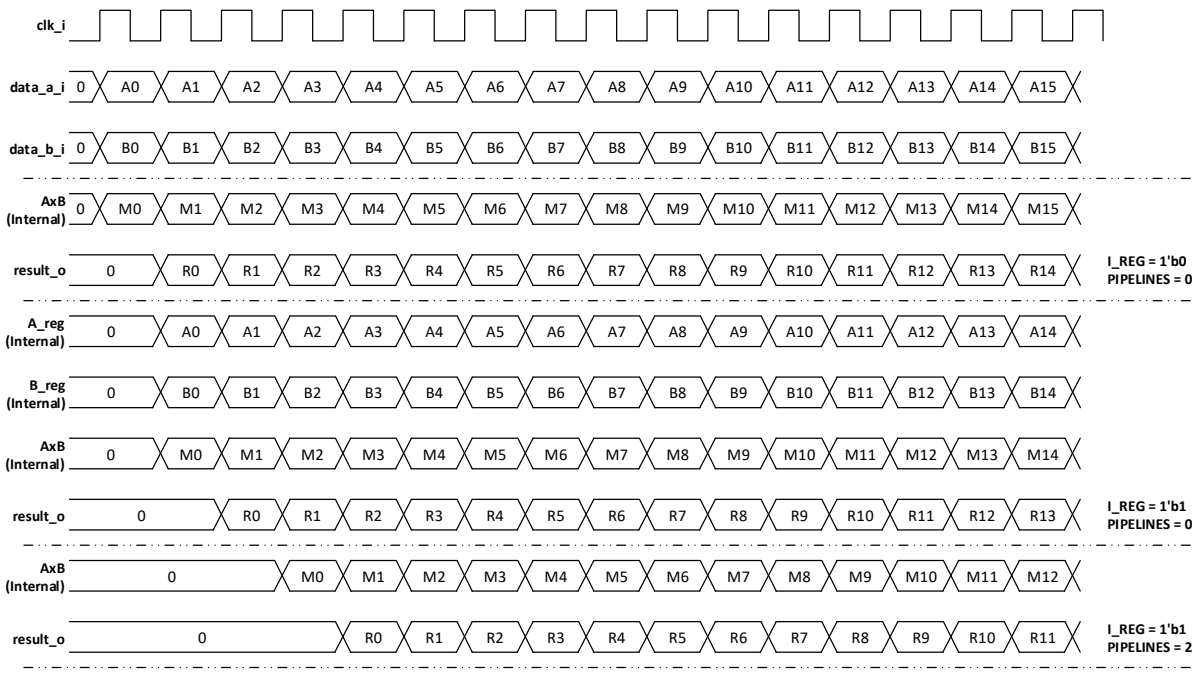

Figure 2.14. Multiply_Accumulate Timing Waveforms

2.8. 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_o carries the Sum/Difference of Products.

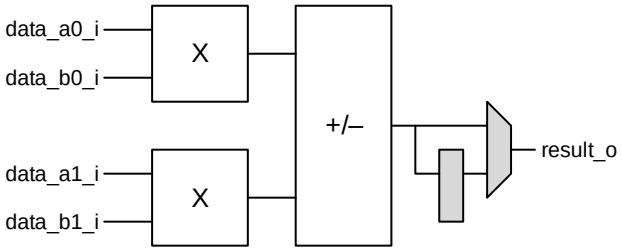

Figure 2.15. Multiply_Add_Subtract Schematic

2.8.1. Ports

Table 2.15. 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 – Retains the previous state 1'b1 – Toggles 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 with the clock during integration.
data_a0_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication pair data_a0_i - data_b0_i.
data_a1_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication pair data_a1_i - data_b1_i.
data_b0_i	IN	B_WIDTH	This signal contains the multiplier value of the multiplication pair data_a0_i - data_b0_i.
data_b1_i	IN	B_WIDTH	This signal contains the multiplier value of the multiplication pair data_a1_i - data_b1_i.
result_o	OUT	M_WDT	This signal carries the result_o of the addition/subtraction of the multiplication products. Core functionality is represented below. M0 = (data_a0_i × data_b0_i) M1 = (data_a1_i × data_b1_i) ADD_SUB== add: {M0 + M1} ADD_SUB== sub: {M0 – M1}

2.8.2. Attributes

Table 2.16. Multiply_Add_Subtract Attributes

Configuration	Range	Default Value	Description
ADD_SUB	Addition or Subtraction	Addition	This configuration controls the operation to be performed on the multiplication product pairs, data_a0_i-data_b0_i and data_a1_i - data_b1_i.
A_WIDTH	2–64	9	This configuration controls the width of the inputs data_a0_i and data_a1_i.
B_WIDTH	2–64	9	This configuration controls the width of the inputs data_b0_i and data_b1_i.
A_SIGNED	off, on	on	This configuration controls the interpretation of the input data_a0_i and data_a1_i. off – Unsigned number on – Signed number
B_SIGNED	off, on	on	This configuration controls the interpretation of the input data_b0_i and data_b1_i. off – Unsigned number on – Signed number
USE_IREG	on, off	on	This configuration controls the registering of the inputs, data_a* and 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_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, the design automatically corrects it to the maximum limit. Otherwise, the configured value is honored. IMPLEMENTATION == 'LUT', 9 is the limiting factor IMPLEMENTATION == 'DSP', 3 is the limiting factor
IMPLEMENTATION	LUT, DSP	LUT	This configuration selects the resource to be used for implementation. LUT – Use LUT DSP – Use DSP blocks
O_WDT	5–129	19	This parameter is uneditable. It only gets the sum of A_WIDTH, B_WIDTH + 1. Input A Width + Input B Width + 1 = Result Width
Internal Parameters			
M_WDT	—	—	A_WDT + X_WDT
O_WDT	—	—	M_WDT + 1

2.8.3. Timing Diagram

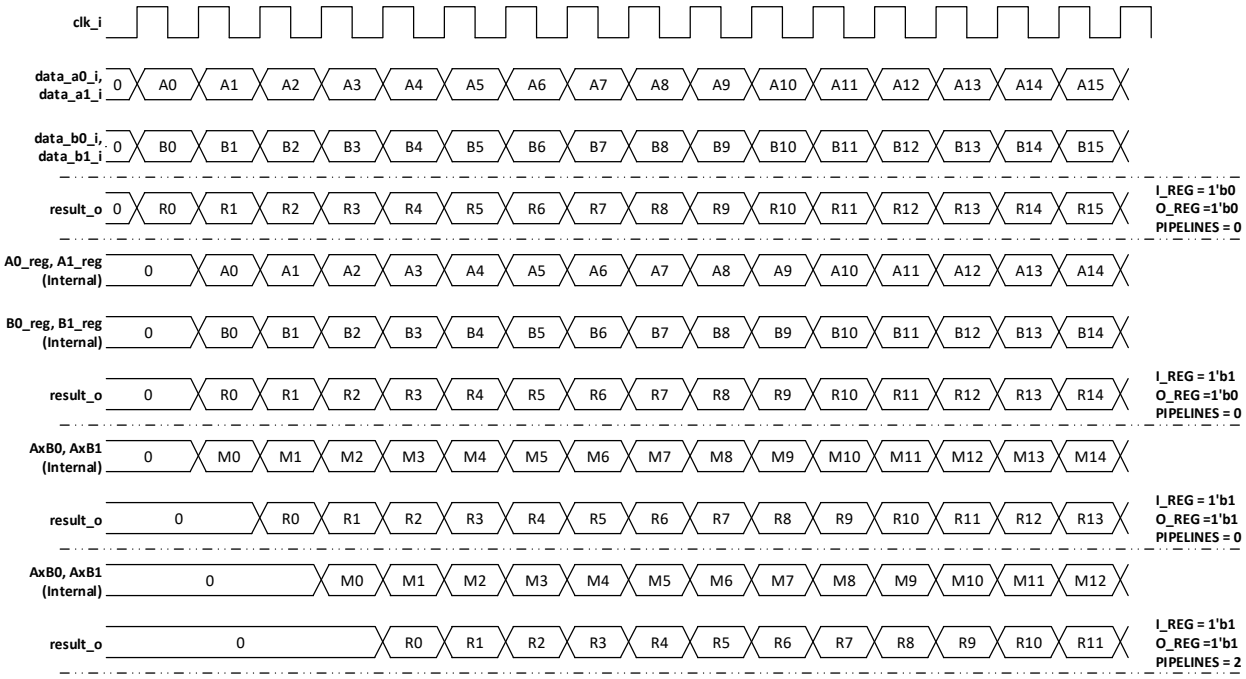

Figure 2.16. Multiply_Add_Subtract Timing Diagram

2.9. Multiply-Add-Subtract-Sum

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

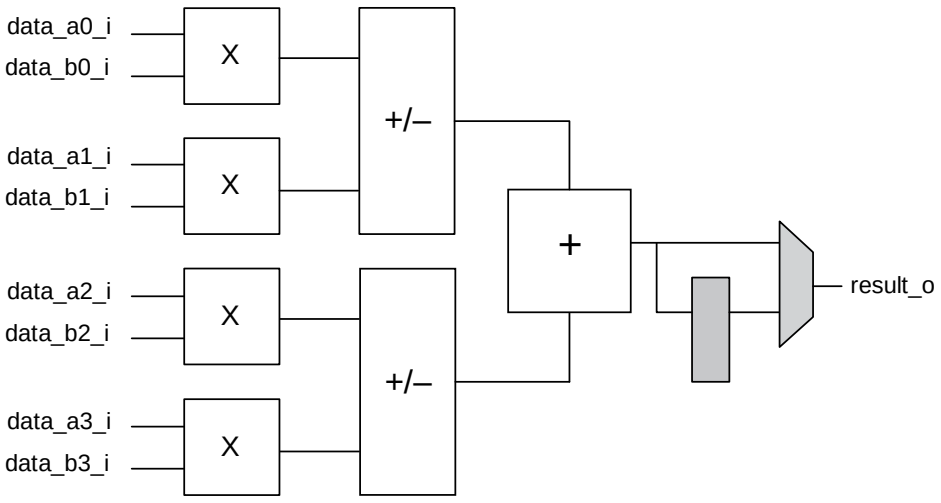

Figure 2.17. Multiply_Add_Subtract_Sum Schematic Diagram

2.9.1. Ports

Table 2.17. Multiply_Add_Subtract_Sum 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 of the flops in the design. 1'b0 – Retains the previous state 1'b1 – Toggles 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 with the clock during integration.
data_a0_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication pair, data_a0_i - data_b0_i.
data_a1_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication pair, data_a1_i - data_b1_i.
data_a2_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication pair, data_a2_i - data_b2_i.
data_a3_i	IN	A_WIDTH	This signal contains the multiplicand value of the multiplication pair, data_a3_i - data_b3_i.
data_b0_i	IN	B_WIDTH	This signal contains the multiplier value of the multiplication pair, data_a0_i - data_b0_i.
data_b1_i	IN	B_WIDTH	This signal contains the multiplier value of the multiplication pair, data_a1_i - data_b1_i.
data_b2_i	IN	B_WIDTH	This signal contains the multiplier value of the multiplication pair, data_a2_i - data_b2_i.
data_b3_i	IN	B_WIDTH	This signal contains the multiplier value of the multiplication pair, data_a3_i - data_b3_i.
result_o	OUT	O_WDT	This signal carries the sum of the addition/subtraction of the multiplication product pairs data_a0_i - data_b0_i and data_a1_i - data_b1_i with data_a2_i - data_b2_i and data_a3_i - data_b3_i. Core functionality is represented below. M0 = (data_a0_i × data_b0_i) M1 = (data_a1_i × data_b1_i) M2 = (data_a2_i × data_b2_i) M3 = (data_a3_i × data_b3_i) { ADD_SUB0 ? {M0 + M1} : {M0 – M1} + ADD_SUB1 ? {M2 + M3} : {M2 – M3}}

2.9.2. Attributes

Table 2.18. Multiply_Add_Subtract_Sum Attributes

Configuration	Range	Default Value	Description
IMPLEMENTATION	DSP, LUT	LUT	This configuration selects the resource to be used for implementation. DSP – Use DSP blocks LUT – Use LUTs
ADD_SUB0	Add or Sub	Add	This configuration controls the operation to perform on the multiplication product pairs, data_a0_i - data_b0_i and data_a1_i - data_b1_i. Add – Addition Sub – Subtraction
ADD_SUB1	Add or Sub	Add	This configuration controls the operation to perform on the multiplication product pairs, data_a2_i - data_b2_i and data_a3_i - data_b3_i. Add – Addition Sub – Subtraction
A_WIDTH	2-64	9	This configuration controls the width of the inputs, data_a0_i, data_a1_i, data_a2_i and data_a3_i. This configuration accepts integers only and any value less than 2 is automatically corrected to 2 by the design.
A_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the inputs, data_a0_i, data_a1_i, data_a2_i and data_a3_i.
B_WIDTH	2-64	9	This configuration controls the width of the inputs, data_b0_i, data_b1_i, data_b2_i and data_b3_i. This configuration accepts integers only and any value less than 2 is automatically corrected to 2 by the design.
B_SIGNED	Signed or Unsigned	Signed	This configuration controls the interpretation of the inputs, data_b0_i, data_b1_i, data_b2_i and data_b3_i.
PIPELINES	0 - A_WDT or 0 – B_WDT	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_WDT and B_WDT, the design automatically corrects it to the greater of A_WDT and B_WDT.
USE_IREG	on, off	on	This configuration controls the registering of the inputs, data_a0_i, data_a1_i, data_a2_i, data_a3_i, data_b0_i, data_b1_i, data_b2_i and data_b3_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 output, result_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 operation is high.

Configuration	Range	Default Value	Description
Internal Derivations			
A_WDT	—	—	(A_WIDTH< 2) ? 2 : A_WIDTH
B_WDT	—	—	(B_WIDTH< 2) ? 2 : B_WIDTH
M_WDT	—	—	A_WDT + B_WDT
O_WDT	—	—	M_WDT + 2

2.9.3. Timing Diagram

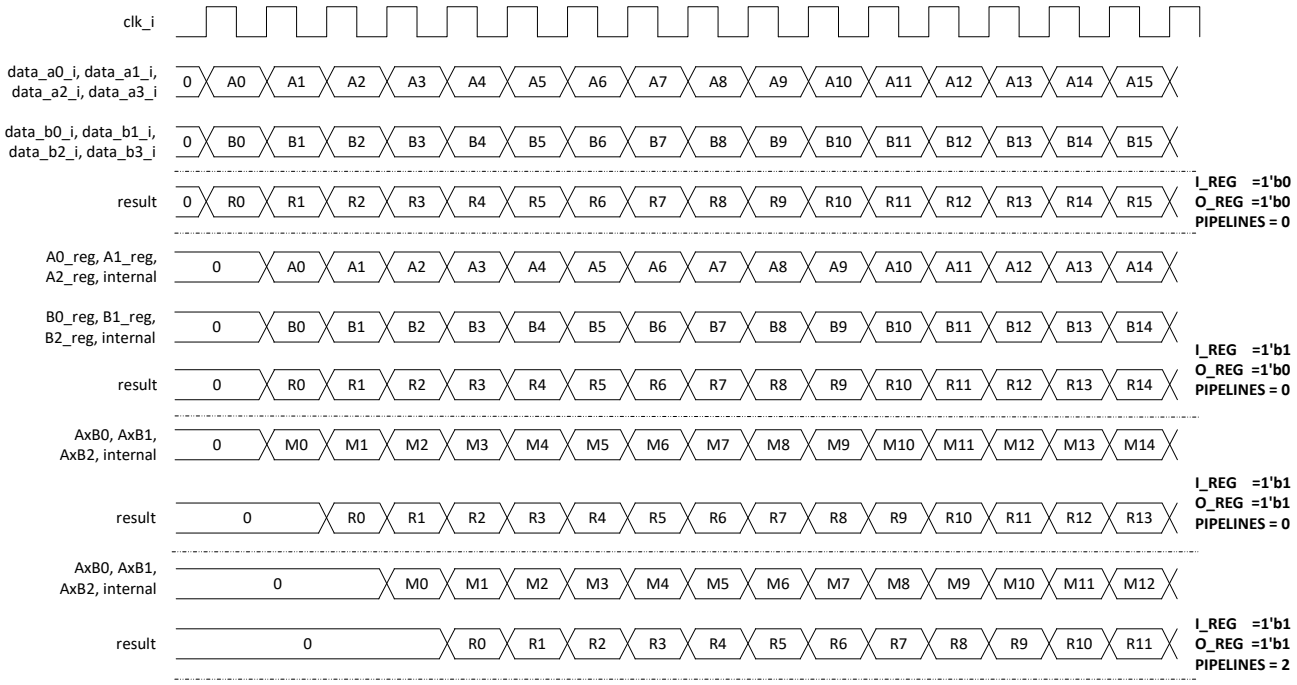

Figure 2.18. Multiply_Add_Subtract_Sum Timing Diagram

2.10. 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.

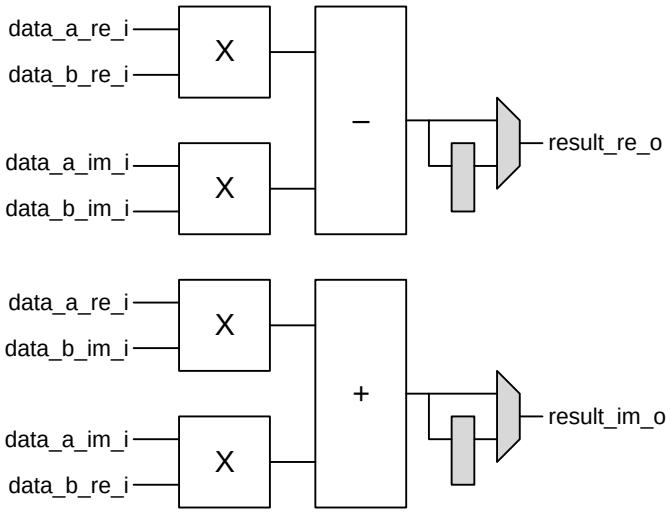

Figure 2.19. Complex_Multiplier Schematic Diagram

2.10.1. Ports

Table 2.19. 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 – Retains the previous state 1'b1 – Toggles 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 with 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)}

Signal Name	Direction	Width (bits)	Description
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)}

2.10.2. Attributes

Table 2.20. 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	Unsigned	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.
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', 9 is the limiting factor IMPLEMENTATION == 'DSP', 3 is the limiting factor
IMPLEMENTATION	LUT, DSP	LUT	This configuration selects the resource to be used for implementation. LUT – Use LUT DSP – Use DSP blocks
Internal Parameters			
M_WDT	—	—	A_WDT + X_WDT
O_WDT	—	—	M_WDT + 1

2.10.3. Timing Diagram

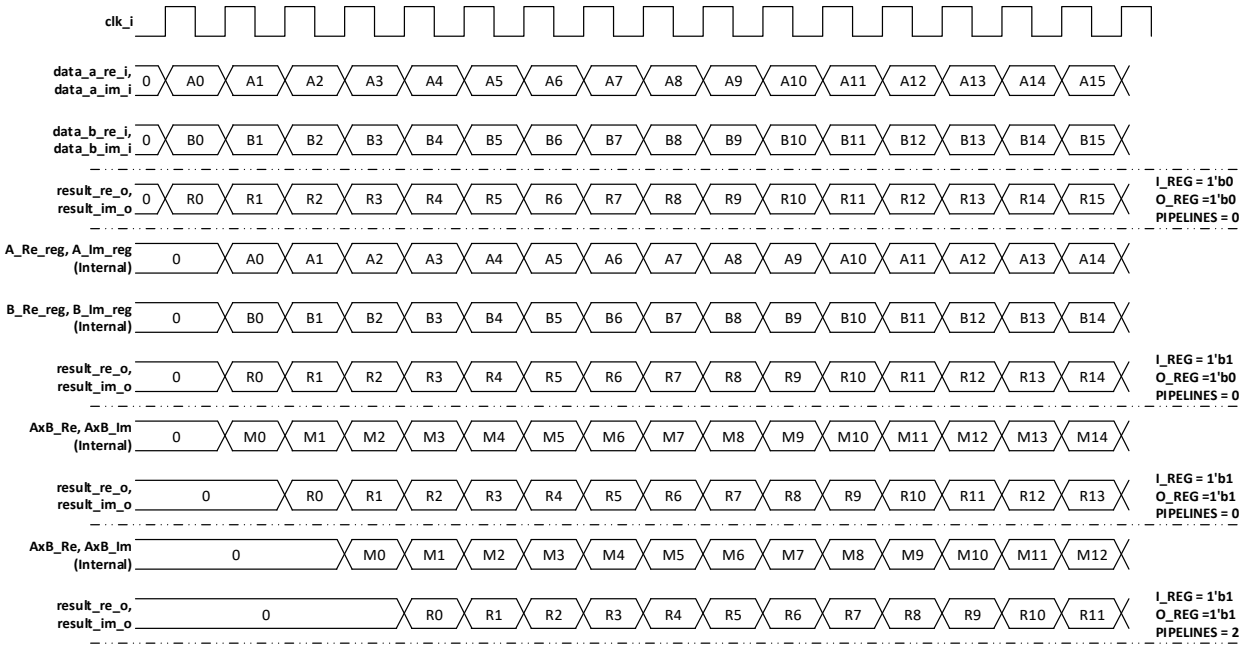

Figure 2.20. Complex_Multiplier Timing Diagram

2.11. LFSR

Linear Feedback Shift Register that supports following configurations.

Types – Fibonacci and Galois

- LFSR Width – Up to 512-bits
- Polynomial – Up to 64-bits

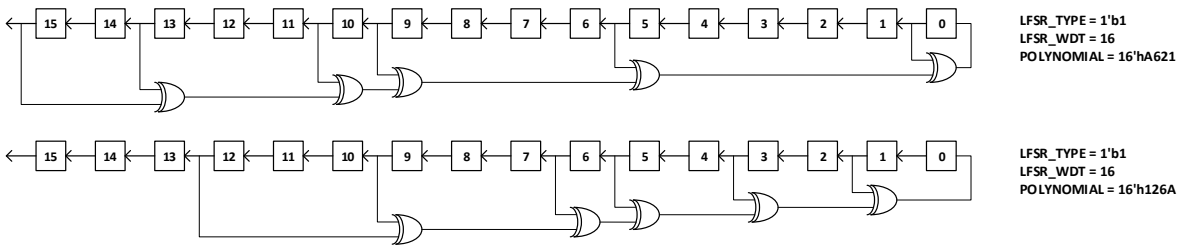

Figure 2.21. Fibonacci LFSR Illustrative Shifter Implementation Schematic

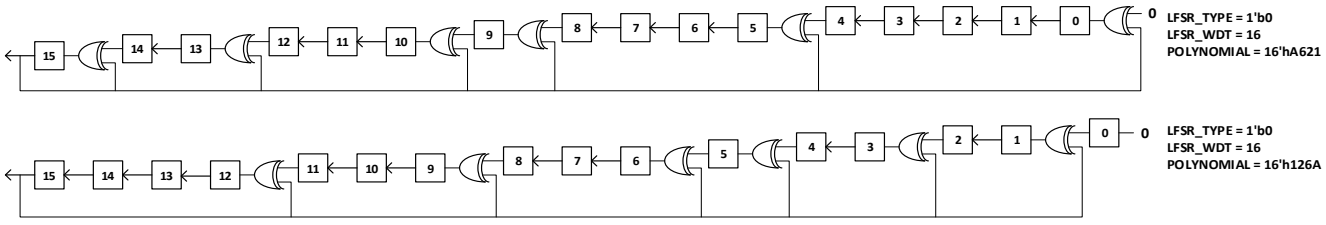

Figure 2.22. Galois LFSR Illustrative Shifter Implementation Schematic

2.11.1. Ports

Table 2.21. LFSR Ports

I/O Name	Direction	Width (bits)	Description
clk_i	IN	1	This signal connects to the clock pin of the flops in the design. All flops toggle on the rising edge of this clock.
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 with the clock, clk_i during integration.
enb_i	IN	1	This signal controls the enabling of the LFSR. 1'b0 – Disabled, that is, LFSR does not function. Last state retained. 1'b1 – Enabled, that is, LFSR functions
len_i	IN	1	This signal controls the loading of the LFSR with the seed. 1'b0 – LFSR keeps functioning 1'b1 – LFSR is loaded with the value of the input, din_i
din_i	IN	LFSR_WIDTH	This signal contains the seed of the LFSR. The value of this signal is loaded into the LFSR when len_i input is HIGH. Avoid changing the value of this signal when len_i goes LOW.

dout_o	OUT	O_WDT	This signal carries the LFSR value.
			O_PARALLEL == 1'b0 – Only MSB of the shift register is output
			O_PARALLEL == 1'b1 – Complete shift register is output
			
			Core functionality is represented below.
			if(LFSR_TYPE)
			begin // Fibonacci
			lfsr_mask = (lshifter & LFSR_POLY)
			lfsr_xor = ^lfsr_mask
			lshifter_c = {lshifter [LFSR_WDT-2:0], (LFSR_GATE ^ lfsr_xor)}
			end
			else
			begin // Galois
			lfsr_xor = (LFSR_GATE ^ lshifter[LFSR_WDT-1])
			lfsr_mask = (LFSR_POLY & {LFSR_WDT{lfsr_xor}})
			lshifter_c = ({lshifter << 1} ^ lfsr_mask)
			end
			if(rst_i) lshifter = LFSR_INIT
			else if(len_i) lshifter = Din
			else if(enb_i) lshifter = lshifter_c
			
			if(O_PARALLEL) dout_o = lshifter
			else dout_o = lshifter[LFSR_WDT-1]

2.11.2. Attributes

Table 2.22. LFSR Attributes

Configuration	Range	Default Value	Description
LFSR_TYPE	Fibonacci or Galois	Fibonacci	This configuration controls the type of LFSR implemented.
LFSR_GATE	XOR or XNOR	XOR	This configuration controls the gate implemented in the LFSR.
LFSR_WIDTH	1-512	8	This configuration controls the width of the shift register. This configuration accepts integers only and any value less than 1 is automatically corrected to 1 by the design.
POLYNOMIAL	0 – 2^POLY_WDT -1	00	This configuration contains the POLYNOMIAL in hexadecimal format of the LFSR. The width of this configuration is limited to 64-bits or the design computed parameter, LFSR_WDT, whichever is smaller. This polynomial is applied on the least significant bits of the LFSR while no polynomial is applied on the registers exceeding 64-bits and the whole LFSR is left shifted in either case.
LFSR_INIT	0 – 2^LFSR_WDT -1	01	This configuration contains the initialization value in hexadecimal format of the LFSR on assertion of the IP Reset. The width of this configuration is limited to the design computed parameter, LFSR_WDT.
O_PARALLEL	on, off	on	This configuration controls the output of the LFSR from the IP. off – Serial, that is, MSB of the shift register is only output from the IP. The output, dout_o is 1-bit wide. on – Parallel, that is, complete shift register is output from the IP. The width of the output, dout_o is determined by the design computed parameter, LFSR_WDT.
LOAD_SEED	on, off	off	This configuration controls the enabling of Reloadable seed. off – Uses LFSR_INIT as the seed after reset deassertion and does not permit reloading the seed during the operation phase. len_i input is tied LOW in this configuration. on – Uses din_i as the seed whenever the input len_i is set HIGH, which permits reloading the seed during the operation phase. len_i input is provided to you in this configuration to load the seed dynamically.
Internal Derivations			
LFSR_WDT	—	—	(LFSR_WIDTH < 1) ? 1 : LFSR_WIDTH
O_WDT	—	—	O_PARALLEL ? LFSR_WDT : 1

2.11.3. Timing Diagram

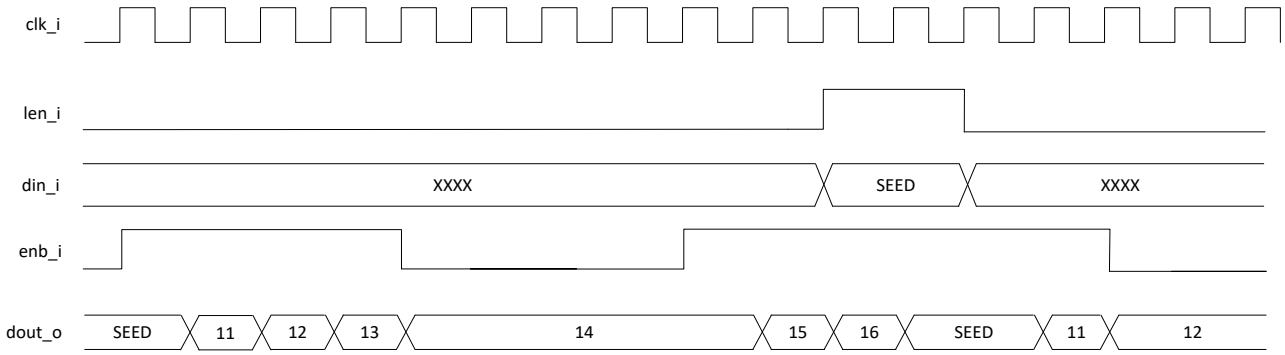

Figure 2.23. LFSR Timing Diagram

2.12.Convert

The Convert block converts from one format to another, performing rounding or saturation as required.

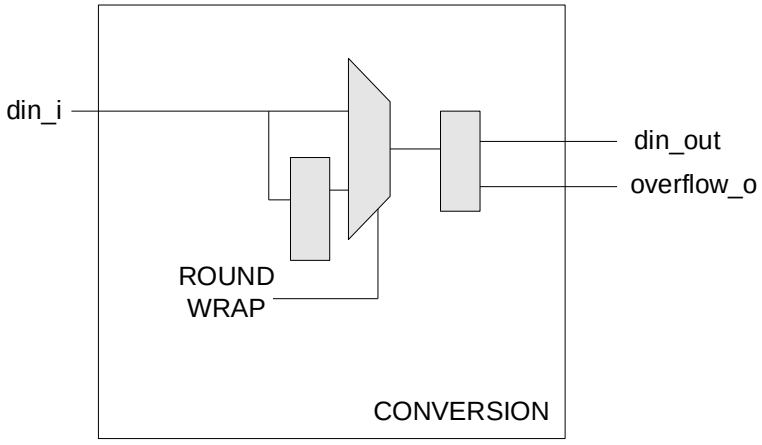

Figure 2.24 Convert Schematic Diagram

2.12.1. Ports

Table 2.23. Convert Ports

Signal Name	Direction	Width (Bbits)	Description
din_i	IN	INPUTWIDTH	The signal contains the input data
dout_o	OUT	OUTPUTWIDTH	Output Data in coverted format
overflow_o	OUT	1	Overflow flag

2.12.2. Attributes

Table 2.24. Convert Attributes

Configuration	Range	Default Value	Description
INPUTWIDTH	1 - 256	8	This configuration controls the bit width of the input data, din_i. A positive integer that specifies the total number of bits that are used to represent the data in twos complement format.
INPUTPOINT	0 – 7	0	This configuration controls the position of the binary point in the input data. Valid values for unsigned data are from 0 to the input width. Valid values for signed data are from 0 to the input width minus 1. The MSB is treated as the sign bit.
OUTPUTWIDTH	1 - 256	8	This configuration controls the bit width of the output data, dout_o. A positive integer that specifies the total number of bits that are used to represent the data in twos complement format.
OUTPUTPOINT	0 – 7	0	This configuration controls the position of the binary point in the output data. Valid values for unsigned word are from 0 to the output width. Valid values for signed data are from 0 to the output width minus 1. The MSB is treated as the sign bit.
INPUTFORMATUNSIGNED	Signed Unsigned	Signed	This configuration controls the interpretation of the input din_i.
ROUND	Truncate Nearest Convergent	Truncate	Three settings exist and determine the manner in which the input format value is shortened to the output format: Truncate, Nearest, and Convergent. *Truncate* – shortens the format by simply discarding the excess bits of the larger input format over the output format. *Nearest* – adds or subtracts an LSB if the excess bits sum to an absolute value that exceeds an LSB of the output format. *Convergent* – same as Nearest except when the excess bits add to exactly one-half an LSB, in which case a refinement to Nearest algorithm is employed. If the number of output fractional bits is less than the input, then quantization is done according to this selection. Note that choosing any selection other than Truncate results in an increase in the hardware requirements and potentially a lower overall performance.
WRAP	Wrap Min_Max	Wrap	If the number of integer output bits is less than that of the input, then any overflow or underflow is treated according to this selection. *Wrap* – discards excess bits. *Min_Max* – sets to minimum or maximum value
Input Parameters			
ROUNDEDWIDTH	—	—	INPUTWIDTH - (INPUTPOINT - OUTPUTPOINT) + 1
SATINPUTWIDTH	—	—	INPUTWIDTH + OUTPUTPOINT-INPUTPOINT + 1

2.12.3. Timing Diagram

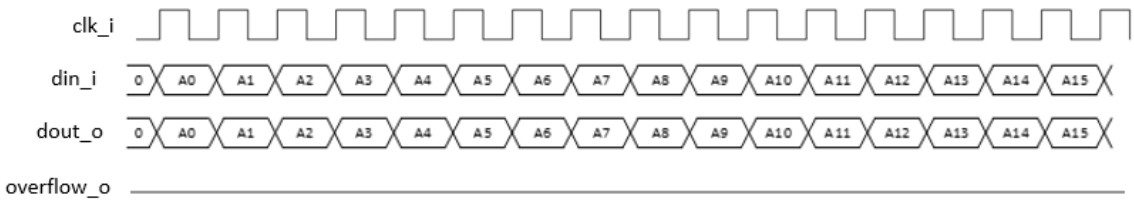

Figure 2.25. Convert Timing Diagram

2.13. Sin-Cos Table

Module for look-up tables for trigonometric sine and cosine functions. These tables can be implemented in EBR blocks or distributed ROMs.

Shaded signal names of the Sin-Cos Table Schematic diagram, as shown in Figure 2.26, are optional and are removed/ignored in certain configurations.

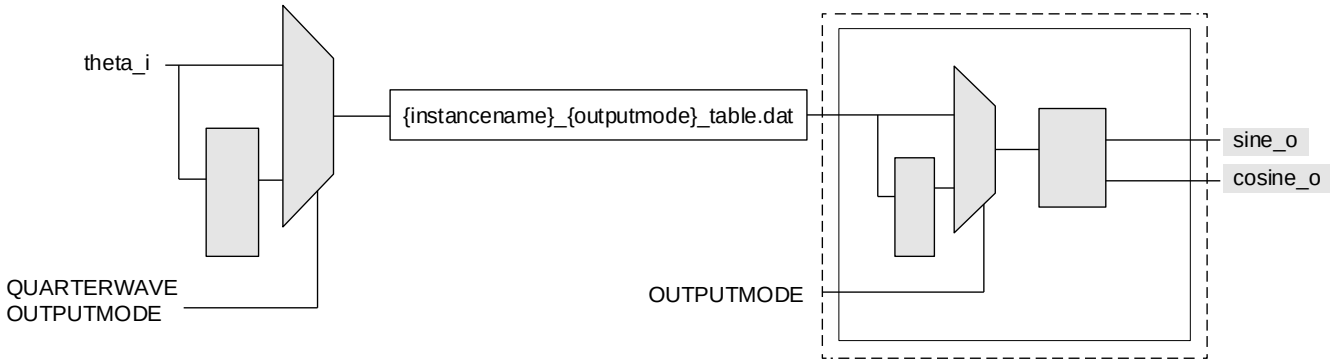

Figure 2.26. Sin-Cos Table Schematic Diagram

2.13.1. Ports

Table 2.25. Sin-Cos Table 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.
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.
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
theta_i	IN	I_WIDTH	Input angle which is given in radians: 360º = 2π radians or 1 radian = 180º/π.
sine_o	OUT	O_WIDTH	This signal carries the result of the sine function. It enables only if OUTPUT_MODE = Sin or OUTPUT_MODE = Sin-Cos
cosine_o	OUT	O_WIDTH	This signal carries the result of the cosine function. It enables only if OUTPUT_MODE = Cos or OUTPUT_MODE = Sin-Cos

2.13.2. Attributes

Table 2.26. Sin-Cos Table Attributes

Configuration	Range	Default Value	Description
TABLE_IMPL	LUT EBR	LUT	This configuration selects the resource to be used for implementation. LUT – Use LUTs EBR – Use EBR blocks
I_WIDTH	3-10	8	This configuration determines the the width of the Theta input bus. This determines precision with which the angle can be specified. The value of Theta is in radians, from 0 to something less than 2π, depending on the width, and in the form of unsigned fixed-point data. The data has a scaling factor of π/2^(width - 1). For example, with an 8-bit Theta, π radians (180º) is given as π x 2^(8 - 1)/π = π x 128/π = h'80.
O_WIDTH	4-32	8	This configuration controls the output width of the Sine and Cosine ports.
OUTPUT_MODE	Sin Cos Sin-Cos	Sin	This configuration controls the selection of function to which the table is used. Take note that more memory resources may be used when the Sin-Cos option is selected, but in some cases, this may be better than two independent Sin and Cos modules.
QUARTERWAVE	on, off	on	This configuration controls the memory reduction by only storing a quarter wave and deriving the rest of it with additional logic.
SIGNED_INT	on, off	off	This configuration controls the value of the output if it should be in twos complement format or sign-and-magnitude format. off – The values are in sign-and-magnitude format with a scaling factor of 1/(2^(width - 2)) on – The values are in twos complement format with a scaling factor of 1/(2^(width - 1)).
REG_INPUT	on, off	off	This configuration controls the registering of the input. off – Unregistered. Inputs are directly routed. on – Registered. Inputs are registered before routing.
REG_OUTPUT	on, off	on	This configuration controls the registering of the output. off – Unregistered. Result is directly routed to the output. on – Registered. Result is registered at the output. Output registers are automatically included in the LUT implementation.
NUM_PIPE	1-3	1	This configuration controls the insertion of pipeline stages into the sine/cosine outputs.
INITFILE	N/A	{InstanceName}_ {OutputMode}_ table.dat	This is the generated dat file which contains the sin-cos table in hexadecimal form. This parameter is uneditable as it is automatically generated and renamed based on the output mode.
TABLE_FULL_PATH	N/A	{ProjectLocation}/ {InstanceName}_ {OutputMode}_table.dat	This parameter is uneditable as it is automatically generated and renamed based on the location of project and the output mode.
Internal Parameters			
O_WIDTH_DH_SC	—	—	(O_WIDTH/4.0000) == 1.0000 ? {O_WIDTH*2} : {O_WIDTH*4};
DWID	—	—	(OUTPUT_MODE == "SIN-COS")? O_WIDTH_DH_SC : O_WIDTH;

Configuration	Range	Default Value	Description
QTWID	—	—	(I_WIDTH - 2)
MSIZ	—	—	(QUARTERWAVE)? (1 << QTWID)+1 : (1 << I_WIDTH)
AWID	—	—	$clog2(MSIZ)
REGMODE	—	—	(REG_OUTPUT) ? "reg" : "noreg"

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 available arithmetic modules in the Lattice Radiant Software IP catalog are:

- Adder
- Adder_Subtractor
- Comparator
- Complex_Mult
- Convert
- Counter
- LFSR
- Mult_Accumulate
- Mult_Add_Sub
- Multiplier
- Mult_Add_Sub_Sum
- Sin-Cos Table
- Subtractor

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

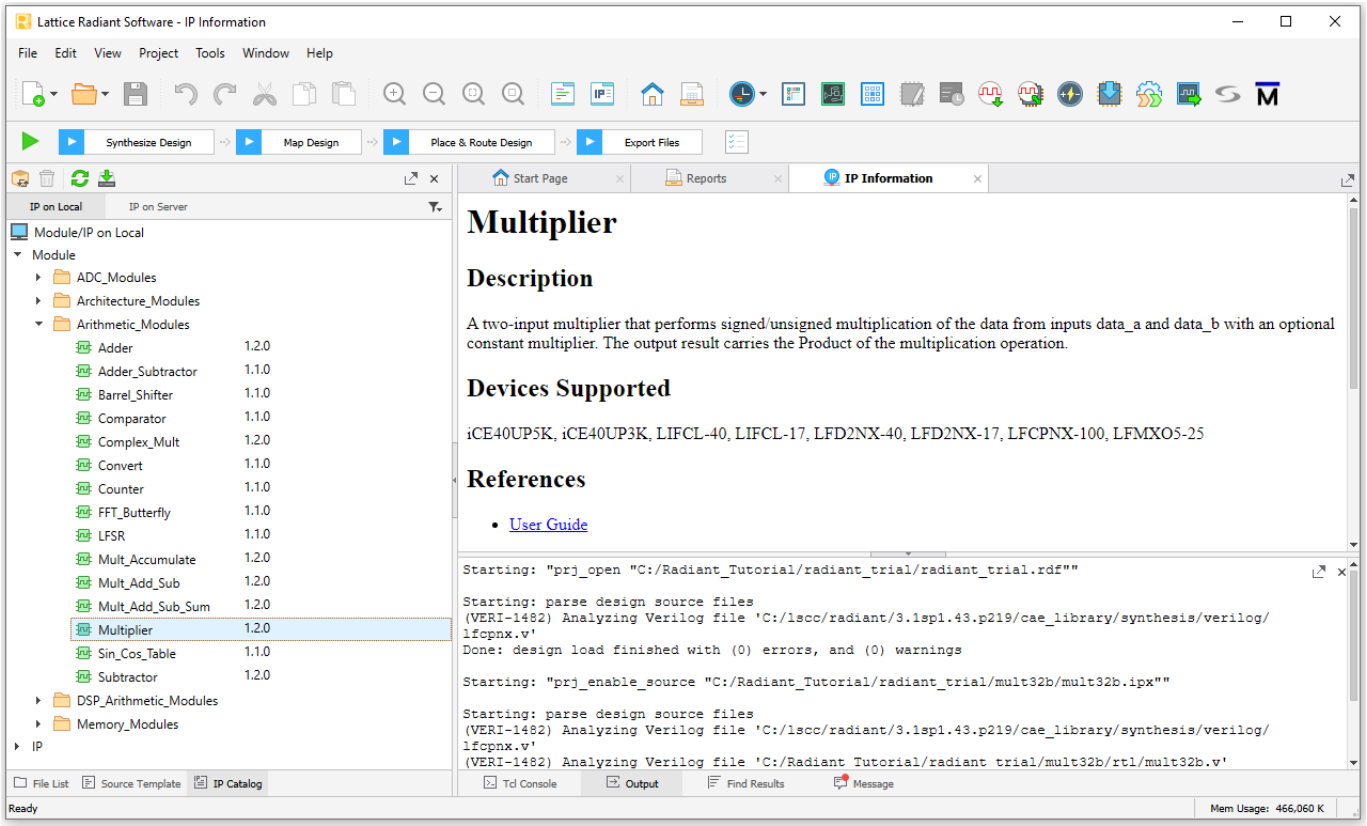

Figure 3.1. Arithmetic Modules under Module/IP on Local in the Lattice Radiant Software

3.1. Generating a 32-bit Multiplier Module

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

To generate a 32-bit Multiplier module:

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

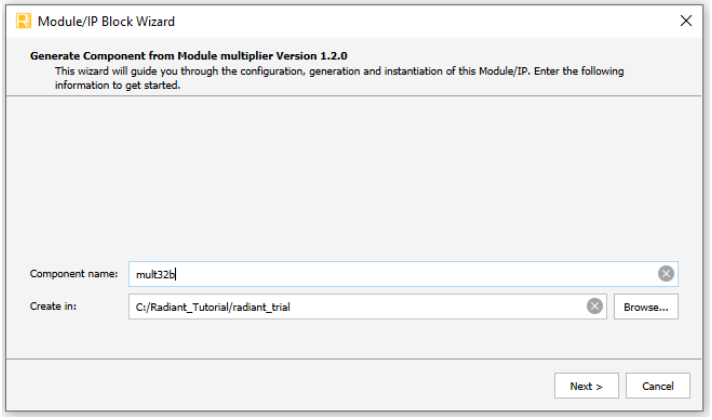

Figure 3.2. Example: Generating 32-bit Multiplier Using Module/IP Block Wizard

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

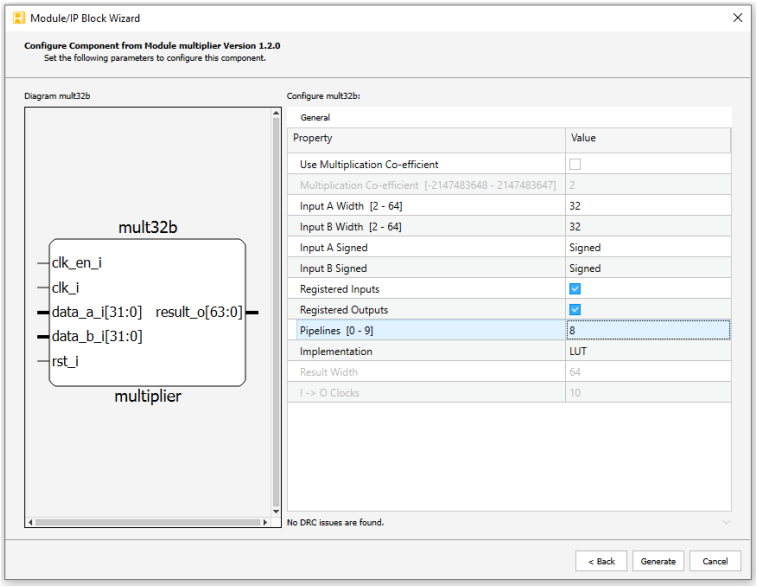

Figure 3.3. Example: Generating 32-bit Multiplier in IP Configuration

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

Once this module is in the Lattice Radiant Software project, it can be instantiated in other modules within the project. You can view the files and instance/s added in the **File List** tab.

4. Simulation Flow

Lattice Radiant Software also allows you to simulate configured arithmetic modules, so you can observe the conditions of the output with a given stimuli. This section details the individual steps needed to execute the testbench in Modelsim. In the sample procedure, a Multiplier on default settings is generated, including an IP name for the multiplier.

Table 4.1 provides a summary description of the files used in the project.

Table 4.1. File List

File	Sim	Synthesis	Description
<IP_name>.v	Yes	Yes	Top Level RTL file with the selected configuration. The main IP file
dut_params	Yes	—	Top level parameters of the generated RTL file
dut_inst	Yes	—	Instantiated version of the <IP_name>.v file for simulation use
tb_top.v	Yes	—	Test bench template; you can edit this to match your specific needs.
<IP_name>.cfg	—	—	This file contains the configuration options used to recreate or modify the core in the IP Platform.
<IP_name>.ipx	—	—	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IP Platform user interface. The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP Platform when the IP/Module is being regenerated.

4.1. Generating a Multiplier on Default Settings

To generate a Multiplier on default settings:

1. First, create a new IP instance. Refer to the IP Generation section for details on how to generate an IP in a project.

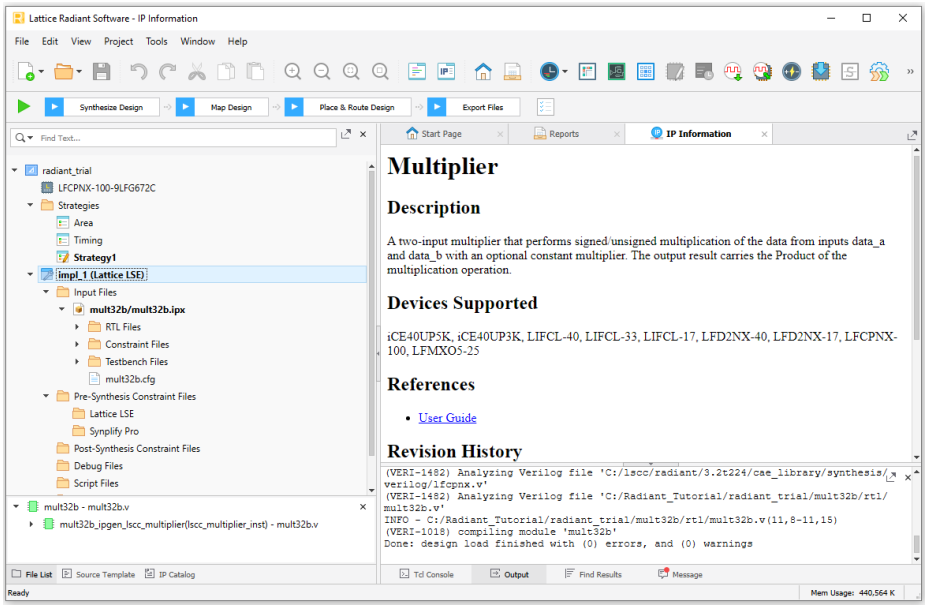

Figure 4.1. Project with an Instance of Multiplier

2. Modify the contents of the testbench to add a specific stimuli or check a specific option. For this example, however, it is left on the default RTL file. To simulate the project, click **Tools > Simulation Wizard**.
3. The **Simulation Wizard** window appears. Click **Next**. You are prompted to select the working folder and the test bench name. Enter **test**.

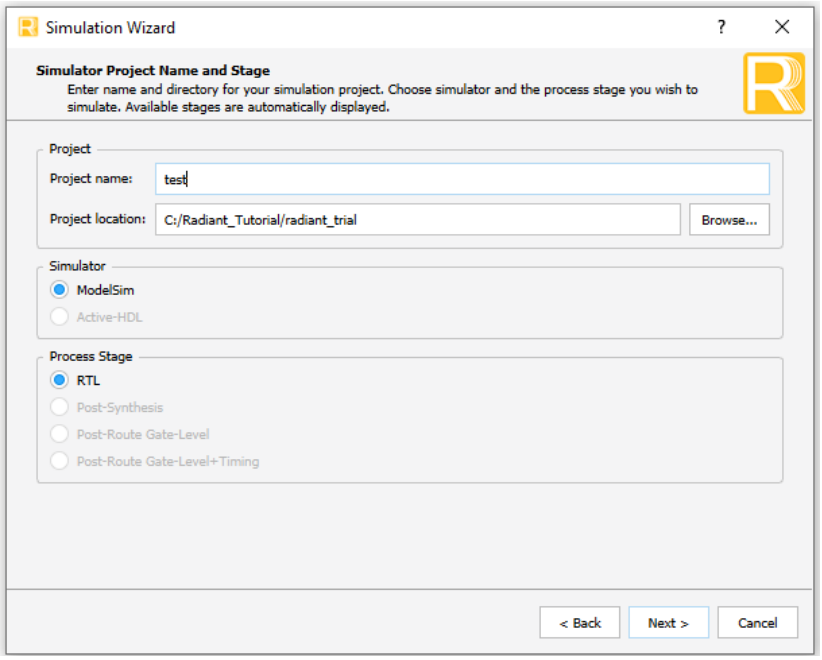

Figure 4.2. Simulation Wizard

4. Click **Next**.
5. In the prompt for Folder Creation, click **Yes**.
6. Moving to the processing stage, select **RTL**. Click **Next**.
7. The Source Files list appears. This shows the list of files included for simulation. Here you can add or subtract any other files you missed. Leave the file order for now. Click **Next**.

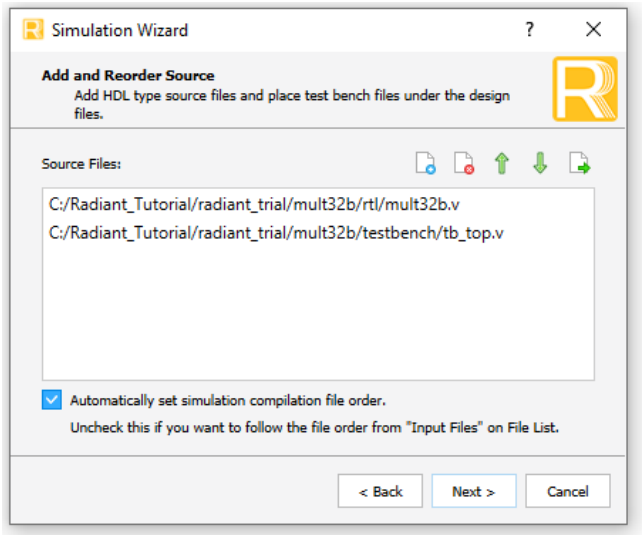

Figure 4.3. Final Simulation Files

8. In the next section, the files, as well as the other dependencies, are parsed to check for final errors. You can also select the top simulation module, before passing the files to Modelsim for execution. Click **Next** for a summary view.

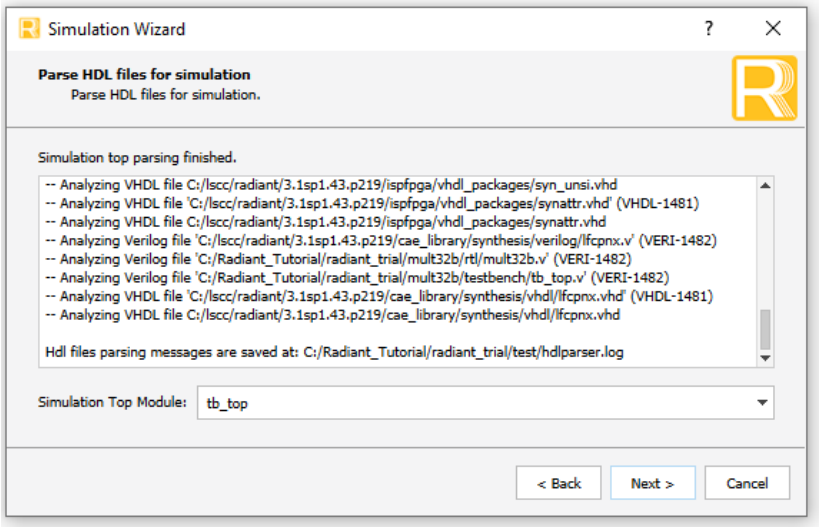

Figure 4.4. File Parsing and Top Module Selection

9. Click **Finish**. The Modelsim software opens, which automatically compiles and runs the RTL simulation.

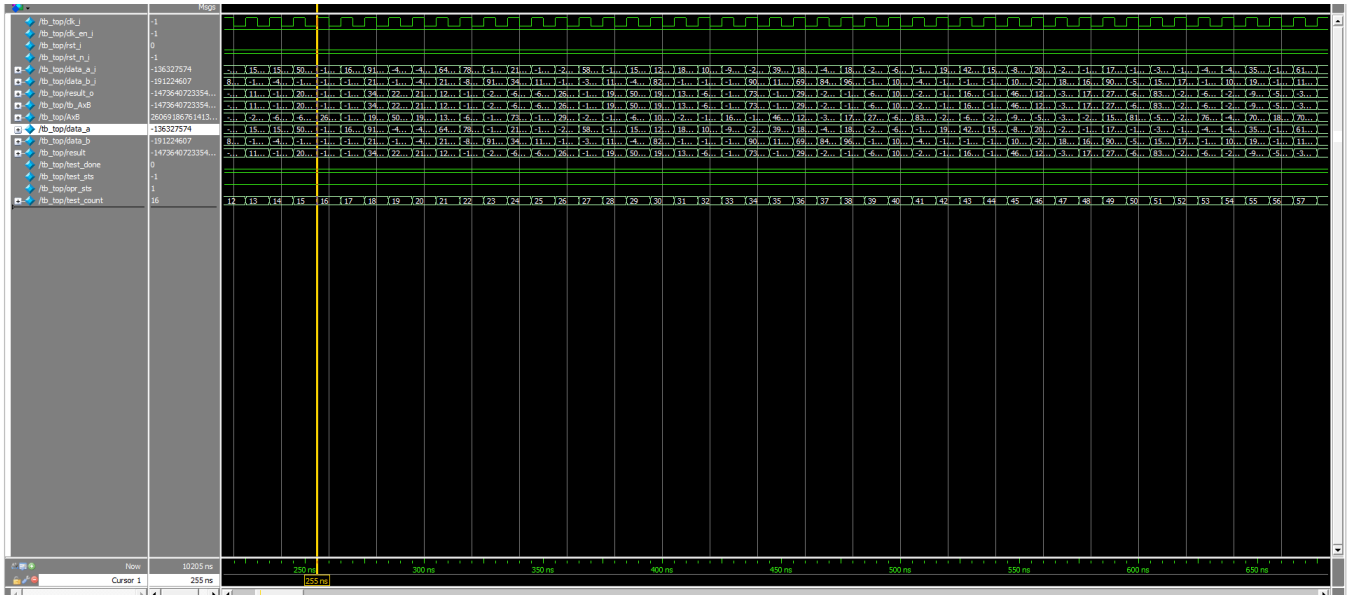

Figure 4.5. Modelsim Initial Simulation

When Modelsim opens, compiles the included RTL files and runs the simulation up to the first 100 ns. To view the rest of the simulation, click the **Play** button on top and zoom out for the overview of the signals.

5. Constraining the IP

You need to provide proper timing and physical design constraints to ensure that your design meets the desired performance goals on the FPGA. Add the content of the following IP constraint file to your design constraints: <Instance_Path>/<Instance_Name>/eval/constraint.pdc

The above constraint file has been verified during IP evaluation with the IP instantiated directly at the top-level module. You can modify the constraints in this file provided you understand the effect of each constraint.

To use this constraint file, copy the contents of constraint.pdc to the top-level design constrain for post-synthesis.

Refer to Lattice Radiant Timing Constraints Methodology (FPGA-AN-02059) for details on how to constrain your design.

6. 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 are used.

6.1. pmi_add

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

Table 6.1. 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 6.2. Attribute Definitions for pmi_add

Attributes	Description	Values	Default Value
pmi_data_width	Defines Bit width of DataA and DataB ports	2–64	8
pmi_sign	This configuration controls the interpretation of the inputs whether a signed or an unsigned number.	on, off	off
pmi_family	Defines the FPGA family being used in the module	common	common

Verilog Definition for `pmi_add`

```
module pmi_add
#(
    parameter pmi_data_width = 8,
    parameter pmi_sign       = "off",
    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
);
endmodule // pmi_add
```

VHDL Definition `pmi_add`

```
component pmi_add is
    generic(
        pmi_data_width : integer := 8;
        pmi_sign       : string  := "off";
        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;
```

6.2. pmi_sub

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

Table 6.3. 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 6.4. Attribute Definitions for pmi_sub

Attributes	Description	Values	Default Value
pmi_data_width	Defines Bit width of DataA and DataB ports	2-64	8
pmi_sign	Controls the interpretation of the inputs whether a signed or an unsigned number.	on, off	off
pmi_family	Defines the FPGA family being used in the module	common	common

Verilog Definition for `pmi_sub`

```
module pmi_sub
#(
    parameter pmi_data_width = 8,
    parameter pmi_sign       = "off",
    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
);
endmodule // pmi_sub
```

VHDL Definition `pmi_sub`

```
component pmi_sub is
    generic(
        pmi_data_width : integer := 8;
        pmi_sign       : string  := "off";
        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;
```

6.3. pmi_addsub

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

Table 6.5. Port Definitions for pmi_addsub

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
I	Add_Sub	Bit	N/A
O	Result	Bus	(pmi_data_width-1):0
O	Cout	Bit	N/A
O	Overflow	Bit	N/A

Table 6.6. Attribute Definitions for pmi_addsub

Attributes	Description	Values	Default Value
pmi_data_width	Defines Bit width of Data port	2-64	8
pmi_sign	Defines the sign of the input data	on, off	off
pmi_family	Defines the FPGA family being used in the module	common	common

Verilog Definition for pmi_addsub

```
module pmi_addsub
#(
      parameter                pmi_data_width   = 8,
      parameter                pmi_result_width = 8,
      parameter                pmi_sign         = "off",
      parameter                pmi_family       = "common",
      parameter                module_type      = "pmi_addsub"
)
(
      input                   [pmi_data_width-1:0] DataA,
      input                   [pmi_data_width-1:0] DataB,
      input                   Cin,
      input                   Add_Sub,
      output                 [pmi_data_width-1:0] Result,
      output                  Cout,
      output                  Overflow);

endmodule // pmi_addsub
```

VHDL Definition for pmi_addsub

```
component pmi_addsub is
    generic (
       pmi_data_width : integer := 8;
       pmi_result_width : integer := 8;
       pmi_sign : string := "off";
       pmi_family : string := "common";
       module_type : string := "pmi_addsub"
    );
    port (
     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;
     Add_Sub: 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_addsub;
```

6.4. pmi_counter

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

Table 6.7. Port Definitions for pmi_counter

Direction	Port Name	Type	Size (Buses Only)
I	UpDown	Bit	N/A
I	Clock	Bit	N/A
I	Clk_En	Bit	N/A
I	Aclr	Bit	N/A
O	Q	Bus	(pmi_data_width - 1):0

Table 6.8. Attribute Definitions for pmi_counter

Attributes	Description	Values	Default Value
pmi_data_width	Defines Bit width of the counter	1-64	8
pmi_updown	Controls the direction of the counter.	down up updown	up
pmi_family	Defines the FPGA family being used in the module	common	common

Verilog Definition for pmi_counter

```
module pmi_counter
#(
      parameter                     pmi_data_width     = 8,
      parameter                     pmi_updown         = "up",
      parameter                     pmi_family         = "common",
      parameter                     module_type        = "pmi_counter"
(
      input                         Clock,
      input                         Clk_En,
      input                         Aclr,
      input                         UpDown,
      output                        [pmi_data_width-1:0] Q)
      ;

endmodule // pmi_counter
```

VHDL Definition for pmi_counter

```
component pmi_counter is
    generic (
       pmi_data_width : integer := 8;
       pmi_updown : string := "up";
       pmi_family : string := "common";
       module_type : string := "pmi_counter"    );
    port (
     Clock: in std_logic;
     Clk_En: in std_logic;
     Aclr: in std_logic;
     UpDown: in std_logic;
     Q : out std_logic_vector(pmi_data_width-1 downto 0)
      );
end component pmi_counter;
```

6.5. pmi_mult

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

Table 6.9. 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_dataa_width + pmi_datab_width - 1):0

Table 6.10. Attribute Definitions for pmi_mult

Attributes	Description	Values	Default Value
pmi_dataa_width	Defines Bit width of DataA port	2-36	8
pmi_datab_width	Defines Bit width of DataB port	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 used in the module	common	common
pmi_implementation	Selects the resource to be used for implemention.	DSP, LUT	LUT

Verilog Definition for pmi_mult

```
module pmi_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_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
);
endmodule // pmi_mult
```

VHDL Definition for pmi_mult

```
component pmi_mult is
    generic (
        pmi_dataa_width         : integer := 8;
        pmi_datab_width         : integer := 8;
        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;
```

6.6. pmi_mac

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

Table 6.11. 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_dataa_width + pmi_datab_width - 1):0

Table 6.12. Attribute Definitions for pmi_mac

Attributes	Description	Values	Default Value
pmi_dataa_width	Defines Bit width of DataA port	2-36	8
pmi_datab_width	Defines Bit width of DataB port	2-36	8
pmi_accum_width	Accumulator width	(pmi_dataa_width + pmi_datab_width + 1) – (pmi_dataa_width + pmi_datab_width + 32)	32
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 perform on the accumulated value and multiplication product.	add, sub	add
pmi_input_reg	Enables input 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 used in the module	common	common
pmi_implementation	Selects the resource to be used for implemention.	DSP, LUT	LUT

Verilog Definition for pmi_mac

```
module pmi_mac
#(
    parameter pmi_dataa_width         = 8,
    parameter pmi_datab_width         = 8,
    parameter pmi_accum_width         = 32,
    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
);
endmodule // pmi_mac
```

VHDL Definition for pmi_mac

```
component pmi_mac is
    generic (
        pmi_dataa_width         : integer := 8;
        pmi_datab_width         : integer := 8;
        pmi_accum_width         : integer := 32;
        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;
```

6.7. pmi_multaddsub

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

Table 6.13. 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 6.14. Attribute Definitions for pmi_multaddsub

Attributes	Description	Values	Default Value
pmi_dataa_width	Defines Bit width of DataA0 and DataA1 ports	2-36	8
pmi_datab_width	Defines Bit width of DataB0 and DataB1 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_add_sub	Controls the operation to perform on the 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 used in the module	common	common
pmi_implementation	Selects the resource that to be used for implemention.	DSP, LUT	LUT

Verilog Definition for pmi_multaddsub

```
module pmi_multaddsub
#(
    parameter pmi_dataa_width         = 8,
    parameter pmi_datab_width         = 8,
    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
);
endmodule // pmi_multaddsub
```

VHDL Definition for pmi_multaddsub

```
component pmi_multaddsub is
    generic (
        pmi_dataa_width         : integer := 8;
        pmi_datab_width         : integer := 8;
        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;
```

6.8. pmi_multaddsubsum

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

Table 6.15. Port Definitions for pmi_multaddsubsum

Direction	Port Name	Type	Size (Buses Only)
I	DataA0	Bus	(pmi_dataa_width-1):0
I	DataA1	Bus	(pmi_dataa_width-1):0
I	DataA2	Bus	(pmi_dataa_width-1):0
I	DataA3	Bus	(pmi_dataa_width-1):0
I	DataB0	Bus	(pmi_datab_width-1):0
I	DataB1	Bus	(pmi_datab_width-1):0
I	DataB2	Bus	(pmi_datab_width-1):0
I	DataB3	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 + 1):0

Table 6.16. Attribute Definitions for pmi_multaddsubsum

Attributes	Description	Values	Default Value
pmi_add_sub0	Controls the operation to be performed on the multiplication product.	add, sub	add
pmi_add_sub1	Controls the operation to be performed on the multiplication product.	add, sub	add
pmi_dataa_width	Defines Bit width of DataA0, DataA1, DataA2 and DataA3 ports	2-64	8
pmi_datab_width	Defines Bit width of DataB0, DataB1, DataB2 and DataB3 ports	2-64	8
pmi_sign	Controls the interpretation of the inputs DataA0, DataA1, DataA2, DataA3, DataB0, DataB1, DataB2 and DataB3 whether a signed or an unsigned number.	on, off	on
pmi_additional_pipeline	Specifies the number of pipeline stages.	(pmi_dataa_width> pmi_datab_width)? 0- pmi_dataa_width : 0 - pmi_datab_width	1
pmi_input_reg	Enables input register	on, off	on
pmi_output_reg	Enables output register	on, off	on
pmi_family	Defines the FPGA family being used in the module	common	common
pmi_implementation	Selects the resource to be used for implemention.	DSP, LUT	LUT

Verilog Definition for pmi_multaddsubsum

```
module pmi_multaddsubsum
#(
      parameter pmi_dataa_width           = 8,
      parameter pmi_datab_width           = 8,
      parameter module_type               = "pmi_multaddsubsum",
      parameter pmi_sign                  = "on",
      parameter pmi_additional_pipeline   = 1,
      parameter pmi_add_sub0              = "add",
      parameter pmi_add_sub1              = "add",
      parameter pmi_input_reg             = "on",
      parameter pmi_output_reg            = "on",
      parameter pmi_family                = "common",
      parameter pmi_implementation        = "LUT"
)
(     input   [(pmi_dataa_width-1):0]  DataA0,
      input   [(pmi_dataa_width-1):0]  DataA1,
      input   [(pmi_dataa_width-1):0]  DataA2,
      input   [(pmi_dataa_width-1):0]  DataA3,
         input   [(pmi_datab_width-1):0]  DataB0,
         input   [(pmi_datab_width-1):0]  DataB1,
         input   [(pmi_datab_width-1):0]  DataB2,
         input   [(pmi_datab_width-1):0]  DataB3,
         input   Clock, ClkEn, Aclr,
         output  [(pmi_dataa_width + pmi_datab_width + 1):0]  Result)
;

endmodule // pmi_multaddsubsum
```

VHDL Definition for pmi_multaddsubsum

```
component pmi_multaddsubsum is
    generic (
       pmi_dataa_width         : integer := 8;
       pmi_datab_width         : integer := 8;
       module_type             : string  := "pmi_multaddsubsum";
       pmi_sign                : string  := "on";
       pmi_additional_pipeline : integer := 1;
       pmi_add_sub0            : string  := "add";
       pmi_add_sub1            : string  := "add";
       pmi_input_reg           : string  := "on";
       pmi_output_reg          : string  := "on";
       pmi_family              : string  := "common";
       pmi_implementation      : string  := "LUT"
   );
    port (
     DataA0 : in std_logic_vector((pmi_dataa_width-1) downto 0);
     DataA1 : in std_logic_vector((pmi_dataa_width-1) downto 0);
     DataA2 : in std_logic_vector((pmi_dataa_width-1) downto 0);
     DataA3 : 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);
     DataB2 : in std_logic_vector((pmi_datab_width-1) downto 0);
     DataB3 : in std_logic_vector((pmi_datab_width-1) downto 0);
     Clock, ClkEn, Aclr : in std_logic;
     Result : out std_logic_vector((pmi_dataa_width + pmi_datab_width + 1) downto
0)
   );
end component pmi_multaddsubsum;
```

6.9. pmi_complex_mult

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

Table 6.17. 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 6.18. 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 to be used for implemention	DSP, LUT	LUT

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]  DataA_Re,
    input   [(pmi_dataa_width-1):0]  DataA_Im,
    input   [(pmi_datab_width-1):0]  DataB_Re,
    input   [(pmi_datab_width-1):0]  DataB_Im,
    output  [(pmi_dataa_width+pmi_datab_width):0]  Result_Re,
    output  [(pmi_dataa_width+pmi_datab_width):0]  Result_Im
);
endmodule // pmi_complex_mult
```

VHDL Definition for pmi_complex_mult

```
component pmi_complex_mult is
    generic (
        pmi_dataa_width         : integer := 8;
        pmi_datab_width         : integer := 8;
        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;
        DataA_Re: in std_logic_vector((pmi_dataa_width-1) downto 0);
        DataA_Im: in std_logic_vector((pmi_dataa_width-1) downto 0);
        DataB_Re: in std_logic_vector((pmi_datab_width-1) downto 0);
        DataB_Im: in std_logic_vector((pmi_datab_width-1) downto 0);
        Result_Re: out std_logic_vector((pmi_dataa_width+pmi_datab_width) downto
0);
        Result_Im: out std_logic_vector((pmi_dataa_width+pmi_datab_width) downto 0)
    );
end component pmi_complex_mult;
```

6.10. 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 iCE40 UltraPlus devices. The following are port descriptions, attribute descriptions, Verilog definition and VHDL definition for pmi_dsp.

Table 6.19. Port Definitions for pmi_dsp

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
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 6.20. Attribute Definitions for pmi_dsp

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
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, 0b01, 0b10, 0b11	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, 0b01, 0b10, 0b11	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, 0b01, 0b10, 0b11	0b00
BOTOUTPUT_SELECT	Selects lower output O[15:0] 0b00 = bottom adder/subtractor 0b01 = bottom adder/subtractor (registered), 0b10 = bottom mult8x8 0b11 = mult16x16	0b00, 0b01, 0b10, 0b11	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, 0b01, 0b10, 0b11	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, 0b01, 0b10, 0b11	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;
```

Appendix A. Resource Utilization

This appendix provides for Lattice FPGAs using the Arithmetic Modules. The IP configurations shown in this chapter were generated using the Lattice Radiant tool.

A.1. Adder

Table A.1. Adder Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Adder	18	0	0	0
16-bit Signed Complex Adder	36	0	0	0
16-bit Signed with Carry-In Adder with output register	18	16	0	0
60-bit Signed Adder	62	0	0	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.2. Adder Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Adder	18	0	0	0
16-bit Signed Complex Adder	36	0	0	0
16-bit Signed with Carry-In Adder with output register	18	16	0	0
60-bit Signed Adder	62	0	0	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX-100 family or in a different software version.

Table A.3. Adder Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Adder	16	0	0	0
16-bit Signed Complex Adder	32	0	0	0
16-bit Signed with Carry-In Adder with output register	16	16	0	0
60-bit Signed Adder	60	0	0	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.2. Subtractor

Table A.4. Subtractor Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Subtractor	18	0	0	0
16-bit Signed Complex Subtractor	36	0	0	0
16-bit Signed with Carry-In Subtractor with output register	54	16	0	0
60-bit Signed Subtractor	62	0	0	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.5. Subtractor Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Subtractor	18	0	0	0
16-bit Signed Complex Subtractor	36	0	0	0
16-bit Signed with Carry-In Subtractor with output register	54	16	0	0
60-bit Signed Subtractor	62	0	0	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.6. Subtractor Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Subtractor	16	0	0	0
16-bit Signed Complex Subtractor	32	0	0	0
16-bit Signed with Carry-In Subtractor with output register	48	16	0	0
60-bit Signed Subtractor	60	0	0	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.3. Adder-Subtractor

Table A.7. Adder-Subtractor Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Adder-Subtractor	18	0	0	0
16-bit Signed Complex Adder-Subtractor	36	0	0	0
16-bit Signed with Carry-In Adder-Subtractor	52	0	0	0
16-bit Signed with Carry-In and Overflow Adder-Subtractor	56	0	0	0
16-bit Signed with Carry-In Adder-Subtractor with output register	52	16	0	0
60-bit Signed Adder-Subtractor	62	0	0	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.8. Adder-Subtractor Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Adder-Subtractor	18	0	0	0
16-bit Signed Complex Adder-Subtractor	36	0	0	0
16-bit Signed with Carry-In Adder-Subtractor	52	0	0	0
16-bit Signed with Carry-In and Overflow Adder-Subtractor	56	0	0	0
16-bit Signed with Carry-In Adder-Subtractor with output register	52	16	0	0
60-bit Signed Adder-Subtractor	62	0	0	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.9. Adder - Subtractor Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
16-bit Unsigned Adder -Subtractor	18	0	0	0
16-bit Signed Complex Adder - Subtractor	35	0	0	0
16-bit Signed with Carry-In Adder-Subtractor	49	0	0	0
16-bit Singed with Carry-In and Overflow Adder-Subtractor	83	0	0	0
16-bit Signed with Carry-In Adder-Subtractor with output register	49	16	0	0
60-bit Signed Adder-Subtractor	62	0	0	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.4. Comparator

Table A.10. Comparator Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Signed “not equal” Comparator	5	0	0	0
8-bit “greater than” Comparator	13	0	0	0
8-bit “not equal” with output register Comparator	5	1	0	0
60-bit “not equal” Comparison	63	0	0	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.11. Comparator Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Signed “not equal” Comparator	5	0	0	0
8-bit “greater than” Comparator	13	0	0	0
8-bit “not equal” with output register Comparator	5	1	0	0
60-bit “not equal” Comparison	63	0	0	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.12. Comparator Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Signed “not equal” Comparator	5	0	0	0
8-bit “greater than” Comparator	13	0	0	0
8-bit “not equal” with output register Comparator	5	1	0	0
60-bit “not equal” Comparison	63	0	0	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.5. Counter

Table A.13. Counter Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Up Counter	13	8	0	0
8-bit UpDown Counter	24	8	0	0
8-bit Up Counter with enabled load	21	8	0	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.14. Counter Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Up Counter	13	8	0	0
8-bit UpDown Counter	24	8	0	0
8-bit Up Counter with enabled load	21	8	0	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.15. Counter Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Up Counter	19	8	0	0
8-bit UpDown Counter	24	8	0	0
8-bit Up Counter with enabled load	34	8	0	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.6. Multiplier

Table A.16. Multiplier Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed Multiplier with input register using LUT blocks	242	54	0	0
9-bit Signed Multiplier with input register using DSP blocks	0	18	0.5	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.17. Multiplier Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed Multiplier with input register using LUT blocks	242	54	0	0
9-bit Signed Multiplier with input register using DSP blocks	0	18	0.5	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.18. Multiplier Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed Multiplier with input register using LUT blocks	193	54	0	0
9-bit Signed Multiplier with input register using DSP blocks	0	18	1	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.7. Multiply Accumulate

Table A.19. Multipliy Accumulate Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
18-bit Signed Mult_Acc (Addition) using LUT blocks	625	73	0	0
18-bit Signed Mult_Acc (Addition) using DSP blocks	39	37	1	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.20. Multipliy Accumulate Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
18-bit Signed Mult_Acc (Addition) using LUT blocks	625	73	0	0
18-bit Signed Mult_Acc (Addition) using DSP blocks	39	37	1	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.21. Multiply Accumulate Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
18-bit Signed Mult_Acc (Addition) using LUT blocks	494	73	0	0
18-bit Signed Mult_Acc (Addition) using DSP blocks	0	0	1	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.8. Mult-Add-Sub

Table A.22. Mult Add Sub Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed Mult-Add-Sub (Subtraction) using LUT blocks	499	91	0	0
9-bit Signed Mult-Add-Sub (Subtraction) using DSP blocks	20	19	1	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.23. Mult Add Sub Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed Mult-Add-Sub (Subtraction) using LUT blocks	499	91	0	0
9-bit Signed Mult-Add-Sub (Subtraction) using DSP blocks	20	19	1	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.24. Mult Add Sub Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed Mult-Add-Sub (Subtraction) using LUT blocks	403	91	0	0
9-bit Signed Mult-Add-Sub (Subtraction) using DSP blocks	0	0	1	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.9. Mult-Add-Sub-Sum

Table A.25. Mult Add Sub Sum Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed/Unsigned Mult_add_Sub_Sum using LUT blocks	1020	164	0	0
9-bit Signed/Unsigned Mult_add_Sub_Sum using DSP blocks	62	20	2	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.26. Mult Add Sub Sum Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed/Unsigned Mult_add_Sub_Sum using LUT blocks	1020	164	0	0
9-bit Signed/Unsigned Mult_add_Sub_Sum using DSP blocks	62	20	2	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -40 family or in a different software version.

Table A.27. Mult Add Sub Sum Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
9-bit Signed/Unsigned Mult_Add_Sub_Sum using LUT blocks	834	164	0	0
9-bit Signed/Unsigned Mult_Add_Sub_Sum using DSP blocks	0	0	1	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.10. Complex Multiplier

Table A.28. Complex Multiplier Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Unsigned Complex Multiplier without input register using LUT blocks	656	130	0	0
8-bit Unsigned Complex Multiplier using DSP blocks	37	34	2	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.29. Complex Multiplier Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Unsigned Complex Multiplier without input register using LUT blocks	656	130	0	0
8-bit Unsigned Complex Multiplier using DSP blocks	37	34	2	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.30. Complex Multiplier Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Unsigned Complex Multiplier without input register using LUT blocks	557	98	0	0
8-bit Unsigned Complex Multiplier using DSP blocks	18	34	2	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.11. LFSR

Table A.31. LFSR Resource Utilization (LIFCL)

Configura2ion	LUTs	Registers	DSP	EBRs
8-bit Galois-type LFSR with XOR gate	1	8	0	0
16-bit Fibonacci-type LFSR with XOR gate	1	16	0	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.32. LFSR Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Galois-type LFSR with XOR gate	1	8	0	0
16-bit Fibonacci-type LFSR with XOR gate	1	16	0	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.33. LFSR Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
8-bit Galois-type LFSR with XOR gate	1	8	0	0
16-bit Fibonacci-type LFSR with XOR gate	1	16	0	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.12. Convert

Table A.34. Convert Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
16-bit to 8-bit Truncate Converter	5	0	0	0
8-bit to 8-bit Nearest Converter	1	0	0	0

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.35. Convert Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
16-bit to 8-bit Truncate Converter	5	0	0	0
8-bit to 8-bit Nearest Converter	1	0	0	0

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.36. Convert Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
16-bit to 8-bit Truncate Converter	8	0	0	0
8-bit to 8-bit Nearest Converter	0	0	0	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

A.13. Sin-Cos Table

Table A.37. Sin-Cos Table Resource Utilization (LIFCL)

Configuration	LUTs	Registers	DSP	EBRs
8-bit SIN table (Quarterwave) using LUT blocks	76	16	0	1
8-bit SIN-COS table (Fullwave) using EBR blocks	1	8	0	1

Note: Performance and utilization characteristics are generated targeting an LIFCL-40-9BG400I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LIFCL-40 family or in a different software version.

Table A.38. Sin-Cos Table Resource Utilization (LFCPNX)

Configuration	LUTs	Registers	DSP	EBRs
8-bit SIN table (Quarterwave) using LUT blocks	76	16	0	1
8-bit SIN-COS table (Fullwave) using EBR blocks	1	8	0	1

Note: Performance and utilization characteristics are generated targeting an LFCPNX-100-9LFG672I device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LFCPNX -100 family or in a different software version.

Table A.39. Sin-Cos Table Resource Utilization (LAV-AT)

Configuration	LUTs	Registers	DSP	EBRs
8-bit SIN table (Quarterwave) using LUT blocks	97	16	0	0
8-bit SIN-COS table (Fullwave) using EBR blocks	0	16	0	0

Note: Performance and utilization characteristics are generated targeting an LAV-AT-E70-1LFG1156C device using Lattice Radiant Software and LSE Synthesis Tool. Performance may vary when using this IP in a different density, speed or grade within the Lattice LAV-AT family or in a different software version.

References

- For complete information on the Lattice Radiant Project-Based Environment, Design Flow, Implementation Flow and Tasks, as well as on the Simulation Flow, refer to the Lattice Radiant software user guide.
- Lattice Radiant Timing Constraints Methodology (FPGA-AN-02059)
- Avant-E web page
- CrossLink-NX web page
- CertusPro-NX web page
- Lattice Radiant Software web page
- Lattice Insights for Lattice Semiconductor training courses and learning plan.

Technical Support Assistance

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

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/en/Support/AnswerDatabase.

Revision History

Document Revision 2.5, Lattice Radiant SW Version 2023.2, November 2023

Section	Change Summary
Disclaimers	Updated with the latest disclaimers.
IP Generation	Updated Table 2.2. Adder Attributes and Table 2.4. Subtractor Attributes to remove 'O_WDT'.
Constraining the IP	Newly added section.
PMI Support:	• Updated Table 6.2. Attribute Definitions for pmi_add, Table 6.4. Attribute Definitions for pmi_sub, Table 6.8. Attribute Definitions for pmi_counter, Table 6.10. Attribute Definitions for pmi_mult, Table 6.12. Attribute Definitions for pmi_mac, Table 6.14. Attribute Definitions for pmi_multaddsub, Table 6.16. Attribute Definitions for pmi_multaddsubsum, and Table 6.18. Attribute Definitions for pmi_complex_mult to remove 'module_type'. • Updated Table 6.6. Attribute Definitions for pmi_addsub to remove 'pmi_result_width' and 'module_type'.
Appendix A. Resource Utilization	Updated this section to add resource utilization information for LAV-AT family.
References	Newly added section.
Technical Support Assistance	Added link to the Lattice Answer Database.

Document Revision 2.4, Lattice Radiant SW Version 3.2, May 2022

Section	Change Summary
Arithmetic Modules	• Updated Table 2.26. Sin-Cos Table Attributes. Added the O_WIDTH_DH_SC, DWID rows. • Updated Table 2.24. Convert Attributes. Removed OUPUTWIDTH, OUTPUTPOINT, INPUTFORMATUNSIGNED, ROUND and WRAP rows.
IP Generation	Updated Figure 3.1. Arithmetic Modules under Module/IP on Local in the Lattice Radiant Software, Figure 3.2. Example: Generating 32-bit Multiplier Using Module/IP Block Wizard and Figure 3.3. Example: Generating 32-bit Multiplier in IP Configuration.
Simulation Flow	Updated Section 4.1 Generating a Multiplier on Default Settings.

Document Revision 2.3, Lattice Radiant SW Version 3.1, November 2021

Section	Change Summary
Appendix A. Resource Utilization	Updated section content to add tables for LFCPNX.

Document Revision 2.2, Lattice Radiant SW Version 2.1, June 2020

Section	Change Summary
Arithmetic Modules	• Added support for Certus™-NX family. • Updated Figure 2.21 and Figure 2.22 in LFSR section.

Document Revision 2.1, Lattice Radiant SW Version 2.0, November 2019

Section	Change Summary
Appendix A – Resource Utilization	Updated this section.

Document Revision 2.0, Lattice Radiant SW Version 2.0, September 2019

Section	Change Summary
—	Appended Lattice Radiant Software to document title.
Disclaimers	Added this section.

Arithmetic Modules	• Added support for CrossLink-NX family.
	• Added the following Arithmetic Modules:
	  • Convert
	  • Sin-Cos Table

Appendix A – Resource Utilization	• Added this section.

Document Revision 1.1, Lattice Radiant SW Version 1.1, December 2018

Section	Change Summary

Arithmetic Modules	• Updated the following Arithmetic Modules:
	  • Adder – Removed register input option
	  • Complex Mult – Updated pipeline value range: implementation-dependent
	  • Mult Accumulate – Updated result_o width to match the result width of pmi_mac from the pmi Verilog library
	  • Mult_Add_Sub – Updated pipeline value range: implementation-dependent
	  • Multiplier – Updated pipeline value range: implementation-dependent
	  • Subtractor – Removed register input option
	• Added the following Arithmetic Modules:
	  • Adder Subtractor
	  • Comparator
	  • Counter
	  • LFSR (Linear Feedback Shift Register)
	  • Mult_Add_Sub_Sum
	• Added Simulation Flow section.
	• Initial availability of testbench for all modules.

Document Revision 1.0, Lattice Radiant SW Version 1.0, January 2018

Section	Change Summary
All	Initial release

www.latticesemi.com