Lattice Avant sysDSP User Guide

Preliminary **Technical Note**

FPGA-TN-02293-0.85

February 2025

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.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
CNN	Convolutional Neural Network
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
GEMM	General Matrix Multiplication
HDL	Hardware Description Language
IIR	Infinite Impulse Response
MSB	Most Significant Bit
RRH	Remote Radio Head
RTI	Round to Infinity
RTZ	Round to Zero
VHDL	Very-High-Speed Integrated Circuit Hardware Description Language

1. Introduction

This technical note discusses how to access the features of the Lattice sysDSP™ (Digital Signal Processing) block for the Lattice Avant™ platform.

Lattice Avant devices are optimized to support high-performance DSP applications, such as wireless base station channel cards, Remote Radio Head (RRH) systems, video and imaging applications, and Fast Fourier Transform (FFT) functions. These applications can be easily built by cascading multiple sysDSP instances with minimal fabric or routing resources to form the core of those operations, such as different FIR filters, 9 bits or 18 bits precision dot product operation which are widely used in general matrix multiplication (GEMM) and convolutional neural network (CNN), as well as high precision floating point multiplier.

2. sysDSP Overview

Figure 2.1 shows the sysDSP block diagram. Lattice Avant sysDSP block consists of pre-adder, multipliers, pipeline registers and wide accumulator with saturation. This multiplier can be configured to different modes, 8 × 8 multiplier, 9 × 9 multiplier, 18 × 18 multiplier, and 9 × 9 dot product. The sysDSP block contains an 18-bit pre-adder that works with the 18 × 18 multiplier mode, and a 48-bit adder/accumulator, to support multiply-add or multiply-accumulate operation. The input, output, and pipeline registers in the sysDSP block can be bypassed.

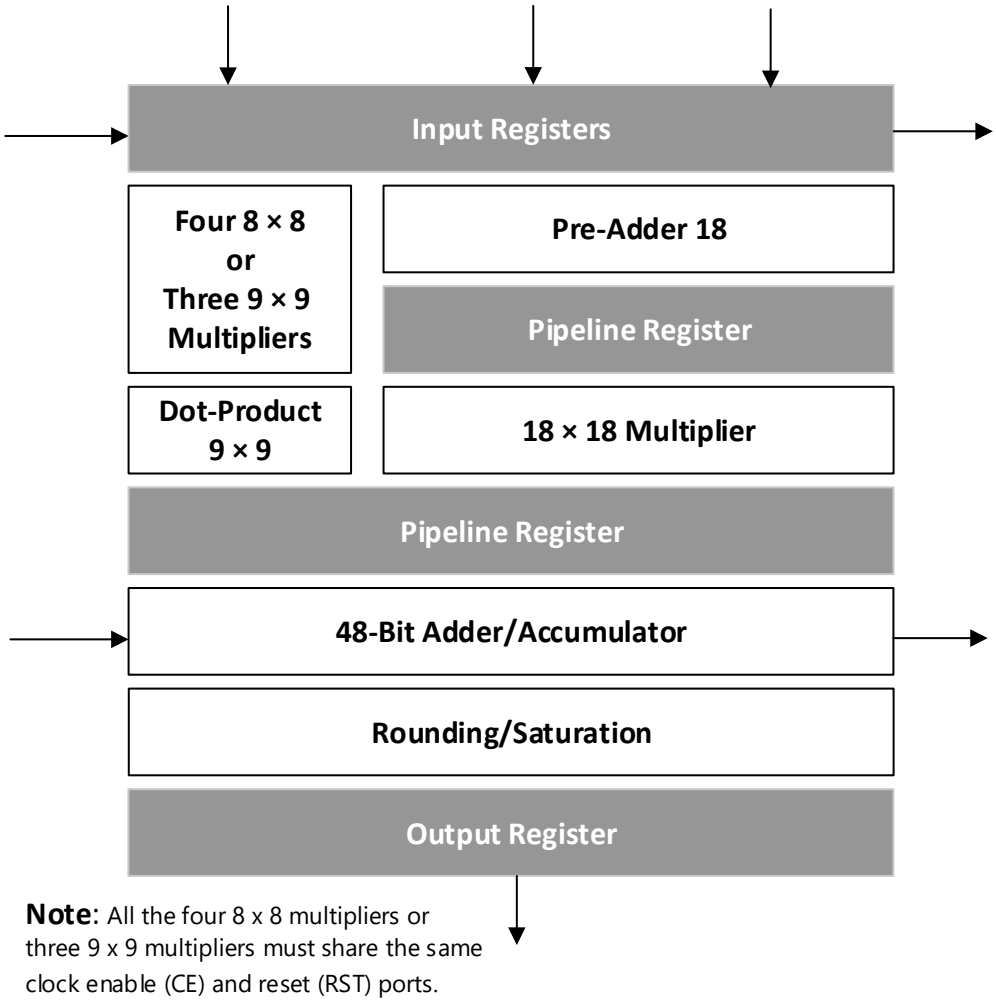

Figure 2.1. sysDSP Block Diagram

Lattice Avant sysDSP block has strong multiplication support and dedicated architectural features that enable efficient implementation of machine learning, wearable, mobile, video, and other DSP-intensive applications.

The architectural features include:

- Dedicated input shift registers, bypassable pipeline registers, and dedicated cascade connections to implement multi-tap FIR, matrix dot product, and higher precision multiplier.
- 2-depth input shift registers with separate clock enable (CE) enable one register to perform as shift register and the other as update register.
- Configured multipliers, which can implement 8 × 8, 9 × 9, or 18 × 18 multiplications natively.
- Rounding support with rounding constant from operand C input that enables rounding point anywhere in the 48-bit accumulator output.
- Configurable width (N=3 to 48 bits) overflow and saturation logic.

3. Targeting the sysDSP Block by Instantiating Primitives

The sysDSP block can be targeted by instantiating the sysDSP primitive into a design. The advantage of instantiating primitives is that it provides access to all the available ports and parameters. The disadvantage of this flow is that the customization requires extra coding and knowledge. This section details the primitives supported by Lattice Avant devices. See the Instantiating sysDSP Primitives in HDL section that shows an HDL example on how to instantiate sysDSP primitives.

Lattice Avant sysDSP supports primitives, such as MULT8×8, MULT9×9A, MULT18×18A, MULTADDSUB18×18A, and DOTPRODADDSUB9×9(A). In addition. Several other library primitives are defined to take advantage of the features of Lattice Avant sysDSP.

Following sections discuss various primitives along with the port definitions and attributes of the primitives.

3.1. MULT8×8 – 8×8 Multiplier with Optional Input/Output Registers

MULT8×8 implements the function Z = A * B. Figure 3.1 shows the MULT8×8 primitive available in Lattice Avant devices. The functional diagram of the multiplier is shown in Figure 3.2.

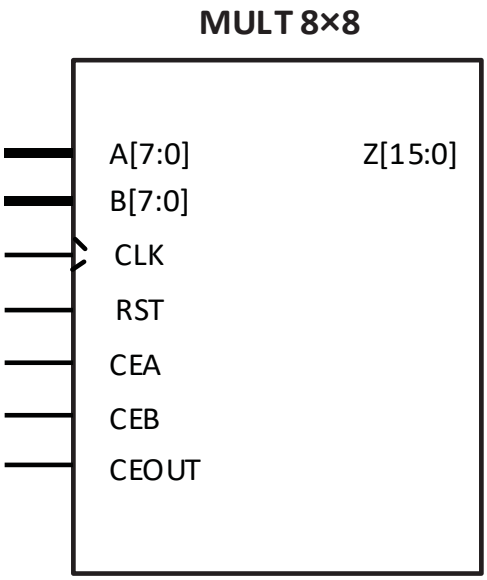

Figure 3.1. MULT8×8 Primitive

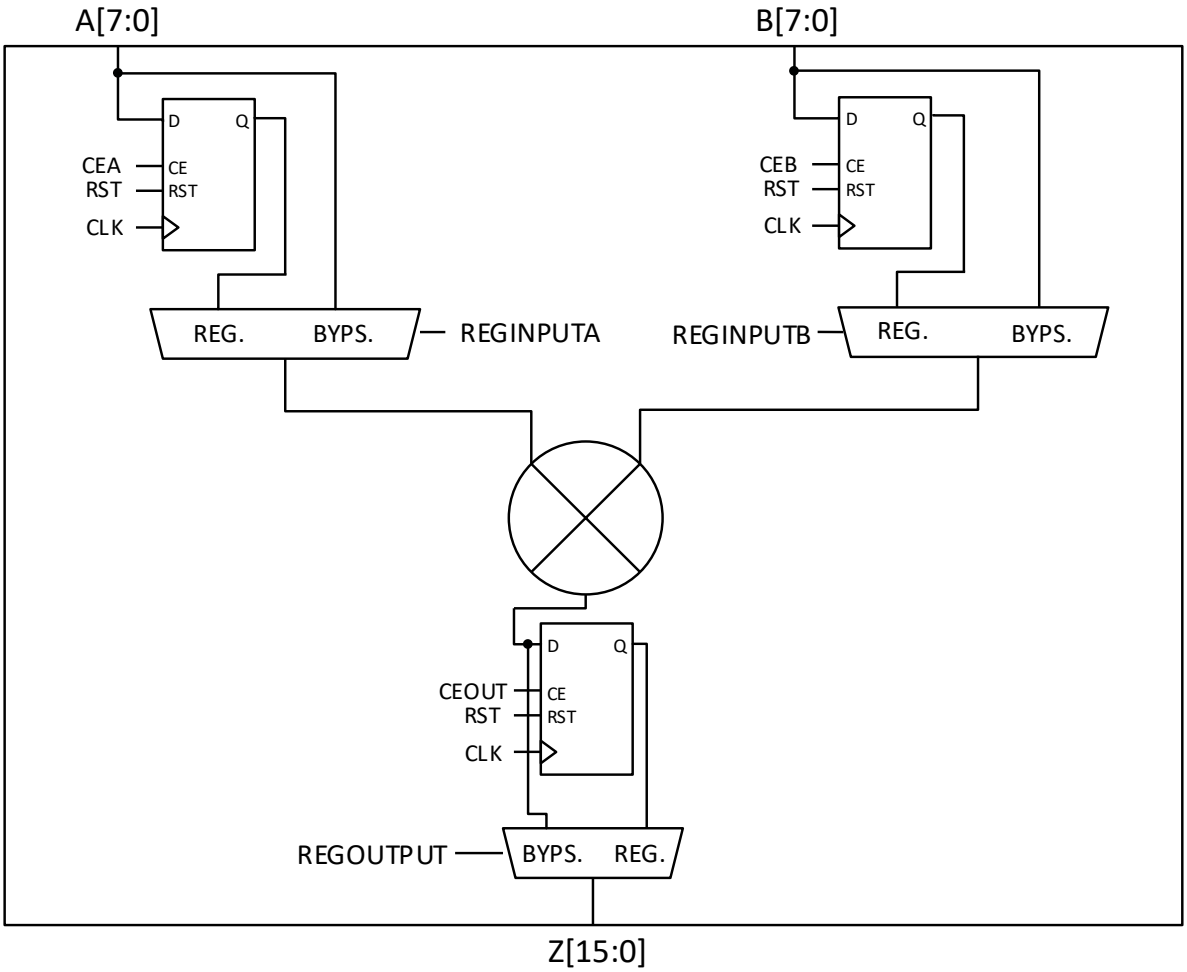

Figure 3.2. MULT8×8 Functional Diagram

3.1.1. MULT8×8 – I/O Ports

Table 3.1 describes the ports available for the primitive.

Table 3.1. MULT8×8 I/O Ports

Port	Input/Output	Description
A[7:0]	Input	Operand A (signed)
B[7:0]	Input	Operand B (signed)
CLK	Input	Clock
RST	Input	Reset
CEA	Input	Operand A register clock enable
CEB	Input	Operand B register clock enable
CEOUT	Input	Output register clock enable
Z[15:0]	Output	Multiplication result

3.1.2. MULT8×8 – Attributes

Table 3.2 describes the attributes for the primitive.

Table 3.2. MULT8×8 Attributes

Attribute Name	Values	Default Value	Description
REGINPUTA	REGISTERED, BYPASSED	REGISTERED	Operand A register
REGINPUTB	REGISTERED, BYPASSED	REGISTERED	Operand B register
REGOUTPUT	REGISTERED, BYPASSED	REGISTERED	Output register
GSR	ENABLED, DISABLED	DISABLED	Enable global set/reset
RESETMODE	SYNC, ASYNC	SYNC	Synchronous/asynchronous reset control.

3.2. MULT9X9A – 9×9 Multiplier with Optional Input/Output Registers

The 9 × 9 multiplier implements the function Z = A * B. Figure 3.3 shows the MULT9×9A primitive available in Lattice Avant devices. The functional diagram of the multiplier is shown in Figure 3.4.

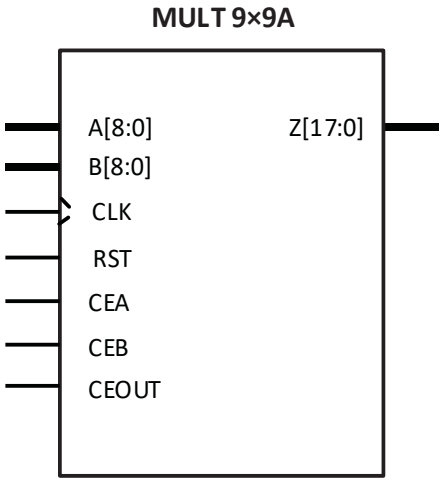

Figure 3.3. MULT9×9A Primitive

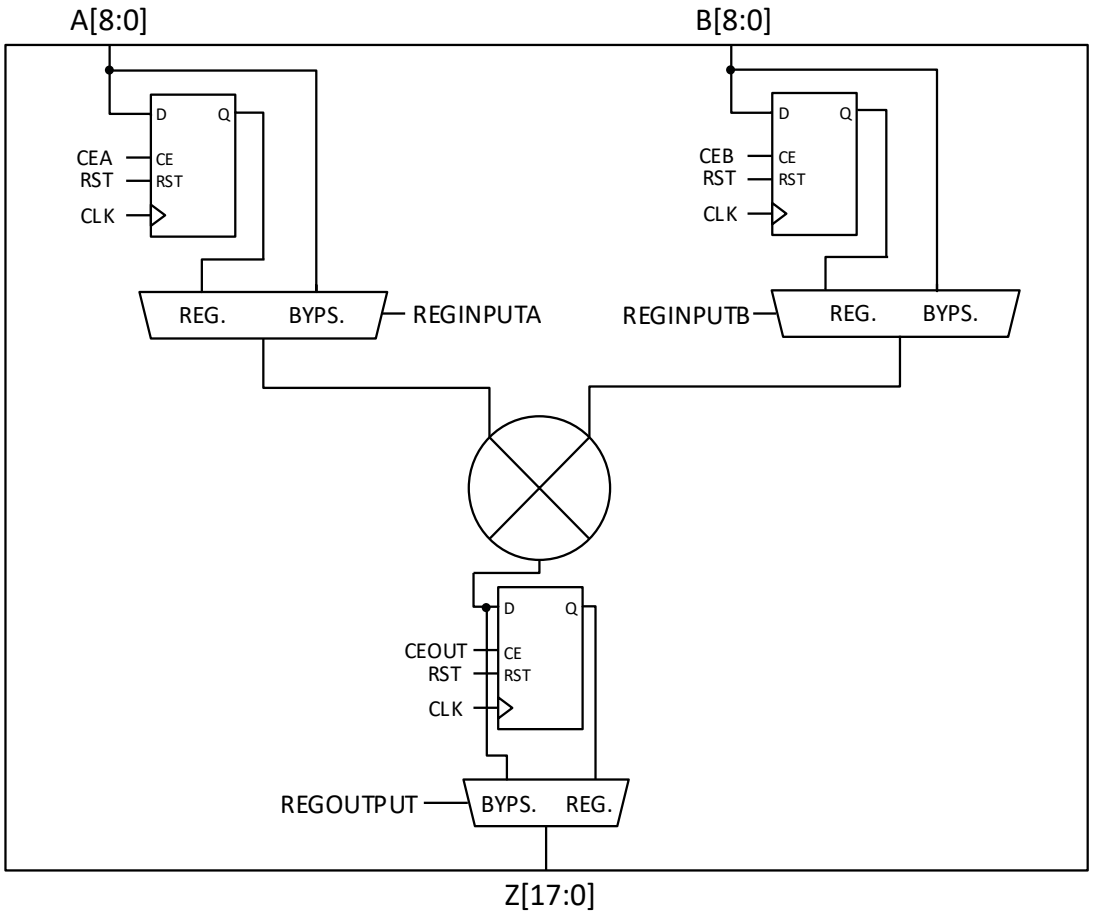

Figure 3.4. MULT9×9A Functional Diagram

3.2.1. MULT9×9A – I/O Ports

Table 3.3 describes the ports available for the primitive.

Table 3.3. MULT9×9A I/O Ports

Port	Input/Output	Description
A[8:0]	Input	Operand A (signed)
B[8:0]	Input	Operand B (signed)
CLK	Input	Clock
RST	Input	Synchronous reset for all registers
CEA	Input	Operand A register clock enable
CEB	Input	Operand B register clock enable
CEOUT	Input	Output register clock enable
Z[17:0]	Output	Multiplication result

3.2.2. MULT9×9A – Attributes

Table 3.4 describes the attributes for the primitive.

Table 3.4. MULT9X9A Attributes

Attribute Name	Values	Default Value	Description
REGINPUTA	REGISTERED, BYPASSED	REGISTERED	Operand A register
REGINPUTB	REGISTERED, BYPASSED	REGISTERED	Operand B register
REGOUTPUT	REGISTERED, BYPASSED	REGISTERED	Output register
GSR	ENABLED, DISABLED	DISABLED	Enable global set/reset
RESETMODE	SYNC, ASYNC	SYNC	Synchronous/asynchronous reset control

3.3. MULT18×18A – 18×18 Multiplier with Optional Input/Output Registers

The 18 × 18 multiplier implements the function Z = A * B. Figure 3.5 shows the MULT18X18A primitive available in Lattice Avant devices. The functional diagram of the multiplier is shown in Figure 3.6.

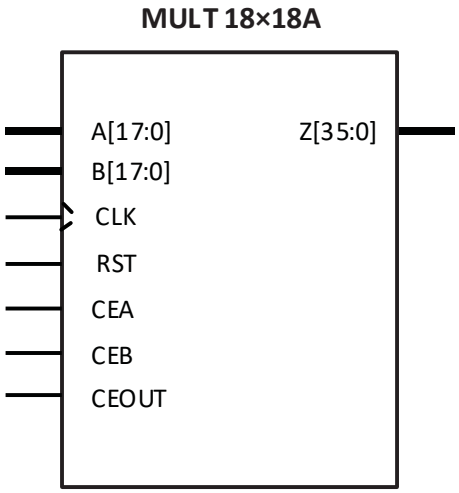

Figure 3.5. MULT18×18A Primitive

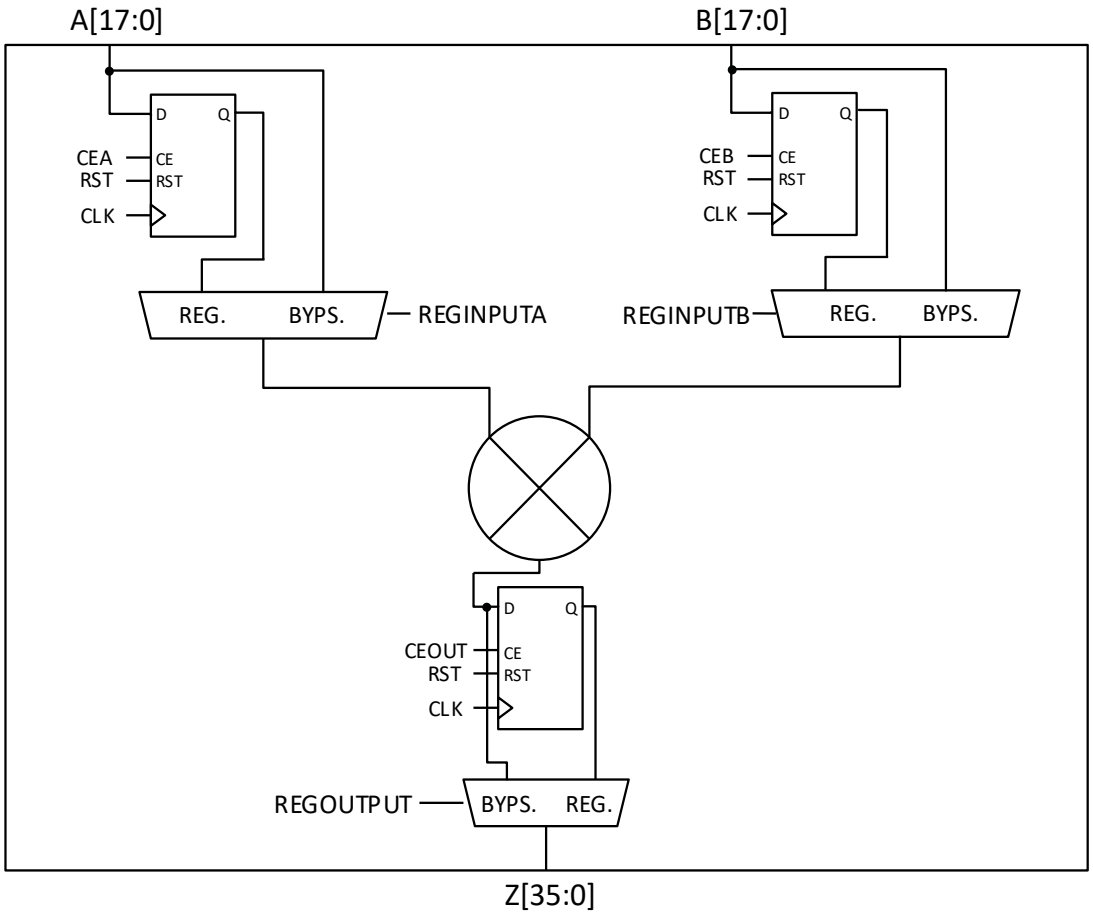

Figure 3.6 MULT18×18A Functional Diagram

3.3.1. MULT18×18A – I/O Ports

Table 3.5 describes the list of ports available for the primitive.

Table 3.5. MULT18×18A I/O Ports

Port	Input/Output	Description
A[17:0]	Input	Operand A (signed)
B[17:0]	Input	Operand B (signed)
CLK	Input	Clock
RST	Input	Reset
CEA	Input	Operand A register clock enable
CEB	Input	Operand B register clock enable
CEOUT	Input	Output register clock enable
Z[35:0]	Output	Multiplication result

3.3.2. MULT18×18A – Attributes

Table 3.6 describes the attributes for the primitive.

Table 3.6. MULT18×18A Attributes

Attribute Name	Values	Default Value	Description
REGINPUTA	REGISTERED, BYPASSED	REGISTERED	Operand A register
REGINPUTB	REGISTERED, BYPASSED	REGISTERED	Operand B register
REGOUTPUT	REGISTERED, BYPASSED	REGISTERED	Output register
GSR	ENABLED, DISABLED	DISABLED	Enable global set/reset
RESETMODE	SYNC, ASYNC	SYNC	Synchronous/asynchronous reset control

3.4. MULTADDSUB18×18A – 18×18 Multiplier with Pre-adder and Accumulator

MULTADDSUB18×18A implements an 18 × 18 multiplication with pre-adder and accumulator. The pre-adder and accumulator can be enabled or disabled to achieve different functions.

When pre-adder and accumulator are both enabled, it implements the function:

$$\boldsymbol{M_{INT} = A * (B \pm C)}$$

$$\boldsymbol{Z_{out} = CAS_{INT} + C_{INT} + CIN_{INT} \pm M_{INT}}$$

Note that the pre-adder and accumulator share the C input. If the pre-adder is used, typically ACCUM_C_EN should be set to DISABLED so that the accumulator ignores the value of the C port.

When pre-adder is enabled and accumulator is disabled, it implements the function:

$$\boldsymbol{Z_{out} = A * (B \pm C)}$$

When pre-adder is disabled and accumulator is enabled, it implements the function:

$$\boldsymbol{M_{int} = A * B}$$

$$\boldsymbol{Z_{out} = CAS_{INT} + C_{INT} + CIN_{INT} \pm M_{INT}}$$

The output of the accumulator is fed to a saturation logic that is responsible for checking if the result exceeds a defined limit. Refer to Appendix B. Rounding and Saturation for more details on saturation for more details.

Figure 3.7 shows the MULTADDSUB18×18A primitive available in Lattice Avant devices. The functional diagram of the multiplier is shown in Figure 3.8.

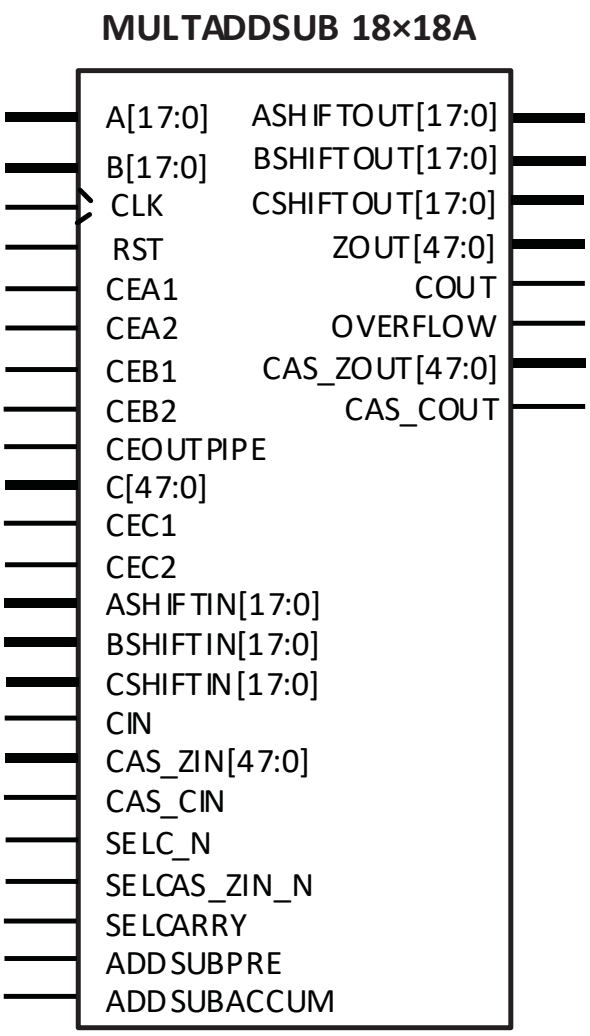

Figure 3.7. MULTADDSUB18×18A Primitive

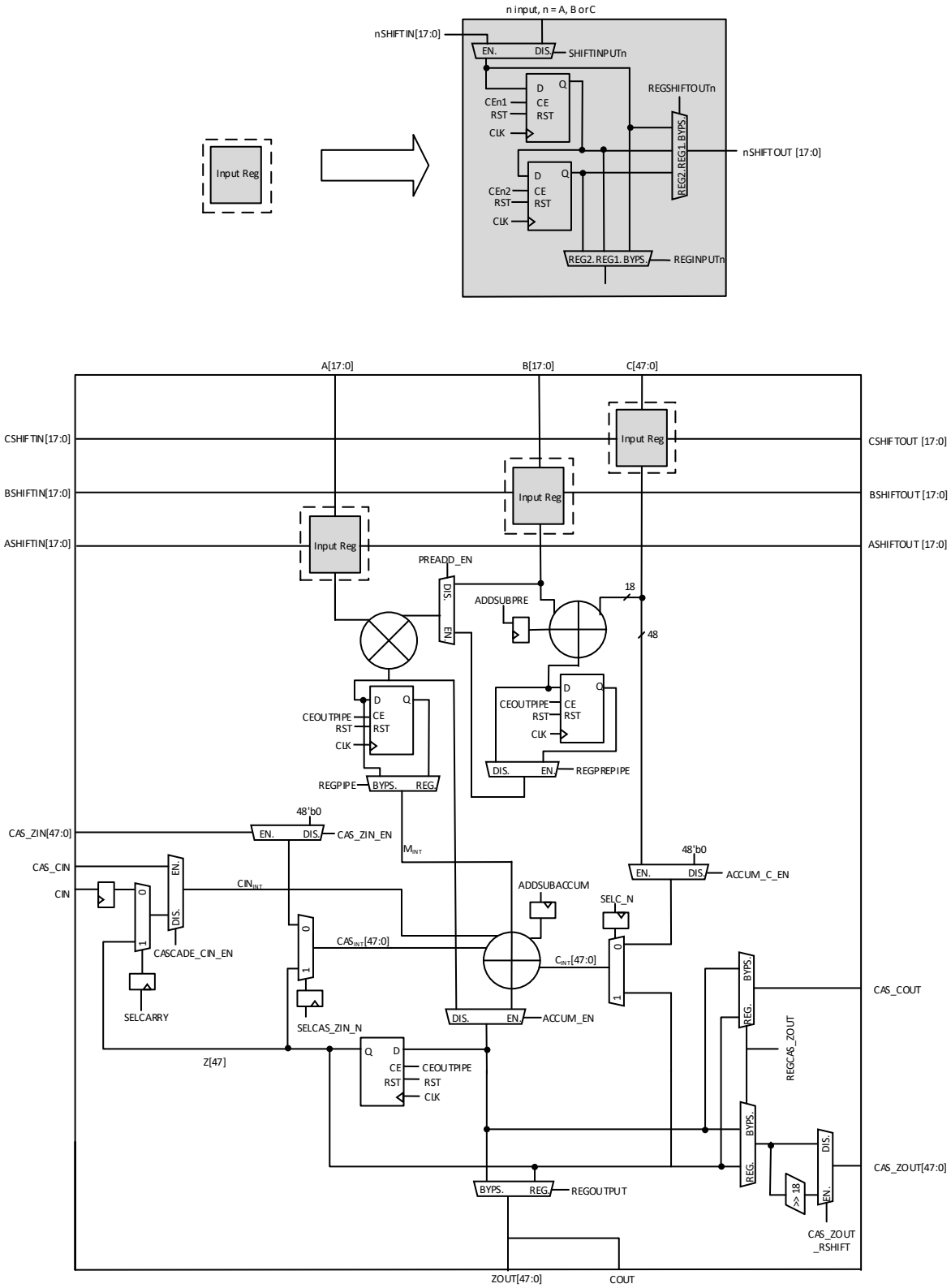

Figure 3.8. MULTADDSUB18×18A Functional Diagram

3.4.1. MULTADDSUB18×18A – I/O Ports

Table 3.7 describes the ports available for the primitive.

Table 3.7. MULTADDSUB18×18A I/O Ports

Port	Input/Output	Description
A[17:0]	Input	Operand A
B[17:0]	Input	Operand B
CLK	Input	Clock
RST	Input	Reset
CEA1	Input	Operand A register clock enable, stage one
CEA2	Input	Operand A register clock enable, stage two
CEB1	Input	Operand B register clock enable, stage one
CEB2	Input	Operand B register clock enable, stage two
CEOUTPIPE	Input	Output and pipeline register clock enable
C[47:0]	Input	Operand C
CEC1	Input	Operand C register clock enable, stage one
CEC2	Input	Operand C register clock enable, stage two
ASHIFTIN[17:0]	Input	Operand A shift chain input
BSHIFTIN[17:0]	Input	Operand B shift chain input
CSHIFTIN[17:0]	Input	Operand C shift chain input
CIN	Input	Carry in
CAS_ZIN[47:0]	Input	Cascade input from previous DSP block
CAS_CIN	Input	Cascade carry input from previous DSP block
SELC_N	Input	Select between C or accumulator output for input to accumulator
SELCAS_ZIN_N	Input	Select between CAS_ZIN or accumulator output for input to accumulator
SELCARRY	Input	Select between CIN or accumulator output Z[47] bit for input to accumulator
ADDSUBPRE	Input	Pre-adder add/sub control 0: add 1: subtract
ADDSUBACCUM	Input	Accumulator add/sub control 0: add 1: subtract
ASHIFTOUT[17:0]	Output	Operand A shift chain output
BSHIFTOUT[17:0]	Output	Operand B shift chain output
CSHIFTOUT[17:0]	Output	Operand C shift chain output
ZOUT[47:0]	Output	Multiplication/accumulator result
COUT	Output	Carry out
OVERFLOW	Output	Overflow flag, asserted if accumulation result exceeds max/min limit set by SATURATION_BITS
CAS_ZOUT[47:0]	Output	Cascade multiplication/accumulator result to next DSP block
CAS_COUT	Output	Cascade carry out to next DSP block

3.4.2. MULTADDSUB18×18A – Attributes

Table 3.8 describes the attributes for the primitive.

Table 3.8. MULTADDSUB18×18A Attributes

Attribute Name	Values	Default Value	Description
REGINPUTA	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A register
REGINPUTB	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B register
REGSHIFTOUTA	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A shift chain output register
REGSHIFTOUTB	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B shift chain output register
SHIFTINPUTA	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand A as input. ENABLED: Select Operand A shift chain input.
SHIFTINPUTB	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand B as input. ENABLED: Select Operand B shift chain input.
REGINPUTC	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand C register
SHIFTINPUTC	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand C as input. ENABLED: Select Operand C shift chain input.
REGSHIFTOUTC	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand C shift chain output register
REGPREPIPE	REGISTERED, BYPASSED	REGISTERED	Pre-adder output pipeline register
REGPIPE	REGISTERED, BYPASSED	REGISTERED	Multiplier output pipeline register
REGOUTPUT	REGISTERED, BYPASSED	REGISTERED	Accumulator output register
REGCAS_ZOUT	REGISTERED, BYPASSED	REGISTERED	CAS_COUT and CAS_ZOUT[47:0] output register
REGACCUMCONTROLS	REGISTERED, BYPASSED	REGISTERED	Accumulator control signals register (register for CIN, SELCARRY, SELCAS_ZIN_N, ADDSUBACCUM, SELC_N)
RESETMODE	SYNC, ASYNC	SYNC	Synchronous/asynchronous reset control
ACCUM_EN	ENABLED, DISABLED	DISABLED	Enable accumulator
CAS_ZOUT_RSHIFT	ENABLED, DISABLED	DISABLED	Cascade output 18 bits right shift
PREADD_EN	ENABLED, DISABLED	DISABLED	Enable pre-adder
REGADDSUBPRE	REGISTERED, BYPASSED	REGISTERED	ADDSUBPRE control signal register
ASIGNED	SIGNED, UNSIGNED	SIGNED	Operand A signed control (only used when cascading to build wider multipliers)
BSIGNED	SIGNED, UNSIGNED	SIGNED	Operand A signed control (only used when cascading to build wider multipliers)
ACCUM_C_EN	ENABLED, DISABLED	DISABLED	Enable C input to accumulator
CAS_ZIN_EN	ENABLED, DISABLED	DISABLED	Enable CAS_ZIN to accumulator
CAS_CIN_EN	ENABLED, DISABLED	DISABLED	Enable CAS_CIN for carry input
ROUNDMODE	ROUND_TO_ZERO, ROUND_TO_INFINITY	ROUND_TO_ZERO	Rounding mode
SATURATION	TO_NEG_MAX, TO_ZERO, DISABLED	DISABLED	Saturation mode
SATURATION_BITS	3 to 48	48	Set N-bit saturation
GSR	ENABLED, DISABLED	DISABLED	Enable global set/reset

3.5. DOTPRODADDSUB9X9 – 9×9 Multiply Add with Accumulator

DOTPRODADDSUB9×9 implements four 9 × 9 multiply-add function. The accumulator can be enabled or disabled to achieve different functions. Do note that C input is not available in this mode.

When accumulator is enabled, it implements the function:

$$M_{INT} = (A_0 * B_0) + (A_1 * B_1) + (A_2 * B_2) + (A_3 * B_3)$$

$$Z_{out} = CAS_{INT} + C_{INT} + CIN_{INT} \pm M_{INT}$$

The output of the accumulator is fed to a saturation logic that is responsible for checking if the result exceeds a defined limit. Refer to Appendix B. Rounding and Saturation for more details on saturation.

When accumulator is disabled, it implements the function:

$$Z_{OUT} = (A_0 * B_0) + (A_1 * B_1) + (A_2 * B_2) + (A_3 * B_3)$$

Figure 3.9 shows the DOTPRODADDSUB9×9 primitive available in Lattice Avant devices. The functional diagrams of the multiplier are shown in Figure 3.10 and Figure 3.11.

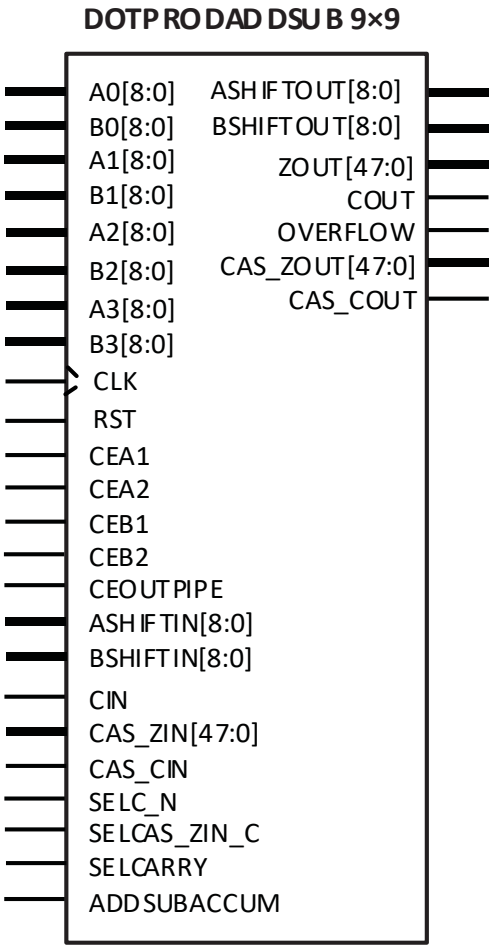

Figure 3.9. DOTPRODADDSUB9×9 Primitive

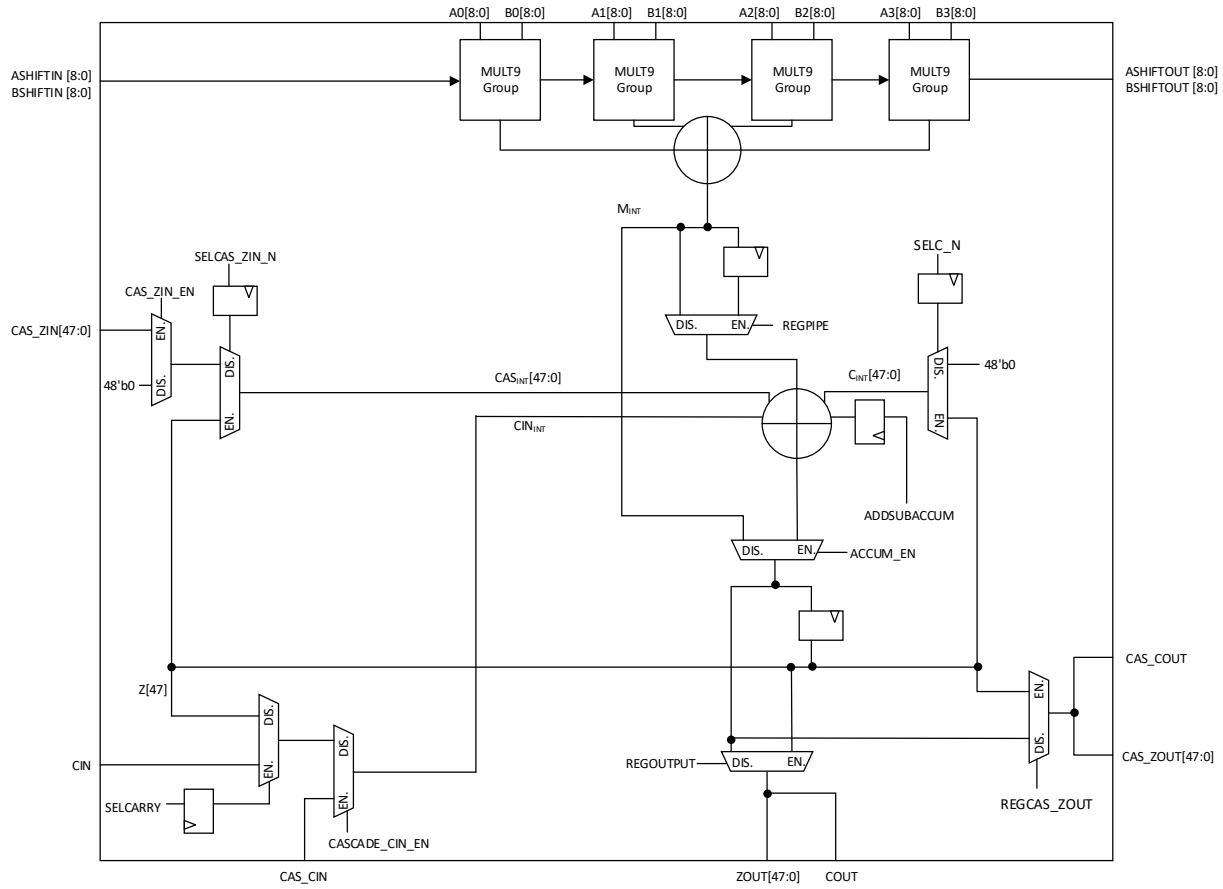

Figure 3.10. DOTPRODADDSUB9×9 Functional Diagram

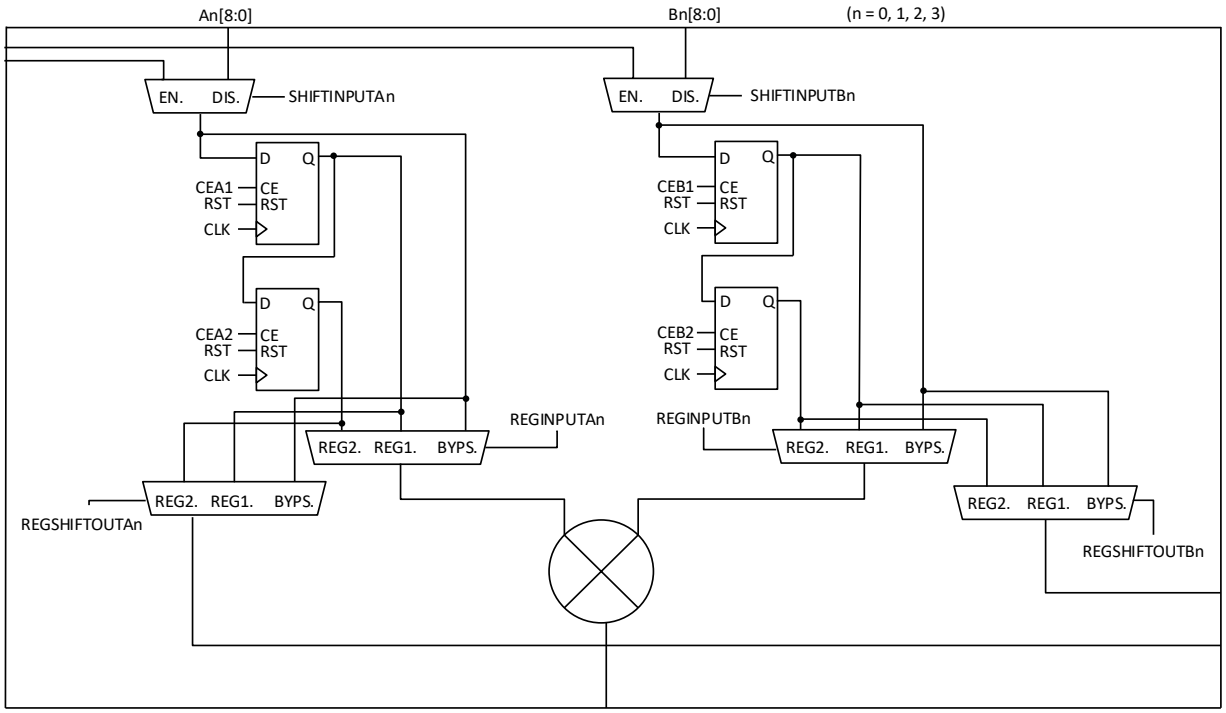

Figure 3.11. MULT9 Group Functional Diagram (1)

3.5.1. DOTPRODADDSUB9×9 – I/O Ports

Table 3.9 describes the ports available for the primitive.

Table 3.9. DOTPRODADDSUB9×9 I/O Ports

Port	Input/Output	Description
A0[8:0]	Input	Operand A0 (signed)
B0[8:0]	Input	Operand B0 (signed)
A1[8:0]	Input	Operand A1 (signed)
B1[8:0]	Input	Operand B1 (signed)
A2[8:0]	Input	Operand A2 (signed)
B2[8:0]	Input	Operand B2 (signed)
A3[8:0]	Input	Operand A3 (signed)
B3[8:0]	Input	Operand B3 (signed)
CLK	Input	Clock
RST	Input	Reset
CEA1	Input	Operand A register clock enable, stage one
CEA2	Input	Operand A register clock enable, stage two
CEB1	Input	Operand B register clock enable, stage one
CEB2	Input	Operand B register clock enable, stage two
CEOUTPIPE	Input	Output & pipeline register clock enable
ASHIFTIN[8:0]	Input	Operand A shift chain input
BSHIFTIN[8:0]	Input	Operand B shift chain input
CIN	Input	Carry in
CAS_ZIN[47:0]	Input	Cascade input from previous DSP block
CAS_CIN	Input	Cascade carry input from previous DSP block
SELC_N	Input	Select between C or accumulator output for input to accumulator.
SELCAS_ZIN_N	Input	Select between CAS_ZIN or accumulator output for input to accumulator.
SELCARRY	Input	Select between CIN or accumulator output Z[47] bit for input to accumulator.
ADDSUBACCUM	Input	Accumulator add/sub control 0: add 1: subtract
ASHIFTOUT[8:0]	Output	Operand A shift chain output
BSHIFTOUT[8:0]	Output	Operand B shift chain output
ZOUT[47:0]	Output	Multiplication/accumulator result
COUT	Output	Carry out
OVERFLOW	Output	Overflow flag, asserted if accumulation result exceeds max/min limit set by SATURATION_BITS
CAS_ZOUT[47:0]	Output	Cascade multiplication/accumulator result to next DSP block
CAS_COUT	Output	Cascade carry out to next DSP block

3.5.2. DOTPRODADDSUB9×9 – Attributes

Table 3.10 shows the attributes for the primitive.

Table 3.10. DOTPRODADDSUB9×9 Attributes

Attribute Name	Values	Default Value	Description
REGINPUTA0	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A0 register
REGINPUTB0	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B0 register
REGSHIFTOUTA0	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A0 shift chain output register
REGSHIFTOUTB0	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B0 shift chain output register
SHIFTINPUTA0	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand A0 as input. ENABLED: Select Operand A0 shift chain input.
SHIFTINPUTB0	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand B0 as input. ENABLED: Select Operand B0 shift chain input.
REGINPUTA1	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A1 register
REGINPUTB1	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B1 register
REGSHIFTOUTA1	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A1 shift chain output register
REGSHIFTOUTB1	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B1 shift chain output register
SHIFTINPUTA1	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand A1 as input. ENABLED: Select Operand A1 shift chain input.
SHIFTINPUTB1	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand B1 as input. ENABLED: Select Operand B1 shift chain input.
REGINPUTA2	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A2 register
REGINPUTB2	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B2 register
REGSHIFTOUTA2	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A2 shift chain output register
REGSHIFTOUTB2	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B2 shift chain output register
SHIFTINPUTA2	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand A2 as input. ENABLED: Select Operand A2 shift chain input.
SHIFTINPUTB2	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand B2 as input. ENABLED: Select Operand B2 shift chain input.
REGINPUTA3	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A3 register
REGINPUTB3	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B3 register
REGSHIFTOUTA3	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A3 shift chain output register

Attribute Name	Values	Default Value	Description
REGSHIFTOUTB3	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B3 shift chain output register
SHIFTINPUTA3	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand A3 as input. ENABLED: Select Operand A3 shift chain input.
SHIFTINPUTB3	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand B3 as input. ENABLED: Select Operand B3 shift chain input.
REGPIPE	REGISTERED, BYPASSED	REGISTERED	Multiplier output pipeline register
REGOUTPUT	REGISTERED, BYPASSED	REGISTERED	Accumulator output register
REGCAS_ZOUT	REGISTERED, BYPASSED	REGISTERED	CAS_COUT and CAS_ZOUT[47:0] output register
REGACCUMCONTROLS	REGISTERED, BYPASSED	REGISTERED	Accumulator control signals register (register for CIN, SELCARRY, SELCAS_ZIN_N, ADDSUBACCUM, SELC_N)
RESETMODE	SYNC, ASYNC	SYNC	Synchronous/asynchronous reset control.
ACCUM_EN	ENABLED, DISABLED	DISABLED	Enable accumulator
CAS_ZIN_EN	ENABLED, DISABLED	DISABLED	Enable CAS_ZIN to accumulator
CAS_CIN_EN	ENABLED, DISABLED	DISABLED	Enable CAS_CIN for carry input
ROUNDMODE	ROUND_TO_ZERO, ROUND_TO_INFINITY	ROUND_TO_ZERO	Rounding mode
SATURATION	TO_NEG_MAX, TO_ZERO, DISABLED	DISABLED	Saturation mode
SATURATION_BITS	3 to 48	48	Set N-bit saturation
GSR	ENABLED, DISABLED	DISABLED	Enable global set/reset

3.6. DOTPRODADDSUB9X9A – 9×9 Multiply Add with Accumulator and Cin

DOTPRODADDSUB9×9A implements four 9 × 9 multiply-add function. The accumulator can be enabled or disabled to achieve different functions. Operands are shifted in through the shift chain input in this mode, which enables operand C to be fed to the accumulator.

When accumulator is enabled, it implements the function:

$$M_{INT} = (A_z * B_z) + (A_{z-1} * B_{z-1}) + (A_{z-2} * B_{z-2}) + (A_{z-3} * B_{z-3})$$

$$Z_{out} = CAS_{INT} + C_{INT} + CIN_{INT} \pm M_{INT}$$

The output of the accumulator is fed to a saturation logic that is responsible for checking if the result exceeds a defined limit. Refer to Appendix B. Rounding and Saturation for more details on saturation.

When accumulator is disabled, it implements the function:

$$Z_{OUT} = (A_z * B_z) + (A_{z-1} * B_{z-1}) + (A_{z-2} * B_{z-2}) + (A_{z-3} * B_{z-3})$$

Figure 3.12 shows the DOTPRODADDSUB9×9A primitive available in Lattice Avant devices. The functional diagrams of the multiplier are shown in Figure 3.13 and Figure 3.14.

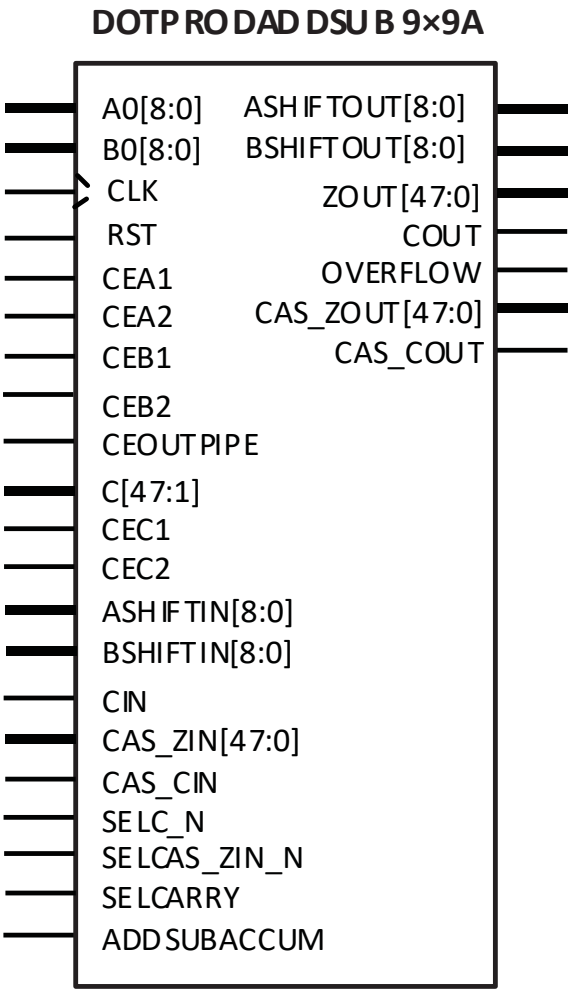

Figure 3.12. DOTPRODADDSUB9×9A Primitive

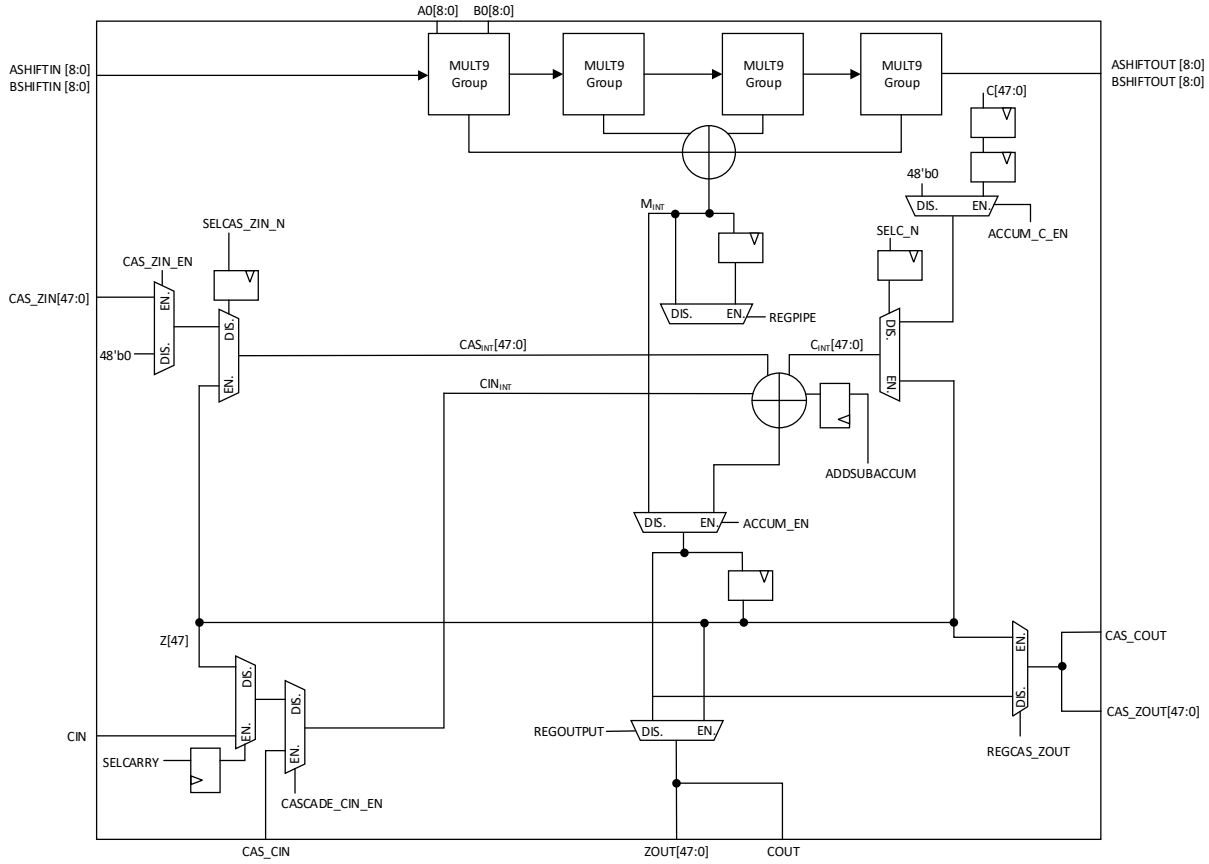

Figure 3.13. DOTPRODADDSUB9×9A Functional Diagram

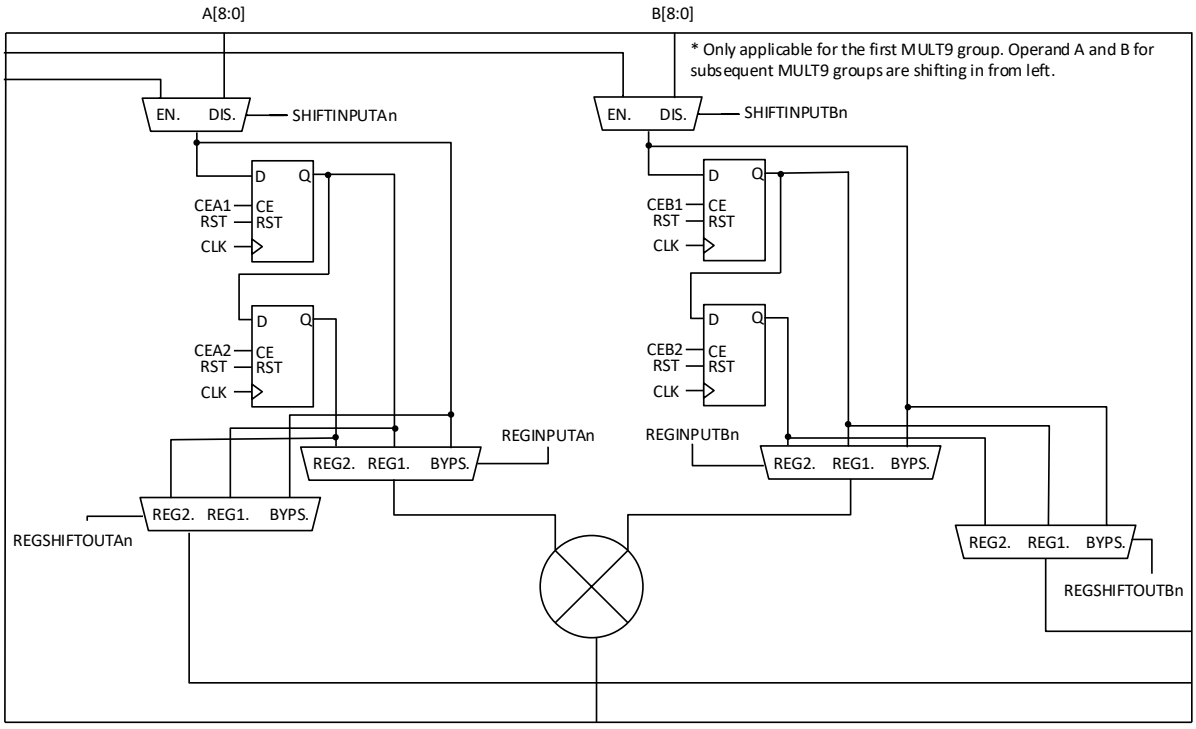

Figure 3.14. MULT9 Group Functional Diagram (2)

3.6.1. DOTPRODADDSUB9×9A – I/O Ports

Table 3.11 describes the ports available for the primitive.

Table 3.11. DOTPRODADDSUB9×9A I/O Ports

Port	Input/Output	Description
A0[8:0]	Input	Operand A0 (signed)
B0[8:0]	Input	Operand B0 (signed)
CLK	Input	Clock
RST	Input	Synchronous reset for all registers.
CEA1	Input	Operand A register clock enable, stage one.
CEA2	Input	Operand A register clock enable, stage two.
CEB1	Input	Operand B register clock enable, stage one.
CEB2	Input	Operand B register clock enable, stage two.
CEOUTPIPE	Input	Output and pipeline register clock enable.
C[47:0]	Input	Operand C
CEC1	Input	Operand C register clock enable, stage one.
CEC2	Input	Operand C register clock enable, stage two.
ASHIFTIN[8:0]	Input	Operand A shift chain input.
BSHIFTIN[8:0]	Input	Operand B shift chain input.
CIN	Input	Carry in
CAS_ZIN[47:0]	Input	Cascade input from previous DSP block.
CAS_CIN	Input	Cascade carry input from previous DSP block.
SELC_N	Input	Select between C or accumulator output for input to accumulator.
SELCAS_ZIN_N	Input	Select between CAS_ZIN or accumulator output for input to accumulator.
SELCARRY	Input	Select between CIN or accumulator output Z[47] bit for input to accumulator.
ADDSUBACCUM	Input	Accumulator add/sub control, 0 = add, 1 = subtract
ASHIFTOUT[8:0]	Output	Operand A shift chain output
BSHIFTOUT[8:0]	Output	Operand B shift chain output
ZOUT[47:0]	Output	Multiplication/accumulator result
COUT	Output	Carry out
OVERFLOW	Output	Overflow flag, asserted if accumulation result exceeds max/min limit set by SATURATION_BITS.
CAS_ZOUT[47:0]	Output	Cascade multiplication/accumulator result to next DSP block
CAS_COUT	Output	Cascade carry out to next DSP block

3.6.2. DOTPRODADDSUB9×9A – Attributes

Table 3.12 describes the attributes for the primitive.

Table 3.12. DOTPRODADDSUB9×9A Attributes

Attribute Name	Values	Default Value	Description
REGINPUTA0	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A0 register
REGINPUTB0	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B0 register
REGSHIFTOUTA0	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A0 shift chain output register
REGSHIFTOUTB0	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B0 shift chain output register
SHIFTINPUTA0	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand A0 as input. ENABLED: Select Operand A0 shift chain input
SHIFTINPUTB0	DISABLED, ENABLED	DISABLED	DISABLED: Select Operand B0 as input. ENABLED: Select Operand B0 shift chain input.
REGINPUTA1	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A1 register
REGINPUTB1	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B1 register
REGSHIFTOUTA1	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A1 shift chain output register
REGSHIFTOUTB1	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B1 shift chain output register
REGINPUTA2	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A2 register
REGINPUTB2	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B2 register
REGSHIFTOUTA2	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A2 shift chain output register
REGSHIFTOUTB2	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B2 shift chain output register
REGINPUTA3	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A3 register
REGINPUTB3	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B3 register
REGSHIFTOUTA3	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand A3 shift chain output register
REGSHIFTOUTB3	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand B3 shift chain output register
REGINPUTC	REGISTERED_ONCE, REGISTERED_TWICE, BYPASSED	REGISTERED_ONCE	Operand C register
REGPIPE	REGISTERED, BYPASSED	REGISTERED	Multiplier output pipeline register
REGOUTPUT	REGISTERED, BYPASSED	REGISTERED	Accumulator output register
REGCAS_ZOUT	REGISTERED, BYPASSED	REGISTERED	CAS_COUT and CAS_ZOUT[47:0] output register
REGACCUMCONTROLS	REGISTERED, BYPASSED	REGISTERED	Accumulator control signals register (register for CIN, SELCARRY, SELCAS_ZIN_N, ADDSUBACCUM, SELC_N)
RESETMODE	SYNC, ASYNC	SYNC	Synchronous/asynchronous reset control
ACCUM_EN	ENABLED, DISABLED	DISABLED	Enable accumulator
ACCUM_C_EN	ENABLED, DISABLED	DISABLED	Enable C input to accumulator

Attribute Name	Values	Default Value	Description
CAS_ZIN_EN	ENABLED, DISABLED	DISABLED	Enable CAS_ZIN to accumulator
CAS_CIN_EN	ENABLED, DISABLED	DISABLED	Enable CAS_CIN for carry input
ROUNDMODE	ROUND_TO_ZERO, ROUND_TO_INFINITY	ROUND_TO_ZERO	Rounding mode
SATURATION	TO_NEG_MAX, TO_ZERO, DISABLED	DISABLED	Saturation mode
SATURATION_BITS	3 to 48	48	Set N-bit saturation
GSR	ENABLED, DISABLED	DISABLED	Enable global set/reset

4. Using sysDSP

The sysDSP blocks can be used in a number of ways in Lattice Avant devices, as described in the following sections.

4.1. sysDSP Primitive Instantiation

Lattice sysDSP primitives can be directly instantiated in the design. Each of the primitives has a fixed set of attributes that can be customized to meet the design requirements. Lattice Avant sysDSP supports the following primitives:

- MULT8 × 8 (8-bit multiply)
- MULT9 × 9A (9-bit multiply)
- MULT18 × 18A (18-bit multiply)
- MULTADDSUB18 × 18A (18-bit multiply with pre-adder and accumulator)
- DOTPRODADDSUB9 × 9/DOTPRODADDSUB9 × 9A (9 × 9 multiply add with accumulator)

An example of the MULT18×18 primitive instantiation is provided in the following Instantiating sysDSP Primitives in HDL section.

4.2. Instantiating sysDSP Primitives in HDL

This section illustrates how to instantiate Lattice Avant sysDSP primitives in both Verilog HDL and VHDL.

4.2.1. Verilog HDL Example Showing Snippet of the MULT18×18A Instantiation

```
defparam dsp_mult_0.REGINPUTA = "REGISTERED" ;
defparam dsp_mult_0.REGINPUTB = "REGISTERED" ;
defparam dsp_mult_0.REGOUTPUT = "REGISTERED" ;
defparam dsp_mult_0.GSR = "ENABLED" ;
defparam dsp_mult_0.RESETMODE = "SYNC" ;

MULT18X18A dsp_mult_0 (.CLK(clk_i),
RST(rst_i),
CEA(ce_a_i),
CEB(ce_b_i),
CEOUT(ce_o_i),
A(data_a_i[17:0]),
B(data_b_i[17:0]),
Z(result_o[35:0])) ;
```

4.2.2. VHDL Example Showing Snippet of the MULT18×18A Instantiation

```
dsp_mult_0 : MULT18X18A
             generic map (
                    REGINPUTA => "REGISTERED",
                    REGINPUTB => "REGISTERED",
                    REGOUTPUT => "REGISTERED",
                    GSR => "ENABLED",
                    RESETMODE => "SYNC"
                    )
              port map (
                    CLK => clk_i,
                    RST => rst_i,
                    CEA => ce_a_i,
                    CEB => ce_b_i,
                    CEOUT=> ce_o_i,
                    A => data_a_i,
```

```
                B => data_b_i,
                Z => result_o
        );
```

4.3. Inferencing sysDSP Block

sysDSP can be inferred by synthesis tools. An example of the HDL inference for sysDSP is provided in Appendix A. HDL Inference for sysDSP.

Appendix A. HDL Inference for sysDSP

Synthesis inference flow enables the design tools to infer sysDSP blocks from an HDL design. It is important to note that when using the inference flow, unless the code style matches the sysDSP block, results are not optimal. You can infer the sysDSP block with Synplify Pro® from Synopsys in Lattice Radiant software if certain coding guidelines are followed. Following are Verilog HDL and VHDL examples. These examples would not have functional simulation support. This is for example purposes only.

Verilog HDL Example to Infer Fully Pipelined Multiplier

```
module mult (dataout, dataax, dataay, clk, reset);
output signed [35:0] dataout;
input signed [17:0] dataax, dataay;
input clk,reset;

reg signed [35:0] dataout;
reg signed [17:0] dataax_reg, dataay_reg;
wire signed [35:0] dataout_node;

always @(posedge clk or posedge reset)
       begin
          if (reset) begin
             dataax_reg <= 0;
             dataay_reg <= 0;
          end else begin
              dataax_reg <= dataax;
              dataay_reg <= dataay;
          end
       end

assign dataout_node = dataax_reg * dataay_reg;

always @(posedge clk or posedge reset)
        begin
            if (reset)
               dataout <= 0;
            else
               dataout <= dataout_node;
         end
endmodule
```

VHDL Example to Infer Fully Pipelined Multiplier

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity mult is
        port (reset, clk : in std_logic;
        dataax, dataay : in signed (17 downto 0);
        dataout : out signed (35 downto 0));
end;
```

```
architecture arch of mult is
     signal dataax_reg, dataay_reg : signed (17 downto 0);
begin
     process (clk, reset)
     begin
        if (reset='1') then
           dataax_reg <= (others => '0');
           dataay_reg <= (others => '0');
        elsif (clk'event and clk='1') then
           dataax_reg <= dataax;
           dataay_reg <= dataay;
        end if;
      end process;

      process (clk, reset)
      begin
         if (reset='1') then
            dataout <= (others => '0');
        elsif (clk'event and clk='1') then
            dataout <= dataax_reg * dataay_reg;
        end if;
      end process;
end arch;
```

Appendix B. Rounding and Saturation

Rounding operation is done by adding a round constant (Rconst) to the adder/accumulator result before truncating the less significant bits. For example, to round a 16-bit number to an 8-bit number, a rounding constant is added to the 16-bit number and the upper byte is retained as rounded 8-bit result, while the lower byte is discarded.

Lattice Avant sysDSP block can support several rounding modes with rounding constant provided through C input. This approach enables rounding to any bit position. This section discusses three rounding modes, namely Round-Half-Up (RHU), Round-to-Zero (RTZ), and Round-to-Infinity (RTI).

Round-Half-Up (RHU) is to round every half point to $+\infty$ and is done by adding a rounding constant of 0.5 to the number to be rounded and truncating the lower bits. In binary form, this is done by adding Rconst=2^(n-1) where n is the LSB to rounded integer.

Integer LSB bit N

$$(5.5)_{10} = (0101.10)_2$$

bit N-1

The examples below show the rounding of a 6-bit signed binary number (xxxx.xx) to 4 bits (xxxx) with RHU where the rounding constant is $(0000.10)_2 = (0.10)_2$.

Binary	Decimal	Rounding by Adding Rconst = 0.10	Rounded Binary	Rounded Decimal
0101.01	5.25	0101.01 + 0.10 = 0101.11	0101	5
0101.10	5.5	0101.10 + 0.10 = 0110.00	0110	6
0101.11	5.75	0101.11 + 0.10 = 0110.01	0110	6
1010.11	–5.25	1010.11 + 0.10 = 1011.01	1011	–5
1010.10	–5.5	1010.10 + 0.10 = 1011.00	1011	–5
1010.01	–5.75	1010.01 + 0.10 = 1010.11	1010	–6

Round-to-Zero (RTZ) rounds every half-point toward zero. It can be done by adding all 1s from bit 0 to bit N-2 and sign of the number to round (z[47]); N is the rounding position or the LSB of the rounded number. In sysDSP, Rconst can be fed to C input and z[47] feedback provides the sign of the number to be rounded.

$$z[47] + \sum_{i=0}^{N-2} 2^i$$

With z[47] fed back as the sign bit, an extra clock cycle is required to obtain the accurate rounded result. Note that it is possible to perform rounding using the current z[47] value, but this may introduce a 1 LSB error to the rounded result.

The following table lists examples of rounding 7-bit 2's complement number to 4 bits with RTZ. The rounding constant is $(0000.011)_2$, with addition of signed bit $(0000.00x)_2$.

Binary	Decimal	Rounding by Adding Rconst = 0.011 and Signed Bit	Rounded Binary	Rounded Decimal
0101.001	5.125	0101.001 + 0.011 + 0.000 = 0101.100	0101	5
0101.010	5.25	0101.010 + 0.011 + 0.000 = 0101.101	0101	5
0101.100	5.5	0101.100 + 0.011 + 0.000 = 0101.111	0101	5
0101.101	5.625	0101.101 + 0.011 + 0.000 = 0110.000	0110	6
0101.110	5.75	0101.110 + 0.011 + 0.000 = 0110.001	0110	6
0101.111	5.875	0101.111 + 0.011 + 0.000 = 0110.010	0110	6
1010.111	–5.125	1010.111 + 0.011 + 0.001 = 1011.011	1011	–5
1010.110	–5.25	1010.110 + 0.011 + 0.001 = 1011.010	1011	–5
1010.100	–5.5	1010.100 + 0.011 + 0.001 = 1011.000	1011	–5
1010.011	–5.625	1010.011 + 0.011 + 0.001 = 1010.111	1010	–6
1010.010	–5.75	1010.010 + 0.011 + 0.001 = 1010.110	1010	–6
1010.001	–5.875	1010.001 + 0.011 + 0.001 = 1010.101	1010	–6

Round-to-Infinity (RTI) rounds every half point away from zero. This involves adding a rounding constant of all 1s from bit N-2 to bit 0 through the C input, as well as inverse of the signed bit to the result before truncating. The z[47] feedback to carry input mux can be used for this purpose.

For example, to round a 7-bit signed binary number (xxxx.xxx) to 4 bits (xxxx) with RTI, the rounding constant is $(0000.011)_2$, with addition of inverted signed bit $(0000.00x)_2$.

Binary	Decimal	Rounding by adding Rconst = 0.011 and Signed Bit	Rounded Binary	Rounded Decimal
0101.001	5.125	0101.001 + 0.011 + 0.001 = 0101.101	0101	5
0101.010	5.25	0101.010 + 0.011 + 0.001 = 0101.110	0101	5
0101.100	5.5	0101.100 + 0.011 + 0.001 = 0110.000	0110	6
0101.101	5.625	0101.101 + 0.011 + 0.001 = 0110.001	0110	6
0101.110	5.75	0101.110 + 0.011 + 0.001 = 0110.010	0110	6
0101.111	5.875	0101.111 + 0.011 + 0.001 = 0110.011	0110	6
1010.111	–5.125	1010.111 + 0.011 + 0.000 = 1011.010	1011	–5
1010.110	–5.25	1010.110 + 0.011 + 0.000 = 1011.001	1011	–5
1010.100	–5.5	1010.100 + 0.011 + 0.000 = 1010.111	1010	–6
1010.011	–5.625	1010.011 + 0.011 + 0.000 = 1010.110	1010	–6
1010.010	–5.75	1010.010 + 0.011 + 0.000 = 1010.101	1010	–6
1010.001	–5.875	1010.001 + 0.011 + 0.000 = 1010.100	1010	–6

Figure B.1. below shows the carry input mux selection for z[47] to support rounding operation.

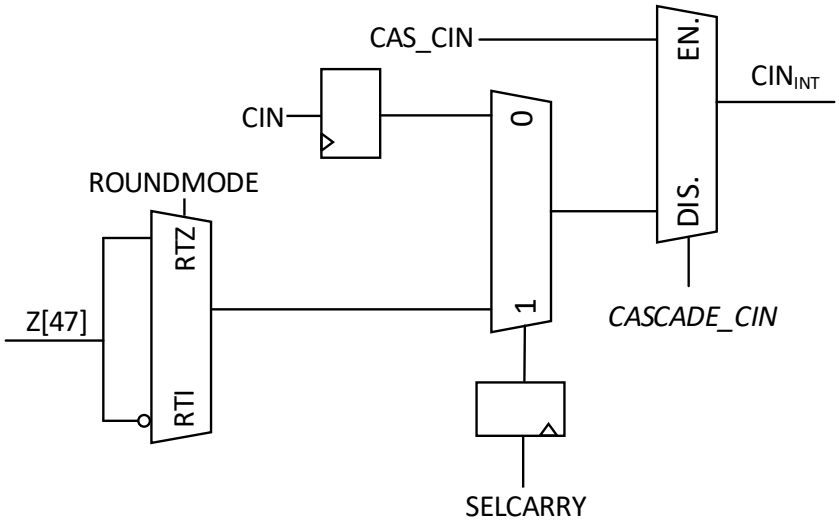

Figure B.1. Carry Input Mux Selection

Below is an example of accumulating and rounding a 16-bit two's complement number to a 13-bit number with RTZ. The rounding constant is $(0000.011)_2$, with addition of signed bit $(0000.001)_2$.

See attribute values listed below. Other attribute values are default.

```
REGINPUTA=REGISTERED_ONCE
REGINPUTB=REGISTERED_ONCE
REGINPUTC=REGISTERED_ONCE
REGACCUMCONTROLS=REGISTERED
REGPIPE=BYPASSED
REGOUTPUT=REGISTERED
PREADD_EN=DISABLE
ACCUM_EN=ENABLED
ACCUM_C_EN=ENABLED
```

```
PREADD_EN=DISABLED
ROUNDMODE=ROUND_TO_ZERO
```

Figure B.2 shows the high-level functional diagram of the DSP configuration in this example.

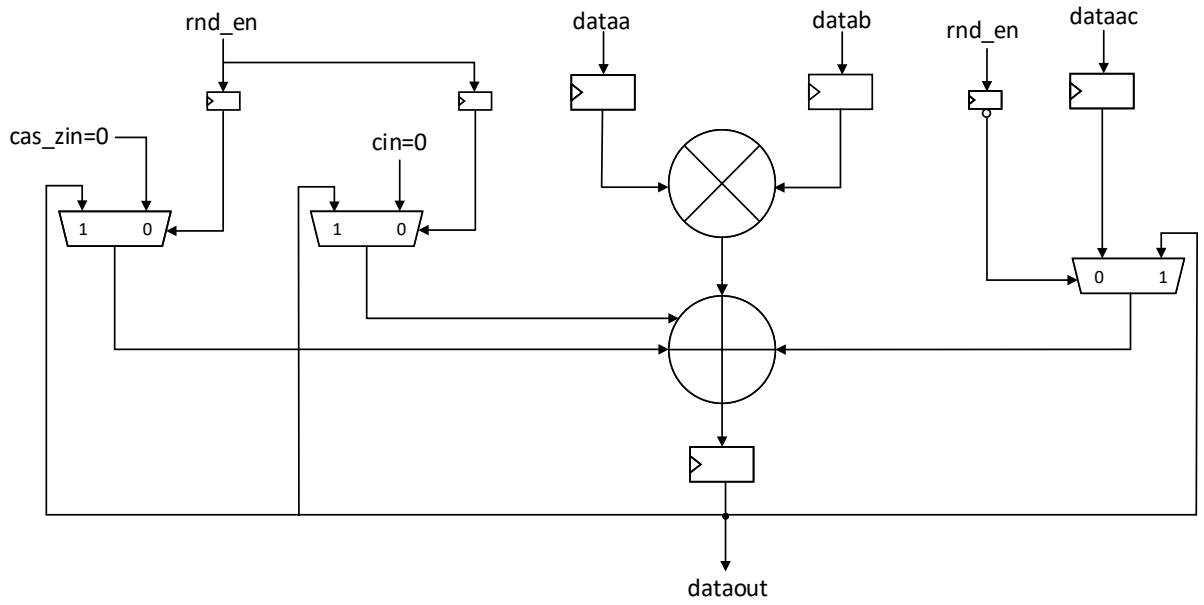

Figure B.2. Accumulating and Rounding 16-Bit Two's Complement Number to 13-Bit with RTZ

Figure B.3 shows the simulation waveform for this example. Note that there is a two-clock cycle latency from input to output.

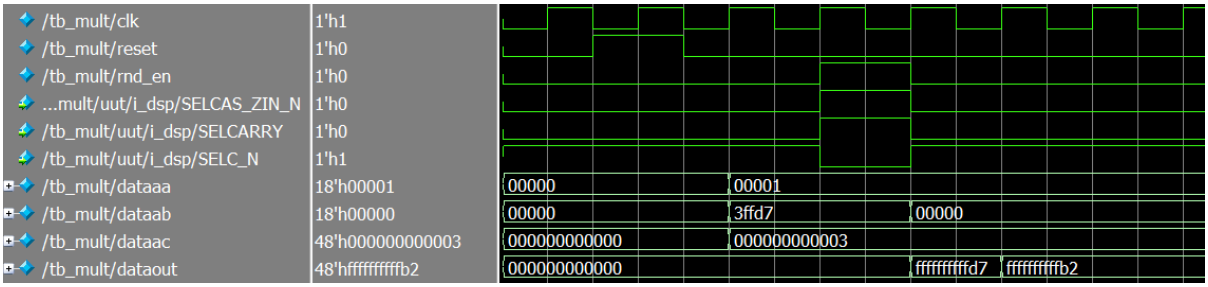

Figure B.3. Simulation Waveform

Rounding by adding Rconst and signed bit:

–5.125 $(1111111111010.111)_2$ + –5.125 $(1111111111010.111)_{16}$ + 0.375 $(0.011)_2$ + 0.125 $(0.001)_2$ = –9.75 $(1111111111111010.010)_2$

The simulation values are shown in HEX format for easier viewing. For example, the 48-bit HEX value of –9.75 is FFFFFFFFFFB2.

With the upper 13 bits are retained as rounded result:

Rounded binary number = $(1111111111111010)_2$.

Rounded decimal number = –10

For saturation, the result of accumulator is fed to a saturation logic that is responsible for checking if the result exceeds the maximum or minimum limit defined by SATURATION_BITS. If the result exceeds the limit, the maximum or minimum value is chosen instead as accumulator output and overflow flags are asserted. The saturated values for a given N-bit saturation ("TO_NEG_MAX" mode) are:

$$-2^{N-1} \leq ZOUT < 2^{N-1}$$

For example, an eight-bit saturation has the range of (Min, Max) = (-2^7, 2^7-1) = (-128, 127), which is the range of eight-bit 2's complement number.

For saturation TO_ZERO mode, the output is saturated to zero every time the input is negative. In this mode, the positive value saturation and overflow flag behave the same way, but the negative overflow flag is asserted when the input is negative and the saturated value is zero. The saturated values for a given N-bit saturation ("TO_ZERO" mode) are:

$$\mathbf{0} \leq \boldsymbol{ZOUT} < \mathbf{2}^{\boldsymbol{N-1}}$$

When saturation is enabled, carryout output is no longer valid and must be ignored. Saturation logic only works in one sysDSP. When multiple sysDSP blocks are used to extend accumulator width or to form wider multiplier, the saturation logic must be disabled. Saturation logic is available in MULTADDSUB18 × 18A, DOTPRODADDSUB9 × 9, and DOTPRODADDBUS9 × 9A and requires ACCUM_EN = ENABLED. When SATURATION = DISABLED, overflow flag still reflects the overflow condition set by SATURATION_BITS but the output is not saturated.

References

- Avant-E web page
- Avant-G web page
- Avant-X web page

A variety of technical notes for the Lattice Avant platform are available.

- Lattice Avant Embedded Memory User Guide (FPGA-TN-02289)
- Lattice Avant Hardware Checklist (FPGA-TN-02317)
- Lattice Avant High-Speed I/O and External Memory Interface (FPGA-TN-02300)
- Lattice Avant Multi-Boot User Guide (FPGA-TN-02314)
- Lattice Avant Power User Guide (FPGA-TN-02291)
- Lattice Avant SED/SEC User Guide (FPGA-TN-02290)
- Lattice Avant SERDES/PCS User Guide (FPGA-TN-02313)
- Lattice Avant sysCLOCK PLL Design and User Guide (FPGA-TN-02298)
- Lattice Avant sysCONFIG User Guide (FPGA-TN-02299)
- Lattice Avant sysI/O User Guide (FPGA-TN-02297)
- Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide (FPGA-UG-02039)
- High-Speed PCB Design Considerations (FPGA-TN-02178)
- sub-LVDS Signaling Using Lattice Devices (FPGA-TN-02028)
- Thermal Management (FPGA-TN-02044)
- Using TraceID (FPGA-TN-02084)

For more information on Lattice Avant-related IP, reference designs, and board documents, refer to the following pages:

- SGMII and Gb Ethernet PCS IP Core – Lattice Radiant Software (FPGA-IPUG-02077)
- Avant-E Evaluation Board – User Guide (FPGA-EB-02057)
- Avant-G/X Versa Board User Guide (FPGA-EB-02063)
- IP Cores and Reference Designs for Avant Devices
- Kits, Boards, and Demonstrations for Avant Devices

For further information on interface standards refer to the following websites:

- JEDEC Standards (LVTTL, LVCMOS, SSTL) – www.jedec.org
- PCI – www.pcisig.com

Other references:

- Lattice Insights for Lattice Semiconductor training courses and learning plans
- Lattice Radiant FPGA design software

Technical Support Assistance

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

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

Revision History

Revision 0.85, February 2025

Section	Change Summary

Targeting the sysDSP Block by Instantiating Primitives	• Table 3.8. MULTADDSUB18×18A Attributes:
	For the REGACCUMCONTROLS attribute:
	  • changed SELCAS_ZIN to SELCAS_ZIN_N in Description;
	  • changed SELC to SELC_N in Description.
	• Table 3.10. DOTPRODADDSUB9×9 Attributes:
	For the REGACCUMCONTROLS attribute:
	  • changed SELCAS_ZIN to SELCAS_ZIN_N in Description;
	  • changed SELC to SELC_N in Description.
	• Table 3.12. DOTPRODADDSUB9×9A Attributes:
	For the REGACCUMCONTROLS attribute:
	  • changed SELCAS_ZIN to SELCAS_ZIN_N in Description;
	  • changed SELC to SELC_N in Description.

Appendix B. Rounding and Saturation	• Added *With z[47] fed back as the sign bit, an extra clock cycle is required to obtain the accurate rounded result. Note that it is possible to perform rounding using the current z[47] value, but this may introduce a 1 LSB error to the rounded result* to the description. • Added an example for accumulating and rounding a 16-bit two's complement number to a 13-bit number with RTZ and its simulation waveform under Figure B.1. Carry Input Mux Selection.

Revision 0.84, August 2024

Section	Change Summary
Abbreviations in This Document	• Changed section header to the current; • Newly added HDL, RRH and VHDL.
Introduction	Removed duplicated description about multiple sysDSP instances.
sysDSP Overview	Added a Note to Figure 2.1. sysDSP Block Diagram.
Targeting the sysDSP Block by Instantiating Primitives	• Changed the reference to the Instantiating sysDSP Primitives in HDL in the introduction description of the first paragraph. • Changed the description to Reset for RST port in Table 3.1. MULT8×8 I/O Ports. • Removed the *User Interface* column from Table 3.2. MULT8×8 Attributes. • Removed the *User Interface* column from Table 3.4. MULT9X9A. • Changed the description to Reset for RST port in Table 3.5. MULT18×18A I/O Ports. • Removed the *User Interface* column from Table 3.6. MULT18×18A Attributes . • Updated Figure 3.8. MULTADDSUB18×18A Functional Diagram. • Changed REGPIPELINE to REGPIPE in Figure 3.8. MULTADDSUB18×18A Functional Diagram. • Changed the description to Reset for RST port in Table 3.7. MULTADDSUB18×18A I/O Ports . • Removed the *User Interface* column from Table 3.8. MULTADDSUB18×18A. • Changed the description to Reset for RST port in Table 3.9. DOTPRODADDSUB9×9 I/O Ports . • Removed the *User Interface* column from Table 3.10. DOTPRODADDSUB9×9. • Changed the description to Reset for RST port in Table 3.11. DOTPRODADDSUB9×9A I/O Ports . • Removed the *User Interface* column from Table 3.12. DOTPRODADDSUB9×9A.
Using sysDSP	• Removed You can get the detailed list of the primitives from the synthesis libraries under cae_library\synthesis folder under Radiant™ installation directory from the sysDSP Primitive Instantiation section. • Removed the equation from the Inferencing sysDSP Block section.

Revision 0.83, November 2023

Section	Change Summary
All	Adjusted section structure and exchanged contents section wise.
Disclaimers	Updated this section.
Introduction	General update to the contents.
sysDSP Overview	General update to the contents in the first paragraph.
Using sysDSP	Added equation to the Inferencing sysDSP Block section.
References	Newly added this section.

Revision 0.82, April 2023

Section	Change Summary
All	Added the support for Lattice Avant G/X devices.
Inclusive Language	Newly added section.
sysDSP Overview	Updated Figure 2.1. DSP Block Diagram Overview.
Using sysDSP	Removed description regarding the round mode in DSP blocks, as well as the original Figure 2.2. Roundmode DSP Block Diagram.

Targeting the sysDSP Block by Instantiating Primitives	• Updated the following figures:
	  • Figure 4.8. MULTADDSUB18×18A Functional Diagram
	  • Figure 4.10. DOTPRODADDSUB9×9 Functional Diagram
	  • Figure 4.11. MULT9 Group Functional Diagram (1)
	  • Figure 4.13. DOTPRODADDSUB9×9A Functional Diagram
	  • Figure 4.14. MULT9 Group Functional Diagram (2)
	• Updated the Description in Table 4.8. Attribute Description for MULTADDSUB18×18A.
	• Updated the description regarding the output of accumulator in the DOTPRODADDSUB9X9 – 9×9 Multiply Add with Accumulator section and in the DOTPRODADDSUB9X9A – 9×9 Multiply Add with Accumulator and Cin section.

Revision 0.81. November 2022

Section	Change Summary
All	Changed the wording for Avant device family support.
sysDSP Overview	Updated Figure 2.1. sysDSP Block Diagram .
Targeting the sysDSP Block by Instantiating Primitives	• Updated Figure 3.2. MULT8×8 Functional Diagram and Figure 3.4. MULT9×9A Functional Diagram. • Updated to the current DOTPRODADDSUB9×9A – Attribute Description section, including Table 3.12. DOTPRODADDSUB9×9A.

Revision 0.80, May 2022

Section	Change Summary
All	Preliminary release.

www.latticesemi.com