

Versal ACAP DSP Engine

Architecture Manual

AM004 (v1.1) December 4, 2020



Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/04/2020 Version 1.1	
Figure 9: Hierarchical View of the DSP58 Input Registers and Pre-Adder	Updated to fix a connection on MUX from the output of the B2 register.
Bus Multiplexer	Updated figure to add flop stages. Revised the table to update OPMODE settings.
Division	Updated the dividing with multiplication case to change initial value Q[8-n] from B to A input.
Square Root	Updated description to add the value for C.
Clock Domain Crossing and Time Division Multiplexing	Updated description.
DSPCPLX	Updated information related to Language Templates.
Figure 40: Rounding and Saturation for Adder	Updated figure to add a flop stage in PL as well as in DSP2.
Figure 42: Floating-Point Time-Interleaved Dot-Product Engine	Updated to share data_dp0 between the two DSP58s at the bottom, as well as add a flop after pre-adder in the two DSP58s at the bottom. Also, staged FPINMODE.
Resource Utilization Guideline	Updated LUTRAM to SRL16/LUTRAM in the description. Updated figure to add signal RE from control to the coefficients block.
Single Multiplier MACC FIR Filter	Updated implementation description.
Symmetric MACC FIR Filter	Updated the FIR filter equation. Updated figure to add flop stages to A/B/D inputs of DSP58, as well as updated the input and output data width.
Three Multiplier Semi-Parallel FIR Filter	Updated description to add number of taps. Updated figure to add flop stages to inputs A and B for all the DSP58s and added WE3 in control logic to control the output flop.
Systolic FIR Filter	Added a note about swapping the A and B inputs in the systolic multiply-add processing element. Updated figure to add a note about C.
Symmetric Systolic FIR Filter	Updated equation.
Interpolating	Updated number of taps to 12. Updated figure to add a flop stage to input B.
Figure 51: Multichannel FIR Filter	Updated to add a flop stage to inputs A and B.
Decimating	Updated figure to add a flop stage to input B. Updated the equation for initial latency.
Floating Point FIR Filter	Updated description to add information about C in OPMODE.
Complex FIR Filter	Updated equations.
07/16/2020 Version 1.0	
Initial release.	N/A

Table of Contents

Revision History	2
Chapter 1: Overview	5
Introduction to Versal ACAP.....	5
Navigating Content by Design Process.....	6
DSP58 Architecture.....	7
Features and Functional Modes.....	7
Differences from Previous Generations.....	9
Device Resources.....	10
Recommended Design Flow.....	12
Chapter 2: DSP Resources	13
Design Entry.....	13
Primitives.....	14
Chapter 3: Scalar Fixed-Point ALU	36
Overview.....	36
DSP58 Features.....	37
Architectural Highlights of DSP58.....	40
DSP58 Operation Modes.....	43
Simplified DSP58 Operations	45
Chapter 4: Vector Fixed-Point ALU	71
Overview.....	71
Dot Product Unit.....	72
Pre-Adder Used as a Multiplexer.....	73
Chapter 5: Complex Arithmetic Unit	76
Basic Function and Complex Adder.....	76
Complex Multiplier.....	78
DSPCPLX Pipeline Configuration.....	79
Chapter 6: Floating-Point Arithmetic Unit	81

DSPFP32 Unisim Primitive.....	81
Operational Modes.....	82
Chapter 7: DSP58 Design Considerations.....	86
Design for Performance.....	86
Design for Power.....	86
Adder Tree versus Adder Cascade.....	86
Connecting DSP58s Across Columns.....	89
Time Multiplexing the DSP58.....	89
Notes and Suggestions.....	90
Pre-Adder Block Applications.....	90
Memory-Mapped I/O Register Application.....	91
Rounding.....	91
Overflow/Underflow/Saturation.....	93
Chapter 8: DSP58 Applications.....	95
Introduction.....	95
New Functional Mode Applications.....	95
Logic and Basic Math Application.....	96
Advanced Math Applications.....	101
Filter Designs.....	107
Appendix A: Additional Resources and Legal Notices.....	124
Xilinx Resources.....	124
Documentation Navigator and Design Hubs.....	124
References.....	124
Please Read: Important Legal Notices.....	125

Overview

Introduction to Versal ACAP

Versal™ adaptive compute acceleration platforms (ACAPs) combine Scalar Engines, Adaptable Engines, and Intelligent Engines with leading-edge memory and interfacing technologies to deliver powerful heterogeneous acceleration for any application. Most importantly, Versal ACAP hardware and software are targeted for programming and optimization by data scientists and software and hardware developers. Versal ACAPs are enabled by a host of tools, software, libraries, IP, middleware, and frameworks to enable all industry-standard design flows.

Built on the TSMC 7 nm FinFET process technology, the Versal portfolio is the first platform to combine software programmability and domain-specific hardware acceleration with the adaptability necessary to meet today's rapid pace of innovation. The portfolio includes six series of devices uniquely architected to deliver scalability and AI inference capabilities for a host of applications across different markets—from cloud—to networking—to wireless communications—to edge computing and endpoints.

The Versal architecture combines different engine types with a wealth of connectivity and communication capability and a network on chip (NoC) to enable seamless memory-mapped access to the full height and width of the device. Intelligent Engines are SIMD VLIW AI Engines for adaptive inference and advanced signal processing compute, and DSP Engines for fixed point, floating point, and complex MAC operations. Adaptable Engines are a combination of programmable logic blocks and memory, architected for high-compute density. Scalar Engines, including Arm® Cortex™-A72 and Cortex-R5F processors, allow for intensive compute tasks.

The Versal AI Core series delivers breakthrough AI inference acceleration with AI Engines that deliver over 100x greater compute performance than current server-class of CPUs. This series is designed for a breadth of applications, including cloud for dynamic workloads and network for massive bandwidth, all while delivering advanced safety and security features. AI and data scientists, as well as software and hardware developers, can all take advantage of the high-compute density to accelerate the performance of any application.

The Versal Prime series is the foundation and the mid-range of the Versal platform, serving the broadest range of uses across multiple markets. These applications include 100G to 200G networking equipment, network and storage acceleration in the Data Center, communications test equipment, broadcast, and aerospace & defense. The series integrates mainstream 58G transceivers and optimized I/O and DDR connectivity, achieving low-latency acceleration and performance across diverse workloads.

The Versal Premium series provides breakthrough heterogeneous integration, very high-performance compute, connectivity, and security in an adaptable platform with a minimized power and area footprint. The series is designed to exceed the demands of high-bandwidth, compute-intensive applications in wired communications, data center, test & measurement, and other applications. Versal Premium series ACAPs include 112G PAM4 transceivers and integrated blocks for 600G Ethernet, 600G Interlaken, PCI Express® Gen5, and high-speed cryptography.

The Versal architecture documentation suite is available at: <https://www.xilinx.com/versal>.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

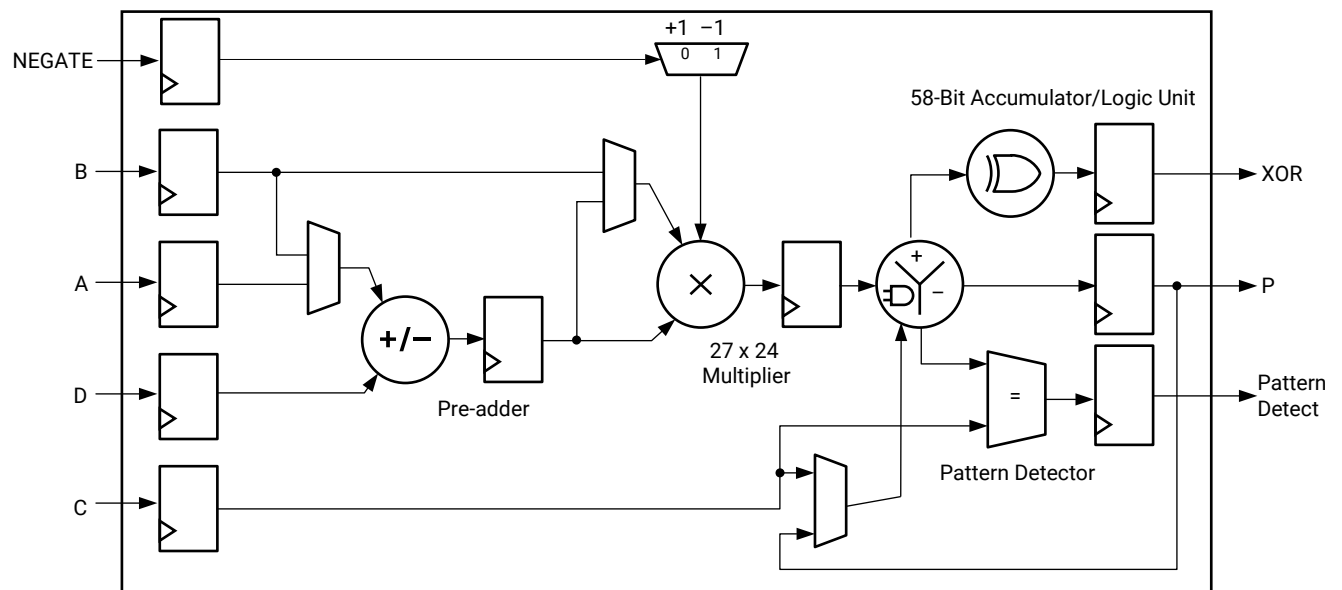
- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine. Topics in this document that apply to this design process include:
 - [Chapter 1: Overview](#): provides an overview of the DSP58 architecture and includes:
 - [DSP58 Architecture](#)
 - [Features and Functional Modes](#)
 - [Chapter 3: Scalar Fixed-Point ALU](#)
 - [Chapter 4: Vector Fixed-Point ALU](#)
 - [Chapter 5: Complex Arithmetic Unit](#)
 - [Chapter 6: Floating-Point Arithmetic Unit](#)
- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
 - [Chapter 8: DSP58 Applications](#): discusses design and implementation details of DSP58 and other new functional modes in video, wired/wireless, and networking applications.

DSP58 Architecture

Programmable logic devices are efficient for digital signal processing (DSP) applications because they can implement custom, fully parallel algorithms. DSP applications use many binary multipliers and accumulators that are best implemented in dedicated DSP resources.

Versal™ devices have many dedicated low-power DSPs combining high speed with small size while retaining system design flexibility. The DSP resources enhance the speed and efficiency of many applications beyond digital signal processing such as wide dynamic bus shifters, memory address generators, wide bus multiplexers, and memory-mapped I/O registers. The DSP Engine in the Versal architecture is defined using the DSP58 primitive.

Figure 1: DSP58 Simplified Block Diagram



X20248-061818

Features and Functional Modes

The DSP Engine can operate in a number of functional modes. Some highlights of the functionality include:

- $27 \times 24 + 58$ two's complement multiply-accumulator with 27-bit pre-addition and optional product negation.
- $18 \times 18 + 58$ two's complement complex multiply-accumulator using two back-to-back DSP58s, each of the two complex inputs can be optionally conjugated.
- Single-precision floating-point (binary32) accumulation.

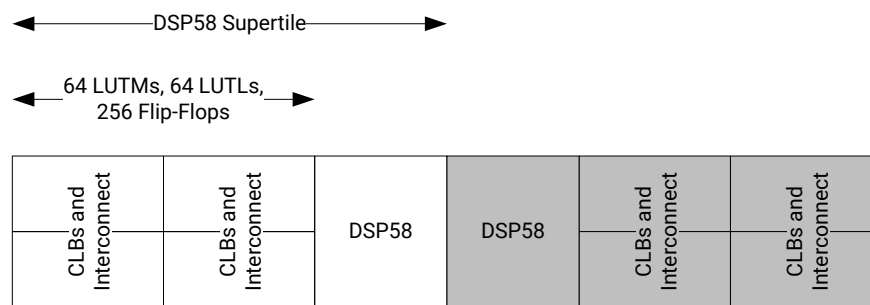
- Mixed-precision floating-point multiply-accumulator with multiplicand and multiplier statically and independently selectable to be either binary16 or binary32, and binary32 biasing and accumulation.
- Three-element two's complement vector dot product with accumulate or post-add in INT8 mode.
- Power saving 27-bit pre-adder that optimizes symmetrical filter applications and reduces DSP logic requirements.
- 58-bit accumulator that can be cascaded to build 116-bit and larger accumulators, adders, and counters.
- Single-instruction-multiple-data (SIMD) arithmetic unit with dual 24-bit or quad 12-bit add/subtract/accumulate.
- 58-bit logic unit: bitwise AND, OR, NOT, NAND, NOR, XOR, and XNOR.
- Pattern detector: terminal counts, overflow/underflow, convergent/symmetric rounding support, and 116-bit wide AND/NOR when combined with logic unit to detect if output matches a pattern.
- Optional pipeline registers and dedicated buses for cascading multiple DSP58s in a column for hierarchical/composite functions such as systolic FIR filters.

In the UltraScale™ architecture, two DSP48E2s with configurable logic blocks (CLBs) and block RAM form the DSP48 tile. The Versal™ architecture introduces the DSP58 supertile (see the following figure) which is made up of two rows and two columns of the new version of configurable logic block (CLBs) always next to a DSP58 to provide:

- 64 LUTMs: LUTs to be used as logic, distributed memory, or shift-register logic (SRL)
- 64 LUTLs to be used as logic resources
- 256 flip-flops

The new CLB contains exactly 50% LUTRAM/SRL capable LUTs to provide single port distributed SRAMs to the DSP. This structure can be replicated through the device to maximize ease of timing closure.

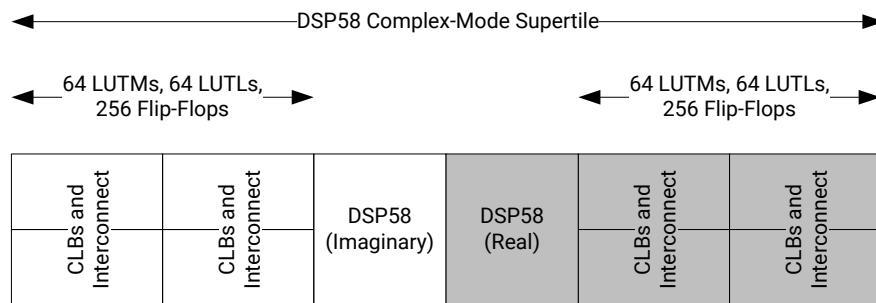
Figure 2: Two Back-to-Back DSP58 Supertiles



X20607-101118

The two back-to-back DSP58s form one complex arithmetic unit with their DSP_MODE attributes set to CINT18 (see the following figure). The right DSP58 in a dual-DSP58 complex arithmetic unit computes the real result P_{RE} . Concurrently, the left DSP58 computes the imaginary result P_{IM} . Shared signals (for example, CLK and ASYNC_RST) are routed only to the interconnect interface of the left DSP58.

Figure 3: One DSP58 Complex Mode Supertile



X20606-101118

Details on the various functional modes are provided in the following chapters.

Differences from Previous Generations

DSP58 is the sixth version of the Xilinx DSP. It is fully backwards compatible with the UltraScale™ architecture DSP48E2. DSP58 is a superset of the DSP48E2. In addition, Versal™ architecture DSP supports floating point operations and logic that interfaces with two back-to-back DSP58s to pair them as a tile-based 18-bit complex multiplier.

DSP58 INT8 Vector Dot Product Mode

- The INT8 multiplier mode is used to implement the dot product unit where the multiplier can be split into three smaller multipliers and their products are summed up to feed the post-adder. Each output of the smaller multipliers can be negated.

DSP58

- 27 × 24 multiplier:
 - B operand is increased from 18-bit to 24-bit.
- 58-bit logic unit:
 - C operand is increased from 48-bit to 58-bit.

- 116-bit wide XOR function (increased from 96-bit):
 - Wide XOR selectable for XOR12, XOR22 (new), XOR24, XOR34 (new), XOR58 (new), and XOR116 (new).

Note: XOR48 and XOR96 are supported when migrating from the UltraScale architecture.
- The A input is a 34-bit bus. The lower 27 bits feed the A input of the multiplier and the entire 34-bit input forms the upper 34 bits of the 58-bit A:B concatenate internal bus.
- The built-in right-shift becomes 23 bits wide.

Note: The 17-bit right-shift is supported when migrating from the UltraScale architecture.
- Multiplier output (X and Y together) sign can be changed by the negate pins.

DSPFP32 Mode

- Single precision floating-point multiplier and adder to produce both floating-point product and sum.
 - Multiplier:
 - Input can be either FP32 or FP16 and the output is always FP32.
 - Adder:
 - The input and output are both in FP32 only.

Note: FP32 is single precision floating-point number and FP16 is half precision floating-point number.

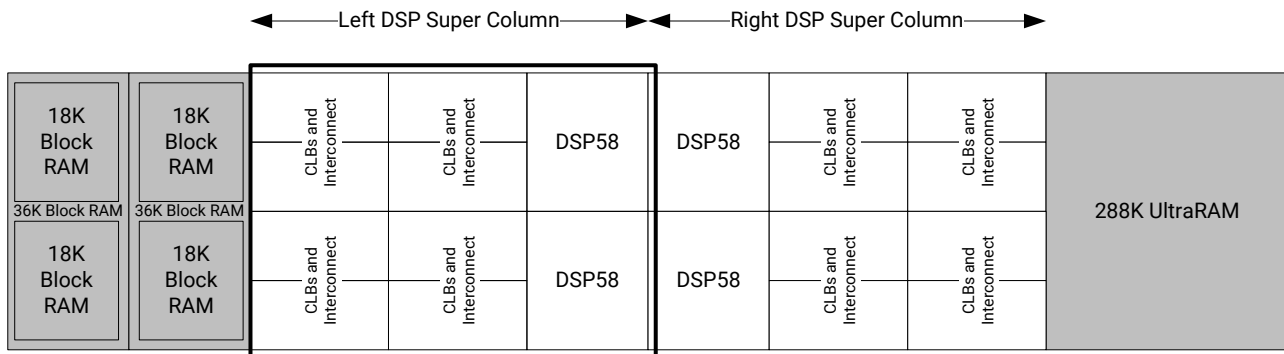
DSPCPLX Mode

- Two back-to-back DSP58s in the same tile can be used together to implement 18×18 complex multiply and accumulate.

Device Resources

The DSP resources are optimized and scalable across the Versal™ portfolio, providing a common architecture that improves implementation efficiency, IP implementation, and design migration. Migration within the Versal portfolio does not require any design changes to DSP58. When migrating from the UltraScale™ architecture to Versal architecture, because DSP58 is a superset of DSP48E2, an instantiation of the DSP48E2 is translated to DSP58.

The DSP super tiles stack vertically to form a DSP super column. The height of a DSP super tile is the same as two configurable logic blocks (CLBs). It matches both the height of one 18K block RAM and half a 288K UltraRAM. Two 18K block RAMs stack vertically to form a 36K block RAM (see the following figure).

Figure 4: DSP Super Columns and RAM Resources


X21266-051420

The Versal adaptive compute acceleration platform (ACAPs) DSP column has 48 DSP58s per clock region. There are 96 DSP58s per column per clock region because with the Versal architecture, the DSP58s always come in back-to-back pairs. DSP58s can be cascaded across clock regions up to the boundary of the device, or of a super logic region (SLR) in 3D ICs based on SSI technology. The number of cascadeable DSPs in a column can be found using the following Tcl command:

```
expr {[llength [get_sites DSP_X0* -of_objects [get_slrs SLR0]]] / 2}
```

The following table shows the maximum number of DSP58s that can be directly cascaded vertically in a column, and the total number of DSP58s for the Versal ACAPs.

Table 1: Maximum Number of Cascadeable DSP58s in Versal ACAPs

Device name	Max Cascade	Number of DSP58s
VM1102	116	472
VM1302	188	736
VM1402	188	1,504
VM1502	164	1,312
VM1802	164	1,968
VM2502	164	3,984
VM2602	188	1,880
VM2702	308	2,500
VM2902	308	3,080
VC1352	116	900
VC1502	164	1,312
VC1702	212	1,272
VC1802	164	1,600
VC1902	164	1,968

Recommended Design Flow

Many DSP58 designs are well suited for Versal™ ACAPs. To obtain best use of the architecture, underlying features, and capabilities must be understood so that the design entry code can take advantage of these resources. DSP resources are used automatically for most DSP functions and many arithmetic functions. In most cases, DSP resources must be inferred. See your preferred synthesis tool documentation for guidelines to ensure proper inference of the DSP. Instantiation of the DSP primitive can be used to directly access specific features. Recommendations for using DSP58 include:

- Use signed values in HDL source
- Pipeline for performance and lower power in DSP58 and programmable logic (PL)
- Use configurable logic block (CLB) shift register LUTs (SRLs), CLB distributed RAM, and/or block RAM to store filter coefficients
- Set USE_MULT to NONE when using only the adder/logic unit to save power
- Cascade using the dedicated resources rather than general-purpose interconnect, keeping usage to one column for highest performance and lowest power
- Consider using time multiplexing if resources are limited in a lower-speed application
- Use the CLB carry logic to implement small multipliers, adders, and counters

For more information on design techniques, see [Chapter 7: DSP58 Design Considerations](#).

DSP Resources

Design Entry

Xilinx offers integrated DSP design flows tailored for the unique needs of hardware, algorithm, and traditional processor-based DSP designers, supporting all mainstream DSP design entry methods to ensure productivity. Vivado® Design Suite includes an extensive library of device-optimized DSP IP to quickly assemble DSP designs that deliver high-quality results without requiring extensive programmable logic design experience. DSP algorithms implemented in RTL can be verified from within DSP specific simulation environments such as MATLAB®/Simulink® or C/C++. The DSP58s are inferred automatically from HDL code for most DSP functions and many arithmetic functions when using synthesis tools. Instantiation of the DSP58 primitive can be used to directly access specific features and provide more advanced user control.

Table 2: Design Entry Methods

Method	Support
Instantiation	Yes
Inference	Recommended
Vivado Design Suite IP catalog	Yes
Macros	Yes

DSP58 is a strict superset of the DSP48E2. When re-targeting from the UltraScale™ architecture, instantiation of the DSP48E2 is translated appropriately to the Versal architecture using the DSP48E5 internal primitive which is used by simulation and appears in a netlist. Code for inferring the following examples is provided by Xilinx.

- Fully pipelined 16×16 multiplier
- Fully pipelined 27×24 multiplier
- Multiply add
- 16-bit adder
- 16-bit adder, the same value on both inputs to the adder
- Loadable multiply
- Complex 18×18 multiplier mapping to one DSPCPLX unit

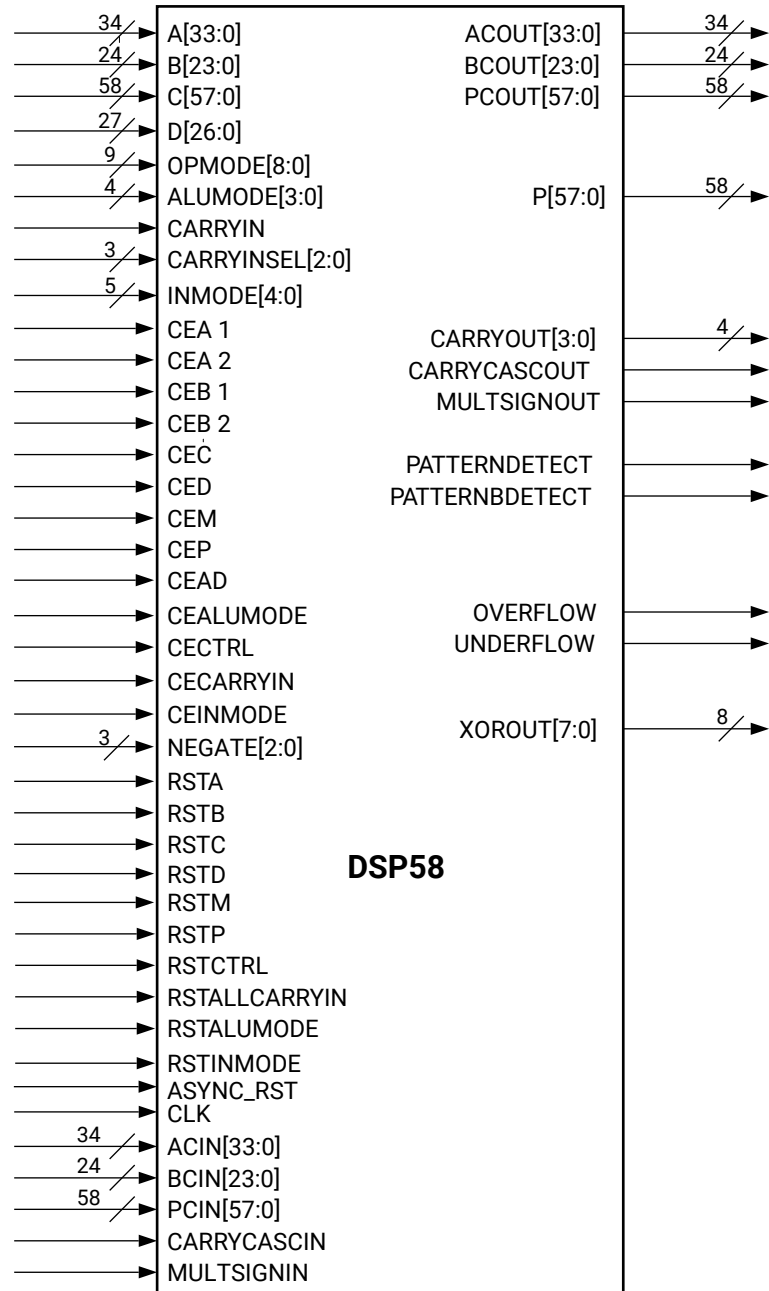
- $3 \times$ dot product of 9-bit and 8-bit two's complement fixed point numbers

Note: Inference for DSPFP32 (single and half precision) is not supported. It can be instantiated and it is recommended that customers use the Floating Point Operator IP core in the Vivado IP catalog to implement the function.

Primitives

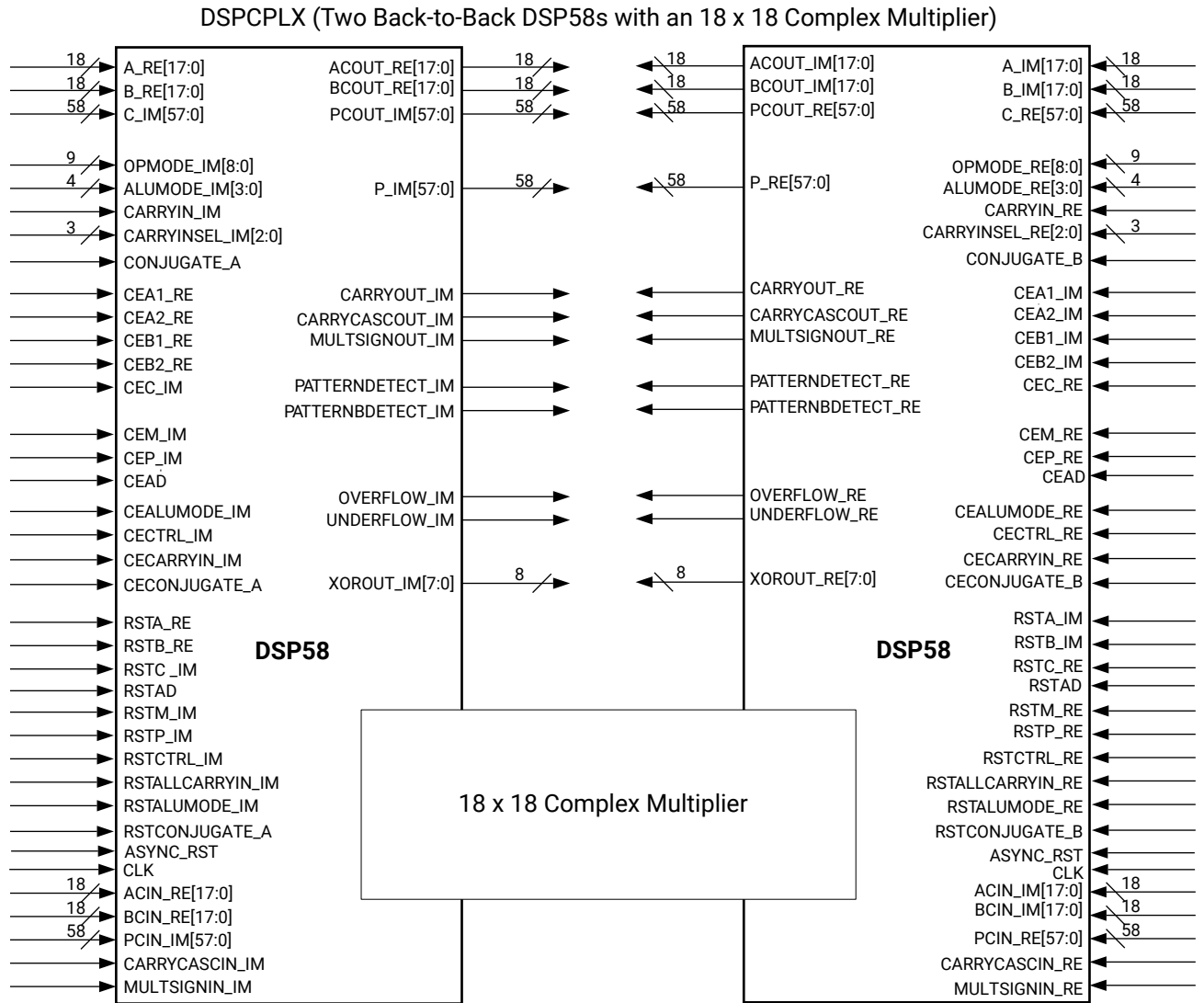
The following figures show the DSP58, DSPCPLX, and DSPFP32 primitives. Each primitive shows the input and output pins along with the bit widths of each port. The port descriptions are consolidated in [Table 3](#) and the attribute descriptions are consolidated in [Table 4](#).

Figure 5: DSP58 Primitive



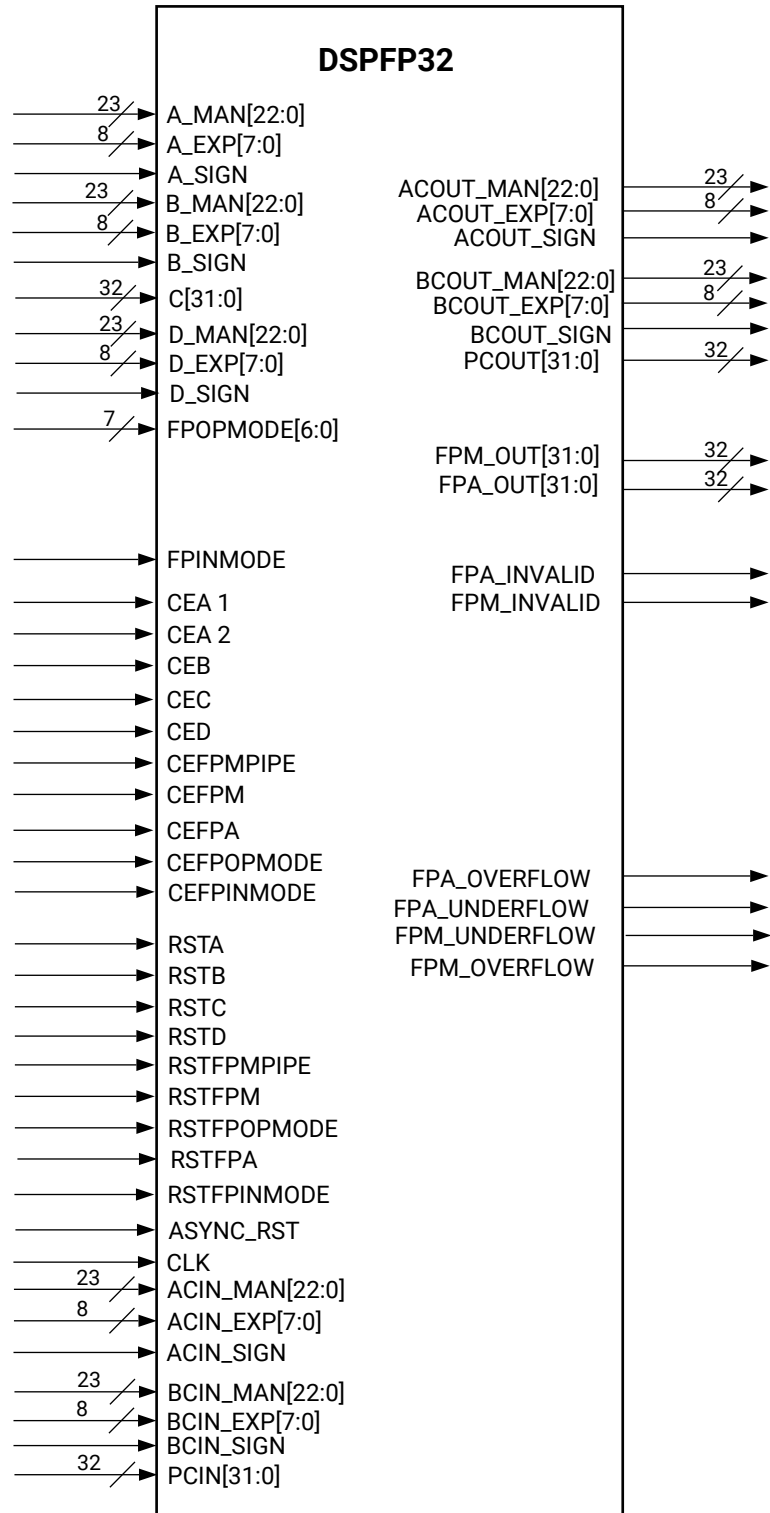
X20249-100318

Figure 6: DSPCPLX Primitive



X20608-101118

Figure 7: DSPFP32 Primitive



X20610-100318

Port Descriptions

Table 3: Port Descriptions

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
A ¹	In	34	A[33:0]	A_IM[17:0] ² A_RE[17:0]	A_SIGN A_EXP[7:0] A_MAN[22:0]	A[26:0] is the A input of the multiplier or the A input of the pre-adder. A[33:0] are the most significant bits (MSBs) of the A:B concatenated input to the second-stage adder/subtractor or logic function. In INT8 mode (dot-product), port A holds three 9-bit two's complement values or three unsigned 8-bit values by setting sign bits: A[26], A[17], and A[8] to 0.
ACIN	In	34	ACIN[33:0]	ACIN_IM[17:0] ACIN_RE[17:0]	ACIN_SIGN ACIN_EXP[7:0] ACIN_MAN[22:0]	Cascaded data input from ACOUT of previous DSP58 (multiplexed with A).
ACOUT	Out	34	ACOUT[33:0]	ACOUT_IM[17:0] ACOUT_RE[17:0]	ACOUT_SIGN ACOUT_EXP[7:0] ACOUT_MAN[22:0]	Cascaded data output to ACIN of the next DSP.
ALUMODE	In	4	ALUMODE[3:0]	ALUMODE_IM[3:0] ALUMODE_RE[3:0]	N/A	Controls the selection of the logic and arithmetic function in the second stage add/sub/logic unit of the DSP.

Table 3: Port Descriptions (cont'd)

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
B	In	24	B[23:0]	B_IM[17:0] B_RE[17:0]	B_SIGN B_EXP[7:0] B_MAN[22:0]	The B input of the multiplier or the B input of the preadder. B[23:0] are the least significant bits (LSBs) of the A:B concatenated input to the second-stage adder/subtractor or logic function. In INT8 mode (dot-product), port B holds three 8-bit two's complement values with sign bits B[23], B[15] and B[7].
BCIN	In	24	BCIN[23:0]	BCIN_IM[17:0] BCIN_RE[17:0]	BCIN_SIGN BCIN_EXP[7:0] BCIN_MAN[22:0]	Cascaded data input from BCOUT of the previous DSP (multiplexed with B).
BCOUT	Out	24	BCOUT[23:0]	BCOUT_IM[17:0] BCOUT_RE[17:0]	BCOUT_SIGN BCOUT_EXP[7:0] BCOUT_MAN[22:0]	Cascaded data output to BCIN of the next DSP58.
C	In	58	C[57:0]	C_IM[57:0] C_RE[57:0]	C[31:0]	Data input to the second-stage adder/subtractor, pattern detector, or logic function.
CARRYCASCIN	In	1	CARRYCASCIN	CARRYCASCIN_IM CARRYCASCIN_RE	N/A	Cascaded carry input from CARRYCASCOUT of the previous DSP58.
CARRYCASCOUT	Out	1	CARRYCASCOUT	CARRYCASCOUT_IM CARRYCASCOUT_RE	N/A	Cascaded carry output to CARRYCASCIN of the next DSP. This signal is internally fed back into the CARRYINSEL multiplexer input of the same DSP.
CARRYIN	In	1	CARRYIN	CARRYIN_IM CARRYIN_RE	N/A	Carry input from the logic.
CARRYINSEL	In	3	CARRYINSEL[2:0]	CARRYINSEL_IM[2:0] CARRYINSEL_RE[2:0]	N/A	Selects the carry source.

Table 3: Port Descriptions (cont'd)

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
CARRYOUT	Out	4	CARRYOUT[3:0]	CARRYOUT_IM CARRYOUT_RE	FPA_INVALID, FPM_INVALID	4-bit carry output from each 12-bit field of the accumulate/adder/logic unit. Normal 58-bit operation uses only CARRYOUT[3]. Only the SIMD mode FOUR12 can use all the four CARRYOUT bits (CARRYOUT[3:0]). In DSPFP32 mode: FPA_INVALID is mapped to CARRYOUT[1] FPM_INVALID is mapped to CARRYOUT[0] In DSPCPLX mode CARRYOUT[3] is used as CARRYOUT_RE and CARRYOUT_IM
CEA1	In	1	CEA1	CEA1_IM CEA1_RE	CEA1	Clock enable for the first A (input) register. A1 is only used if AREG = 2 or INMODE[0] = 1. INMODE control is for multiplier only.
CEA2	In	1	CEA2	CEA2_IM CEA2_RE	CEA2	Clock enable for the second A (input) register. A2 is only used if AREG = 1 or 2 and INMODE[0] = 0. INMODE control is for multiplier only.
CEAD	In	1	CEAD	CEAD	N/A	Clock enable for the pre-adder output AD pipeline register.
CEALUMODE	In	1	CEALUMODE	CEALUMODE_IM CEALUMODE_RE	CEFPA	Clock enable for ALUMODE (control inputs) registers. In DSPFP32 this is the clock enable for the FPA output registers.

Table 3: Port Descriptions (cont'd)

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
CEB1	In	1	CEB1	CEB1_IM CEB1_RE	CEB	Clock enable for the first B (input) register. B1 is only used if BREG = 2 or INMODE[4] = 1. In DSPFP32 there is only one B input so CEB1 is renamed CEB. INMODE control is for multiplier only.
CEB2	In	1	CEB2	CEB2_IM CEB2_RE	N/A	Clock enable for the second B (input) register. B2 is only used if BREG = 1 or 2 and INMODE[4] = 0. In DSPFP32 there is only one B input so no CEB2 pin is available. INMODE control is for multiplier only.
CEC	In	1	CEC	CEC_IM CEC_RE	CEC	Clock enable for the C (input) register. In DSPFP32, CEC enables all C register stages configured by the FPCREG attribute.
CECARRYIN	In	1	CECARRYIN	CECARRYIN_IM CECARRYIN_RE	N/A	Clock enable for the CARRYIN (input from the logic) register.
CECTRL	In	1	CECTRL	CECTRL_IM CECTRL_RE	CEFPOPMODE	Clock enable for the OPMODE and CARRYINSEL (control inputs) registers.
CED	In	1	CED	N/A	CED	Clock enable for the D (input) register.
CEINMODE	In	1	CEINMODE	CECONJUGATE_A CECONJUGATE_B	CEFPINMODE	Clock enable for the INMODE control input registers. Also the clock enable for NEGATE control input registers in DSP58 mode and for CONJUGATE_A/B input registers in DSPCPLX mode.

Table 3: Port Descriptions (cont'd)

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
CEM	In	1	CEM	CEM_IM CEM_RE	CEFPMPPIPE	Clock enable for the post-multiply M (pipeline) register and the internal multiply CARRYIN register (DSP58, DSPCPLX only).
CEP	In	1	CEP	CEP_IM CEP_RE	CEFPM	Clock enable for the P output register in DSP58/ DSPCPLX and FPM output register in DSPFP32.
CLK	In	1	CLK	CLK	CLK	The DSP58 input clock, common to all internal registers and flip-flops.
D	In	27	D[26:0]	N/A	D_SIGN D_EXP[7:0] D_MAN[22:0]	27-bit input to the pre-adder. The pre-adder implements $D \pm A$ or $D \pm B$ as determined by the attribute PREADDINSEL. The INMODE[3] signal determines whether the pre-adder is performing an addition or subtraction. In DSPFP32, this port is an alternative input to the floating-point multiplier (binary16 or binary32), or binary32 input to the standalone binary 32 adder.

Table 3: Port Descriptions (cont'd)

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
INMODE	In	5	INMODE[4:0]	CONJUGATE_A CONJUGATE_B	FPINMODE	These five control bits select the functionality of the pre-adder, the A, B, and D inputs, and the input registers. These bits should be tied to GND if unused. In DSPCPLX INMODE[3] only is used and mapped to CONJUGATE. In DSPFP32 INMODE[4] is mapped to FPINMODE and controls input MUX selection between B and D input ports to FP Multiplier.
MULTSIGNIN	In	1	MULTSIGNIN	MULTSIGNIN_IM MULTSIGNIN_RE	N/A	Signal from the previous DSP for MACC extension.
MULTSIGNOUT	Out	1	MULTSIGNOUT	MULTSIGNOUT_IM MULTSIGNOUT_RE	N/A	Signal cascaded to the next DSP for MACC extension.
NEGATE	In	3	NEGATE[2:0]	N/A	N/A	Select if the multiplier input needs to be negated. In DSPCPLX, the CONJUGATE inputs connected to INMODE[3] is used. For DSP58, in INT24 mode, only NEGATE[0] is used. In INT8 mode, all NEGATE bits are used.
OPMODE	In	9	OPMODE[8:0]	OPMODE_IM[8:0] OPMODE_RE[8:0]	FPOPMODE[6:0]	Controls the input to the W, X, Y, and Z multiplexers in DSP58 and DSPCPLX Unisims only. In DSPFP32 the lower 7 bits control the P0 and P1 inputs to the floating point adder.

Table 3: Port Descriptions (cont'd)

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
OVERFLOW	Out	1	OVERFLOW	OVERFLOW_IM OVERFLOW_RE	FPA_OVERFLOW	Overflow indicator when used with the appropriate setting of the pattern detector. In DSPFP32, this flag indicates overflow of the floating-point adder.
P	Out	58	P[57:0]	P_IM[57:0] P_RE[57:0]	{FPM_OUT [25:0] FPA_OUT[31:0]}	Data output from second stage adder/subtractor or logic. In DSPFP32, bits [57:32] are mapped to floating point multiplier (FPM), and bits [31:0] are mapped to the output of the floating point adder (FPA).
PATTERNBDETECT	Out	1	PATTERNBDETECT	PATTERNBDETECT_IM PATTERNBDETECT_RE	N/A	Match indicator between P[57:0] and the complement of the pattern.
PATTERNDETECT	Out	1	PATTERNDETECT	PATTERNDETECT_IM PATTERNDETECT_RE	N/A	Match indicator between P[57:0] and the pattern.
PCIN	In	58	PCIN[57:0]	PCIN_IM[57:0] PCIN_RE[57:0]	PCIN[31:0]	Cascaded data input from PCOUT of the previous DSP58 to ALU. In floating-point mode, only the lower 32 bits are used and the upper 26 bits are set to zero.
PCOUT	Out	58	PCOUT[57:0]	PCOUT_IM[57:0] PCOUT_RE[57:0]	PCOUT[31:0]	Cascaded data output to PCIN of the next DSP58. In DSPFP32, only the lower 32 bits are used.
RSTA	In	1	RSTA	RSTA_IM RSTA_RE	RSTA	Reset for both A (input) registers.

Table 3: Port Descriptions (cont'd)

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
RSTALLCARRYIN	In	1	RSTALLCARRYIN	RSTALLCARRYIN_IM RSTALLCARRYIN_RE	N/A	Reset for the Carry (internal multiply round) and the CARRYIN register in all fixed-point modes.
RSTALUMODE	In	1	RSTALUMODE	RSTALUMODE_IM RSTALUMODE_RE	RSTFPA	Reset for ALUMODE (control inputs) registers. In DSPFP32 acts as reset for FPA output registers
RSTB	In	1	RSTB	RSTB_IM RSTB_RE	RSTB	Reset for both B (input) registers.
RSTC	In	1	RSTC	RSTC_IM RSTC_RE	RSTC	Reset for the C (input) register.
RSTCTRL	In	1	RSTCTRL	RSTCTRL_IM RSTCTRL_RE	RSTFPOPMODE	Reset for OPMODE and CARRYINSEL (control inputs) registers.
RSTD	In	1	RSTD	N/A	RSTD	Reset for the D (input) register.
RSTAD	In	1	N/A	RSTAD	N/A	Reset for the pre-adder (output) AD pipeline register.
RSTINMODE	In	1	RSTINMODE	RSTINMODE_IM RSTINMODE_RE	RSTFPINMODE	Reset for the INMODE (control input) registers.
RSTM	In	1	RSTM	RSTM_IM RSTM_RE	RSTFPMPIPE	Reset for the M (pipeline) register.
RSTP	In	1	RSTP	RSTP_IM RSTP_RE	RSTFPM	Reset for P output registers in DSP58 and DSPCPLX, and reset for FPM output registers in DSPFP32.
ASYNC_RST	In	1	ASYNC_RST	ASYNC_RST	ASYNC_RST	Asynchronous reset for all registers. Input only valid when attribute RESET_MODE = ASYNC.

Table 3: Port Descriptions (cont'd)

DSP Tile Pin	Direction	Bus Width (Default for DSP58)	DSP58 Unisim Pin	DSPCPLX Unisim Pin	DSPFP32 Unisim Pin	Description
UNDERFLOW	Out	1	UNDERFLOW	UNDERFLOW_IM UNDERFLOW_RE	FPA_UNDERFLOW	Underflow indicator when used with the appropriate setting of the pattern detector. In DSPFP32, this flag indicates underflow of the floating-point adder.
XOROUT	Out	8	XOROUT[7:0]	XOROUT_IM[7:0] XOROUT_RE[7:0]	{FPM_UNDERFLOW FPM_OVERFLOW, FPM_OUT [31:26]}	Data output from wide XOR function. In DSPFP32: XOROUT[5:0] are mapped to FPMOUT[31:26], XOROUT[6] is mapped to overflow status port for FP multiplier and XOROUT[7] is mapped to the underflow status port of the FP multiplier.

Notes:

1. The HW PortName is the general signal name. The specific pin name in each mode including the bus width is specified in the columns under DSP58, DSPCPLX, and DSPFP32.
2. The ports from both DSPs are presented as unique ports in DSPCPLX. Naming convention is <Portname>_IM (imaginary) and <Portname>_RE (real). The exception to this convention are the CONJUGATE ports, which are differentiated as CONJUGATE_A and CONJUGATE_B.

Attributes

The synthesis attributes for the DSP(s) in various modes are described in this section. The attributes call out pipeline registers in the control and datapaths. The value of the attribute sets the number of pipeline registers.

Table 4: Attribute Setting Description

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
Register Control Attributes				
ACASCREG	0, 1, 2 (1) ²			Selects the number of A input registers on the A cascade path, ACOUT. This attribute must be equal to or one less than the AREG value: AREG = 0: ACASCREG must be 0 AREG = 1: ACASCREG must be 1 AREG = 2: ACASCREG can be 1 or 2
ADREG	0, 1 (1)	N/A	0, 1 (1)	Selects the number of AD pipeline registers. Because the common pre-adder output in CINT18 mode is registered in the AD registers of both DSPs, only a single ADREG attribute is used in CINT18.
ALUMODEREG	0, 1 (1)	N/A	0, 1 (1)	Selects the number of ALUMODE input registers.
AREG	0, 1, 2 (1)			Selects the number of A input registers to the X multiplexer to the ALU or multiplier. For ALU, when 1 is selected, the A2 register is used. For multiplier, when 1 is selected and INMODE[0] = 1, A1 register is used.
BCASCREG	0, 1, 2 (1)	N/A	0, 1, 2 (1)	Selects the number of B input registers on the B cascade path, BCOUT. This attribute must be equal to or one less than the BREG value: BREG = 0: BCASCREG must be 0 BREG = 1: BCASCREG must be 1 BREG = 2: BCASCREG can be 1 or 2 (1 only in floating-point mode)

Table 4: Attribute Setting Description (cont'd)

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
BREG	0, 1, 2 (1)	N/A	0, 1, 2 (1)	Selects the number of B input registers to the X multiplexer to the ALU or multiplier. For ALU, when 1 is selected, the B2 register is used. For multiplier, when 1 is selected and INMODE[4] = 1, B1 register is used.
CARRYINREG	0, 1 (1)	N/A	0, 1 (1)	Selects the number of programmable logic (PL) CARRYIN input registers.
CARRYINSELREG	0, 1 (1)	N/A	0, 1 (1)	Selects the number of CARRYINSEL input registers.
CREG	0, 1 (1)	N/A	0, 1 (1)	Selects the number of C input registers.
DREG	0, 1 (1)	N/A	N/A	Selects the number of D input registers.
DSP58/DSPFP32:INMODEREG DSPCPLX:CONJUGATEREG_A/ CONJUGATEREG_B	0, 1 (1)	0, 1 (1)	0, 1 (1)	Selects the number of INMODE and NEGATE input registers. In DSPCPLX, the attribute is COJUGATEREG_A and CONJUGATEREG_B.
MREG	0, 1 (1)	N/A	0, 1 (1)	Selects the number of M pipeline registers.
OPMODEREG	0, 1 (1)	N/A	0, 1 (1)	Selects the number of OPMODE input registers.
RESET_MODE	SYNC, ASYNC (SYNC)			Selects if the enabled registers in the DSP are reset by their register specific synchronous resets or the common ASYNC_RST.
DSP58, DSPCPLX: PREG DSPFP32: FPA_PREG, FPM_PREG	0, 1 (1)			Selects the number of P output registers in non-floating-point mode (also used by CARRYOUT, PATTERNDETECT, PATTERNBDETECT, OVERFLOW, UNDERFLOW, XOROUT, CARRYCASCOUT, MULTSIGNOUT, and PCOUT). In DSPFP32, FPM_PREG and FPA_PREG select the identical number of registers for FPM and FPA respectively.

Table 4: Attribute Setting Description (cont'd)

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
FPBREG	N/A	0, 1 (1)	N/A	Selects number of B input registers in DSPFP32.
FPCREG	N/A	0, 1, 2, 3 (3)	N/A	Selects number of C input registers in DSPFP32.
FPDREG	N/A	0, 1 (1)	N/A	Selects number of D input registers in DSPFP32.
FPOPMREG	N/A	0, 1, 2, 3 (3)	N/A	Select number of OPMODE input registers in DSPFP32.
FPMPIPEREG	N/A	0, 1 (1)	N/A	Select number of M registers in DSPFP32 mode.
Feature Control Attributes				
DSP_MODE	INT24, INT8 (INT24)	(read only)	CINT18	This attribute configures the DSP for a particular mode of operation. INT24 is for the 27 × 24 fixed-point ALU and also for the legacy mode. INT8 is for the three-element 9 × 8 vector dot-product mode.
A_INPUT	DIRECT, CASCADE (DIRECT)			Selects the A input between parallel input (DIRECT) or the cascaded input from the previous DSP (CASCADE).
B_INPUT	DIRECT, CASCADE (DIRECT)			Selects the B input between parallel input (DIRECT) or the cascaded input from the previous DSP (CASCADE).
BCASCSEL	N/A	B, D (B)	N/A	Selects cascade out data in DSPFP32 mode.
PCOUTSEL	N/A	FPM, FPA (FPA)	N/A	Select P cascade output data.
PREADDINSEL	A, B (A)	N/A	N/A	Selects the input to be added/subtracted with D in the pre-adder.
AMULTSEL	A, AD (A)	N/A	N/A	Selects the input to the 27-bit A input of the multiplier.
BMULTSEL	B, AD (B)	N/A	N/A	Selects the input to the 24-bit B input of the multiplier.

Table 4: Attribute Setting Description (cont'd)

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
A_FPTYPE	N/A	B16, B32 (B32)	N/A	Selects floating-point data type for A. B16 is for binary16 (half-precision) and B32 is for binary32 (single-precision).
B_D_FPTYPE	N/A	B16, B32 (B32)	N/A	Selects floating-point data type for B and D for multiplication. B16 is for binary16 (half-precision) and B32 is for binary32 (single-precision). Note: When set to B16, D cannot be sent directly to P1 for binary32 addition. It can be first multiplied by A = 1 and then sent to P0 as FPM for binary32 addition.
USE_MULT	NONE, MULTIPLY (MULTIPLY)	NONE, MULTIPLY (MULTIPLY)	N/A	Selects usage of the multiplier. Set to NONE to save power when using only the Adder/Logic Unit in DSP58 or floating-point modes.
RND	58-bit field (00...00)	N/A	58-bit field (00...00)	This 58-bit value is used as the Rounding Constant into the WMUX.
USE_SIMD	ONE58, TWO24, FOUR12 (ONE58)	N/A		Selects the mode of operation for the adder/subtractor. The attribute setting can be one 58-bit adder mode (ONE58), two 24-bit adder mode (TWO24), or four 12-bit adder mode (FOUR12). Typical Multiply-Add operations are supported when the mode is set to ONE58. When either TWO24 or FOUR12 mode is selected, the multiplier must not be used, and USE_MULT must be set to NONE.
USE_WIDEXOR	TRUE, FALSE (FALSE)	N/A		Determines whether the wide XOR is used or not used.

Table 4: Attribute Setting Description (cont'd)

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
XORSIMD	XOR12_22, XOR24_34_58_116 (XOR24_34_58_116)	N/A	N/A	Selects the mode of operation for the wide XOR. The attribute setting can be one 116-bit, two 58-bit, two 24-bit and two 34-bit XOR mode (XOR24_34_58_116), or six 12-bit and two 22-bit XOR mode (XOR12_22).
Pattern Detector Attributes				
AUTORESET_PATDET	NO_RESET, RESET_MATCH, RESET_NOT_MATCH (NO_RESET)	N/A	NO_RESET, RESET_MATCH, RESET_NOT_MATCH (NO_RESET)	Automatically resets the P register (accumulated value or counter value) on the next clock cycle, if a pattern detect event has occurred on this clock cycle. The RESET_MATCH and RESET_NOT_MATCH settings distinguish between whether the DSP58 must cause an auto reset of the P register on the next cycle: <ul style="list-style-type: none"> when the pattern is matched or whenever the pattern is not matched on the current cycle but was matched on the previous clock cycle
AUTORESET_PRIORITY	RESET, CEP (RESET)	N/A	RESET, CEP (RESET)	When using the AUTORESET_PATDET feature, if the attribute is set to CEP, the P register only resets the pending value of the clock enable. Otherwise, the autoreset will have precedence.
MASK	58-bit field (0011...11)	N/A	58-bit field (0011...11)	This 58-bit value is used to mask out certain bits during a pattern detection. When a MASK bit is set to 1, the corresponding pattern bit is ignored. When a MASK bit is set to 0, the pattern bit is compared.
PATTERN	58-bit field (00...00)	N/A	58-bit field (00...00)	This 58-bit value is used in the pattern detector.

Table 4: Attribute Setting Description (cont'd)

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
SEL_MASK	MASK, C, ROUNDING_MODE1, ROUNDING_MODE2 (MASK)	N/A	MASK, C, ROUNDING_MODE1, ROUNDING_MODE2 (MASK)	Selects the mask to be used for the pattern detector. The C and MASK settings are for standard uses of the pattern detector (counter, overflow detection, etc.). ROUNDING_MODE1 (C-bar left shifted by 1) and ROUNDING_MODE2 (C-bar left shifted by 2) select special masks based off of the optionally registered C input. These rounding modes can be used to implement convergent rounding in the DSP58 using the pattern detector.
SEL_PATTERN	PATTERN, C (PATTERN)	N/A	PATTERN, C (PATTERN)	Selects the input source for the pattern field. The input source can either be a 58-bit dynamic C input or a 58-bit static attribute field.
USE_PATTERN_DETECT	NO_PATDET, PATDET (NO_PATDET)	N/A	NO_PATDET, PATDET (NO_PATDET)	Selects whether the pattern detector and related features, including overflow and underflow, are used (PATDET) or not used (NO_PATDET). This attribute is used for speed specification and Simulation Model purposes only.
Optional Inversion Attributes				
IS_ALUMODE_INVERTED	4-bit binary (4'b0000)	N/A	4-bit binary (4'b0000)	Indicates if the ALUMODE[3:0] is optionally inverted within the DSP. The default 4'b0000 indicates that all bits of the ALUMODE bus are not inverted. Each attribute bit controls its respective bit of the ALUMODE bus.
IS_ASYNC_RST_INVERTED	1-bit binary (1'b0)			Indicates if the ASYNC_RST is optionally inverted within the DSP. The default 1'b0 indicates that the ASYNC_RST is not inverted.

Table 4: Attribute Setting Description (cont'd)

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
IS_CARRYIN_INVERTED	1-bit binary (1'b0)	N/A	1-bit binary (1'b0)	Indicates if the CARRYIN is optionally inverted within the DSP. The default 1'b0 indicates that the CARRYIN is not inverted.
IS_CLK_INVERTED	1-bit binary (1'b0)			Indicates if the CLK is optionally inverted within the DSP. The default 1'b0 indicates that the CLK is not inverted.
IS_INMODE_INVERTED IS_FPINMODE_INVERTED IS_CONJUGATE_INVERTED	5-bit binary (5'b00000)	1-bit binary (1'b0)	1-bit binary (1'b0)	Indicates if the INMODE[4:0] is optionally inverted within the DSP. The default 5'b00000 indicates that all the bits of the INMODE bus are not inverted. Each Attribute bit controls its respective bit of the INMODE bus. In DSPFP32, the corresponding pin is FPINMODE. In DSPCPLX, the corresponding pins are CONJUGATE_A and CONJUGATE_B.
IS_NEGATE_INVERTED	3-bit binary (1'b000)	N/A	N/A	Indicates if all the bits of NEGATE are optionally inverted within the DSP. The default 3'b000 indicates that the NEGATE[2:0] is not inverted.
IS_OPMODE_INVERTED IS_FPOPMODE_INVERTED	9-bit binary (9'b000000000)	7-bit binary (7'b0000000)	9-bit binary (9'b000000000)	For DSP58 and DSPCPLX, indicates if the OPMODE[8:0] is optionally inverted within the DSP. The default 9'b000000000 indicates that all the bits of the OPMODE bus are not inverted. Each attribute bit controls its respective bit of the OPMODE bus. In DSPFP32, the corresponding pins are the FPOPMODE bus (7 bits).
IS_RSTA_INVERTED	1-bit binary (1'b0)			Indicates if the RSTA is optionally inverted within the DSP. The default 1'b0 indicates that the RSTA is not inverted.

Table 4: Attribute Setting Description (cont'd)

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
IS_RSTALLCARRYIN_INVERTED	1-bit binary (1'b0)	N/A	1-bit binary (1'b0)	Indicates if the RSTALLCARRYIN is optionally inverted within the DSP. The default 1'b0 indicates that the RSTALLCARRYIN is not inverted.
IS_RSTALUMODE_INVERTED IS_RSTFPA_INVERTED	1-bit binary (1'b0)			Indicates if the RSTALUMODE is optionally inverted within the DSP. The default 1'b0 indicates that the RSTALUMODE is not inverted. In DSPFP32, the attribute IS_RSTFPA_INVERTED corresponds to the pin RSTFPA.
IS_RSTB_INVERTED	1-bit binary (1'b0)			Indicates if the RSTB is optionally inverted within the DSP. The default 1'b0 indicates that the RSTB is not inverted.
IS_RSTC_INVERTED	1-bit binary (1'b0)			Indicates if the RSTC is optionally inverted within the DSP. The default 1'b0 indicates that the RSTC is not inverted.
IS_RSTCTRL_INVERTED IS_RSTFPOPMODE_INVERTED	1-bit binary (1'b0)			First attribute indicates if the RSTCTRL is optionally inverted within the DSP. The default 1'b0 indicates that the RSTCTRL is not inverted. Second attribute is for RSTFPOPMODE pin in DSPFP32.
IS_RSTD_INVERTED IS_RSTAD_INVERTED	1-bit binary (1'b0)			IS_RSTD_INVERTED Indicates if the RSTD is optionally inverted within the DSP. The default 1'b0 indicates that the RSTD is not inverted. In DSPCPLX, the attribute IS_RSTAD_INVERTED corresponds to the pin RSTAD.

Table 4: Attribute Setting Description (cont'd)

Attribute Name	Settings (Default)			Description
	DSP58	DSPFP32	DSPCPLX ¹	
IS_RSTINMODE_INVERTED IS_RSTFPINMODE_INVERTED IS_RSTCONJUGATE_INVERTED	1-bit binary (1'b0)			The first attribute indicates if the RSTINMODE is optionally inverted within the DSP. The default 1'b0 indicates that the RSTINMODE is not inverted. IS_RSTFPINMODE_INVERTED attribute is for RSTFPINMODE pin in DSPFP32. IS_RSTCONJUGATE_INVERTED is for RSTCONJUGATE pin in DSPCPLX.
IS_RSTM_INVERTED IS_RSTFPMPIPE_INVERTED	1-bit binary (1'b0)			Indicates if the RSTM is optionally inverted within the DSP. The default 1'b0 indicates that the RSTM is not inverted. The attribute IS_RSTFPMPIPE_INVERTED corresponds to RSTFPMPIPE pin in DSP32.
IS_RSTP_INVERTED IS_RSTFPM_INVERTED	1-bit binary (1'b0)			Indicates if the RSTP is optionally inverted within the DSP. The default 1'b0 indicates that the RSTP is not inverted. The attribute IS_RSTFPM_INVERTED corresponds to RSTFPM pin in DSPFP32.

Notes:

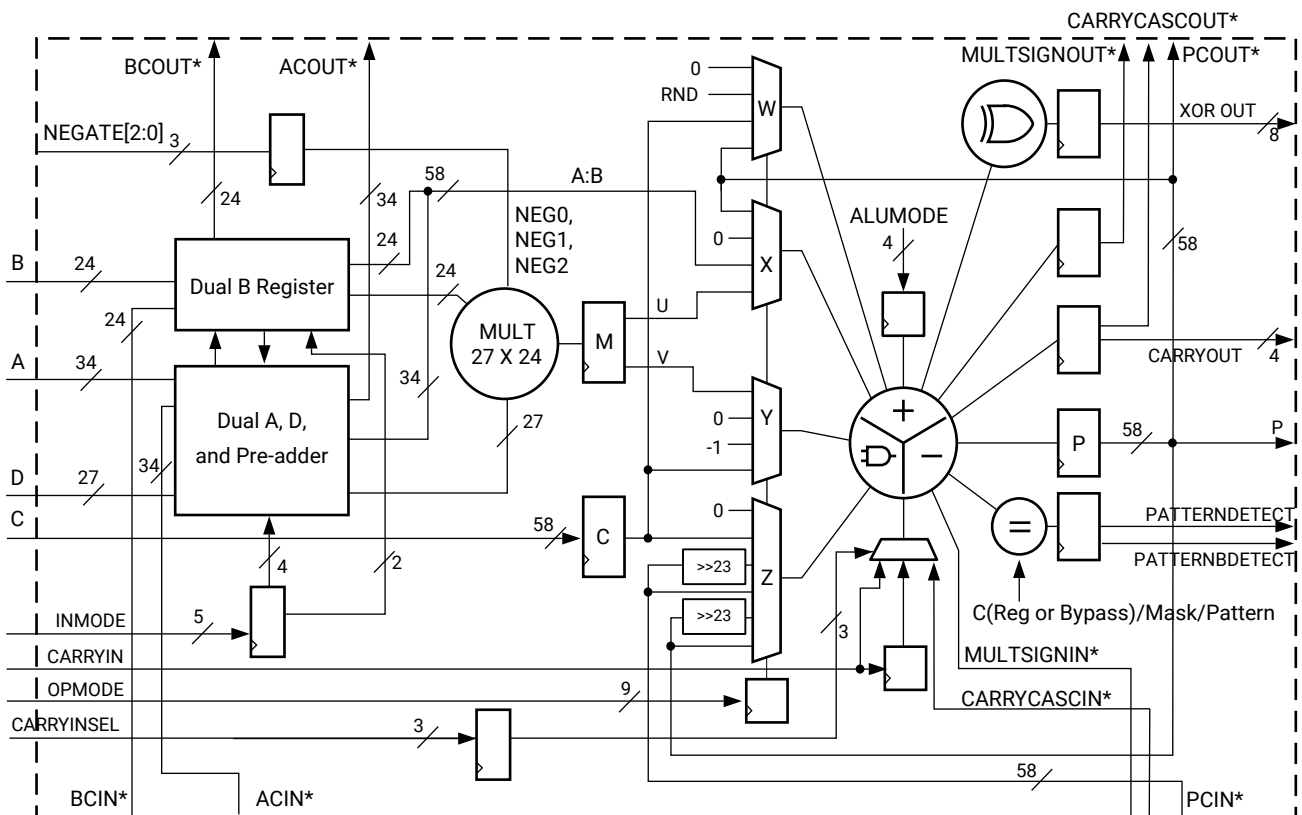
1. The attributes listed below DSPCPLX wherever applicable exist for both DSPs and are represented as unique attributes. The naming convention is <Attribute>_IM (imaginary) and <Attribute>_RE (real) - the exception to this are attributes related to CONJUGATE, where the convention is <Attribute>_A and <Attribute>_B. Additionally, pin inversion attributes follow the convention IS_<PinName>_<IM/RE/A/B>_INVERTED.
2. The value in parenthesis is the default value of the attribute.

Scalar Fixed-Point ALU

Overview

This chapter provides technical details of DSP58 in the scalar fixed-point ALU mode. In this mode, DSP58 consists of a 27-bit pre-adder, 27 × 24 multiplier, and 58-bit ALU that serves as a post-adder/subtractor, accumulator, or logic unit (see the following figure).

Figure 8: Detailed DSP58 Functions as a Scalar Fixed-Point ALU



*These signals are dedicated routing paths internal to the DSP58 column. They are not accessible through general-purpose routing resources.

X21267-051420

The DSP58 scalar fixed-point ALU supports many independent functions. These functions include:

- 27×24 two's complement multiply
- $27 \times 24 + 58$ two's complement multiply accumulate (MACC)
- $27 \times 24 + 58$ two's complement multiply add
- Four-input 58-bit add
- Barrel shifter
- Wide-bus multiplexing
- Magnitude comparator
- Bitwise logic functions
- Wide XOR function
- Pattern detect
- Wide counter

The architecture also supports cascading multiple DSP58s to form width math functions, DSP filters, and complex arithmetic without the use of general logic.

DSP58 Features

Features of DSP58 are as follows.

- Backward compatibility with DSP48E2
- 27-bit pre-adder with D register to enhance the capabilities of the A or B path
- A or B can be selected as pre-adder input to allow for wider multiplication coefficients
- The result of the pre-adder can be sent to both inputs of the multiplier to provide squaring capability
- INMODE control supports balanced pipelining when dynamically switching between multiply ($A \times B$) and add operations ($A+B$) for fixed point, non-complex numbers
- 27×24 two's complement multiplier with optional product negation
- 34-bit A input of which the lower 27 bits feed the A input of the multiplier, and the entire 34-bit input forms the upper 34 bits of the 58-bit A:B concatenated internal bus
- Cascading A and B input:
 - Semi-independently selectable pipelining between direct and cascade paths
 - Separate clock enables for two-deep A and B set of input registers

- Independent 58-bit C input and C register with independent reset and clock enable
- CARRYCASCIN and CARRYCASCOUT internal cascade signals to support 116-bit accumulators/adders/subtractors in two DSP58s, and to support cascading more than two DSP58s
- MULTSIGNIN and MULTSIGNOUT internal cascade signals with special OPMODE setting to support a 116-bit MACC extension
- Single instruction multiple data (SIMD) mode for four-input adder/subtractor, which precludes the use of multiplier in first stage:
 - Dual 24-bit SIMD adder/subtractor/accumulator with two separate CARRYOUT signals
 - Quad 12-bit SIMD adder/subtractor/accumulator with four separate CARRYOUT signals
- 58-bit logic unit:
 - Bitwise logic operations—two-input AND, OR, NOT, NAND, NOR, XOR, and XNOR
 - Logic unit mode dynamically selectable through ALUMODE and OPMODE[3:2]
- 116-bit wide XOR selectable for XOR12, XOR22 (new), XOR24, XOR34 (new), XOR58 (new), and XOR116 (new)

Note: XOR48 and XOR96 are supported when migrating from the UltraScale™ architecture.
- Pattern detector:
 - Overflow/underflow support
 - Convergent rounding support
 - Terminal count detection support and auto resetting: auto resetting can give priority to clock enable
- Cascading 58-bit P bus supports internal low-power adder cascade: 58-bit P bus allows for 12-bit quad or 24-bit dual SIMD adder cascade support
- 23-bit right shift to enable wider multiplier implementation, 17-bit right shift is supported when migrating from the UltraScale architecture
- Dynamic user-controlled operating modes:
 - 9-bit OPMODE control bus provides W, X, Y, and Z multiplexer select signals
 - 5-bit INMODE control bus provides selects for 2-deep A and B registers, pre-adder add-sub control as well as mask gates for pre-adder multiplexer functions.
 - 1-bit NEGATE control bit to conditionally negate the multiplier product
 - 4-bit ALUMODE control bus selects logic unit function and accumulator add-sub control
- Carry in for the second stage adder:
 - Support for rounding

- Support for wider add/subtracts
- 3-bit CARRYINSEL multiplexer
- Carry out for the second stage adder:
 - Support for wider add/subtracts
 - Available for each SIMD adder (up to four)
 - Cascaded CARRYCASCOUT and MULTSIGNOUT allows for MACC extensions up to 116 bits
- Single clock for synchronous operation
- Optional input, pipeline, and output/accumulate registers
- Optional registers for control signals (OPMODE, ALUMODE, and CARRYINSEL)
- Independent clock enable and synchronous resets with programmable polarity for greater flexibility
- Internal multiplier and XOR logic can be gated off when unused to save power

DSP58 consists of a multiplier followed by an accumulator. At least three pipeline registers are required for both multiply and multiply-accumulate operations to run at full speed. The multiply operation in the first stage generates two partial products that need to be added together in the second stage.

When only one or two registers exist in the multiplier design, the M register should always be used to save power and improve performance.

Add/Sub and logic unit operations require at least two pipeline registers (input, output) to run at full speed.

The cascade capabilities of DSP58 are extremely efficient at implementing high-speed pipelined filters built on the adder cascades instead of adder trees.

Multiplexers are controlled with dynamic control signals, such as OPMODE, ALUMODE, and CARRYINSEL, enabling a great deal of flexibility. Designs using registers and dynamic opmodes are better equipped to take advantage of the DSP58's capabilities than combinatorial multiplies.

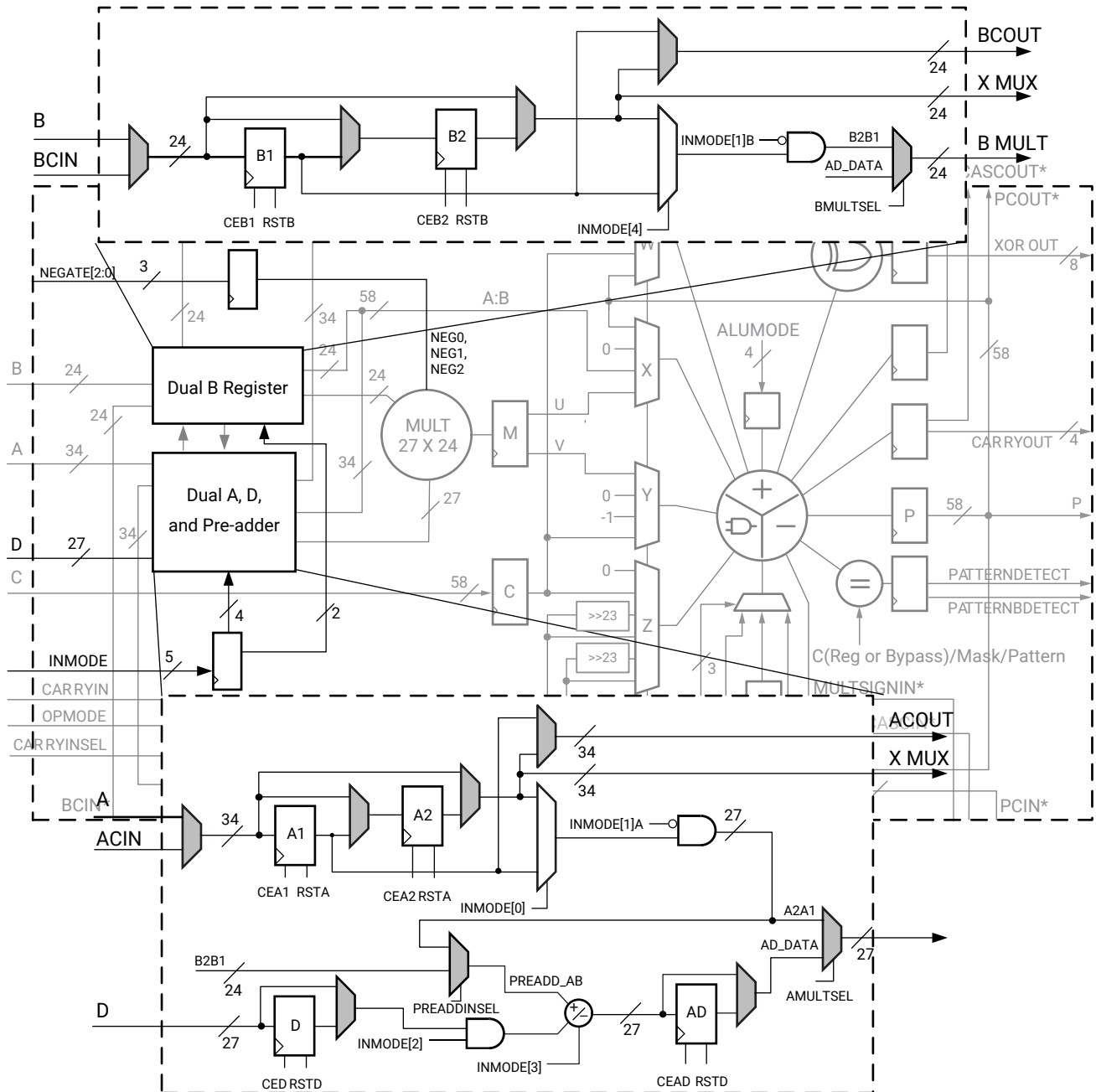
In general, the DSP58 supports both sequential and cascaded operations due to the dynamic OPMODE and cascade capabilities. Fast Fourier transforms (FFTs), floating-point, computation (multiply, add/sub, and divide), counters, and large bus multiplexers are some applications of DSP58.

Additional capabilities of the DSP58 include synchronous resets and clock enables, dual A input pipeline registers, pattern detection, Logic Unit functionality, single instruction/multiple data (SIMD) functionality, and MACC and Add-Acc extension to 116 bits. The DSP58 supports convergent and symmetric rounding, terminal count detection and auto-resetting for counters, and overflow/underflow detection for sequential accumulators. Up to a 116-bit wide XOR function can be implemented as six 12-bit and two 22-bit wide XOR, two 24-bit and two 34-bit wide XOR, or two 48/58-bit wide XOR.

Architectural Highlights of DSP58

The DSP58 contains a pre-adder after the A and B registers with a 27-bit input vector called D. The D register can be used either as the pre-adder register or an alternate input to the multiplier. The DSP58 specific features are highlighted in the following figure.

Figure 9: Hierarchical View of the DSP58 Input Registers and Pre-Adder



X21268-040120

Each DSP58 has a two-input multi-mode multiplier followed by multiplexers and a four-input adder/subtractor/accumulator. The DSP58 multiplier has asymmetric inputs and accepts a 24-bit two's complement operand and a 27-bit two's complement operand. The multiplier stage produces a 51-bit two's complement result in the form of two partial products. These partial products are sign-extended to 58 bits in the X multiplexer and Y multiplexer and fed into four-input adder for final summation. Therefore, when the multiplier is used, the adder effectively becomes a three-input adder.

The second stage adder/subtractor accepts four 58-bit, two's complement operands plus 1-bit CARRYIN and produces a 58-bit, two's complement result when the multiplier is bypassed by setting USE_MULT attribute to NONE and with the appropriate OPMODE setting. In SIMD mode, the adder/subtractor also supports dual 24-bit or quad 12-bit SIMD arithmetic operations with CARRYOUT bits. In the second stage adder/subtractor, bitwise logic operations on two 58-bit binary numbers (and three 58-bit binary numbers in the special XOR3 case) are also supported with dynamic ALUMODE control signals.

Higher level DSP functions are supported by cascading individual DSP58s in a DSP column. Two datapaths (ACOUT and BCOUT) and the DSP58 outputs (PCOUT, MULTSIGNOUT, and CARRYCASCOUT) provide the cascade capability. The ability to cascade datapaths is useful in filter designs. For example, a finite impulse response (FIR) filter design can use the cascading inputs to arrange a series of input data samples and the cascading outputs to arrange a series of partial output results. The ability to cascade provides a high-performance and low-power implementation of DSP filter functions because the general routing in the internal logic is not used.

The C input allows the formation of many 3-input mathematical functions, such as 3-input addition or 2-input multiplication with an addition. One subset of this function is the valuable support of symmetrically rounding a multiplication toward zero or toward infinity. The C input together with the pattern detector also supports convergent rounding. Refer to [Rounding](#) for a discussion on using the C input to implement the different rounding modes.

For multi-precision arithmetic, DSP58 provides a right wire shift by 23 bits (or 17 bits when migrating from the UltraScale™ architecture). Thus, a partial product from one DSP58 can be right justified and added to the next partial product computed in an adjacent DSP58 above it in the same column. Using this technique, the DSP58s in a column can be used to build higher-precision multipliers.

Programmable pipelining of input operands, intermediate products, and accumulator outputs enhances throughput. The 58-bit internal bus (PCOUT/PCIN) allows for aggregation of DSPs in a single column. CLB logic is needed when spanning multiple DSP columns.

The pattern detector at the output of the DSP58 provides support for convergent rounding, overflow/underflow, block floating-point, and support for accumulator terminal count (counter auto reset). The pattern detector can detect if the output of the DSP58 matches a pattern, as qualified by a mask.

DSP58 Operation Modes

Table 5 provides a summary of the key operation modes available in a single DSP58, showing the largest functions available, and the key resources used. Table 6 through Table 10 show similar operation modes extended to two, three, four, six, and eight DSP58s, cascaded.

For DSP48E2 supported operations with bit-widths derived from $27 \times 18 + 48$ operations, refer to *UltraScale Architecture DSP Slice User Guide (UG579)*.

Table 5: Operation Modes: One DSP58

Operation Mode	Pre-Adder	A/B/P Cascade	48-bit C Port	RND Support
$27 \times 24 + C$ MULT/MACC	23/26-bit	N/A	Used	Yes
27×24 Sequential Complex MACC	Optional	N/A	Optional	Yes
27×25 or 28×24	N/A	N/A	Used	Limited
Pre-adder Squared	23-bit	N/A	Optional	Optional
SIMD Add/Sub/Counter/ACC	N/A	N/A	Used	No
58-bit Add/Sub/Counter/ACC	N/A	N/A	Used	Yes
58-Bit 2:1 Bus MUX	N/A	N/A	Used	N/A
XOR116/58/34/24/22/12	N/A	N/A	Used	N/A
AND116/NOR116	N/A	N/A	Used	N/A
58 2-input Logic Operations	N/A	N/A	Used	N/A

Table 6: Operation Modes: Two DSP58s

Operation Mode	Pre-Adder	A/B/P Cascade	48-bit C Port	RND Support
18×18 Complex MULT/MACC	N/A	N/A	Used	yes
$27 \times 24 + C$ MACC116	26-bit	P Used	Used	Yes
$47 \times 27 + C$	26-bit	Yes	Used	Yes
47×28 or 48×27	N/A	Yes	Used	Limited
$50 \times 24 + C$	23-bit	Yes	Used	Yes
50×25 or 51×24	N/A	Yes	Used	Limited
$27 \times 24 + C$ Systolic MultAdd 2-tap Filter	23/26-bit	Yes	Used	Yes
Sum of 2 Pre-adder Squared	23-bit	P Used	Optional	Optional
116-bit Add/Sub/Counter/ACC	N/A	N/A	Used	Yes
24-bit Barrel Shifter	N/A	Yes	N/A	N/A
46-bit Bus Shifter	N/A	Yes	N/A	N/A
58-Bit 4:1 Bus MUX	N/A	P Used	Used	N/A
XOR232/116/68/48/44/24	N/A	P Used	Used	N/A
AND174/NOR174	N/A	P Used	Used	N/A
58 3-input Logic Operations (58 XOR4)	N/A	P Used	Used	N/A

Table 7: Operation Modes: Three DSP58s

Operation Mode	Pre-Adder	A/B/P Cascade	48-bit C Port	RND Support
26 × 23 Complex MULT/MACC	26-bit	A/B Used	Used	Yes
70 × 27 + C	26-bit	Yes	Used	Yes
70 × 28 or 71 × 27	N/A	Yes	Used	Limited
73 × 24 + C	24-bit	Yes	Used	Yes
73 × 25 or 74 × 24	N/A	Yes	Used	Limited
27 × 24 + C Systolic MultAdd 3-tap Filter	23/26-bit	Yes	Used	Yes
Sum of 3 Pre-adder Squared	23-bit	P Used	Optional	Optional
174-bit Add/Sub/Counter/ACC	N/A	N/A	Used	Yes
58-Bit 6:1 Bus MUX	N/A	P Used	Used	N/A
XOR348/174/102/72/66/36	N/A	P Used	Used	N/A
AND232/NOR232	N/A	P Used	Used	N/A
58 4-input Logic Operations (58 XOR6)	N/A	P Used	Used	N/A

Table 8: Operation Modes: Four DSP58s

Operation Mode	Pre-Adder	A/B/P Cascade	48-bit C Port	RND Support
27 × 25 Complex MULT	N/A	P Used	Used	Yes
27 × 24 + C Complex MULT/MACC	26-bit	P Used	Used	Yes
50 × 47 + C	N/A	B/P Used	Used	Yes
50 × 48 or 51 × 47	N/A	B/P Used	Used	Limited
93 × 27 + C	26-bit	Yes	Used	Yes
93 × 28 or 94 × 27	N/A	Yes	Used	Limited
96 × 24 + C	23-bit	Yes	Used	Yes
96 × 25 or 97 × 24	N/A	Yes	Used	Limited
27 × 24 + C Systolic MultAdd 4-tap Filter	23/26 bit	Yes	Used	Yes
Sum of 4 Pre-adder Squared	23-bit	P Used	Optional	Optional
232-bit Add/Sub/Counter/ACC	N/A	N/A	Used	Yes
58-Bit 8:1 Bus MUX	N/A	P Used	Used	N/A
XOR464/232/136/96/88/48	N/A	P Used	Used	N/A
AND290/NOR290	N/A	P Used	Used	N/A
58 5-input Logic Operations (58 XOR8)	N/A	P Used	Used	N/A

Table 9: Operation Modes: Six DSP58s

Operation Mode	Pre-Adder	A/B/P Cascade	48-bit C Port	RND Support
27 × 24 + C Complex MACC116	23-bit	P Used	Used	Yes
73 × 47 + C	N/A	B/P Used	Used	Yes
73 × 48 or 74 × 47	N/A	B/P Used	Used	Yes
139 × 27 + C	26-bit	Yes	Used	Yes

Table 9: Operation Modes: Six DSP58s (cont'd)

Operation Mode	Pre-Adder	A/B/P Cascade	48-bit C Port	RND Support
139×28 or 140×27	N/A	Yes	Used	Limited
$142 \times 24 + C$	23-bit	Yes	Used	Yes
142×25 or 143×24	N/A	Yes	Used	Limited
53×53 Unsigned	N/A	Yes	Used	No
$27 \times 24 + C$ Systolic MultAdd 6-tap Filter	23/26 bit	Yes	Used	Yes
Sum of 6 Pre-adder Squared	23-bit	P Used	Optional	Optional
348-bit Add/Sub/Counter/ACC	N/A	N/A	Used	Yes
58-Bit 12:1 Bus MUX	N/A	P Used	Used	N/A
XOR696/348/204/144/132/72	N/A	P Used	Used	N/A
AND406/NOR406	N/A	P Used	Used	N/A
58 7-input Logic Operations (58 XOR12)	N/A	P Used	Used	N/A

Table 10: Operation Modes: Eight DSP58s

Operation Mode	Pre-Adder	A/B/P Cascade	48-bit C Port	RND Support
$47 \times 27 + C$ Complex MULT	26-bit	B/P Used	Used	Yes
$96 \times 47 + C$	N/A	B/P Used	Used	Yes
96×48 or 97×47	N/A	B/P Used	Used	Limited
$93 \times 50 + C$	N/A	B/P Used	Used	Yes
93×51 or 94×50	N/A	B/P Used	Used	Limited
$27 \times 24 + C$ Systolic MultAdd 8-tap Filter	23/26-bit	Yes	Used	Yes
Sum of 8 Pre-adder Squared	23-bit	P Used	Optional	Optional
464-bit Add/Sub/Counter/ACC	N/A	N/A	Used	Yes
58-Bit 16:1 Bus MUX	N/A	P Used	Used	N/A
XOR928/464/272/192/176/96	N/A	P Used	Used	N/A
AND522/NOR522	N/A	P Used	Used	N/A
58 9-input Logic Operations (48 XOR16)	N/A	P Used	Used	N/A

Simplified DSP58 Operations

The fixed-point math portion of DSP58 consists of a 27-bit pre-adder, a 27-bit by 24-bit two's complement multiplier with optional product negation followed by four 58-bit datapath multiplexers (with outputs W, X, Y, and Z). This is followed by a four-input adder/subtractor or two-input logic unit. When using two-input logic unit, the multiplier cannot be used.

The data and control inputs to DSP58 feed the arithmetic and logic stages. The A and B data inputs can optionally be registered one or two times to assist the construction of different, highly pipelined, DSP application solutions. The D path and the AD path can each be registered once. The other data inputs and the control inputs can be optionally registered once.

The following equation summarizes the combination of W, X, Y, Z, and CIN by the adder/subtractor. The CIN, W multiplexer output, X multiplexer output, and Y multiplexer output are always added together. This combined result can be selectively added to or subtracted from the Z multiplexer output. The second option is obtained by setting the ALUMODE to 0001.

$$\text{Adder / Subtractor Out} = (Z \pm (W + X + Y + \text{CIN})) \text{ or } (-Z + (W + X + Y + \text{CIN}) - 1)$$

A typical use of DSP58 is where A and B inputs are multiplied and the result is added to or subtracted from the C register. Selecting the multiplier function consumes both X and Y multiplexer outputs to feed the adder. The two 51-bit partial products from the multiplier are sign-extended to 58 bits before being sent to the adder/subtractor.

When not using the first stage multiplier, the 58-bit, dual input, bit-wise logic function implements AND, OR, NOT, NAND, NOR, XOR, and XNOR. The inputs to these functions are:

- All 0s on the W multiplexer
- Either A:B or P on the X multiplexer
- Either all 1s or all 0s on the Y multiplexer depending on logic operation
- Either C, P, or PCIN on the Z multiplexer

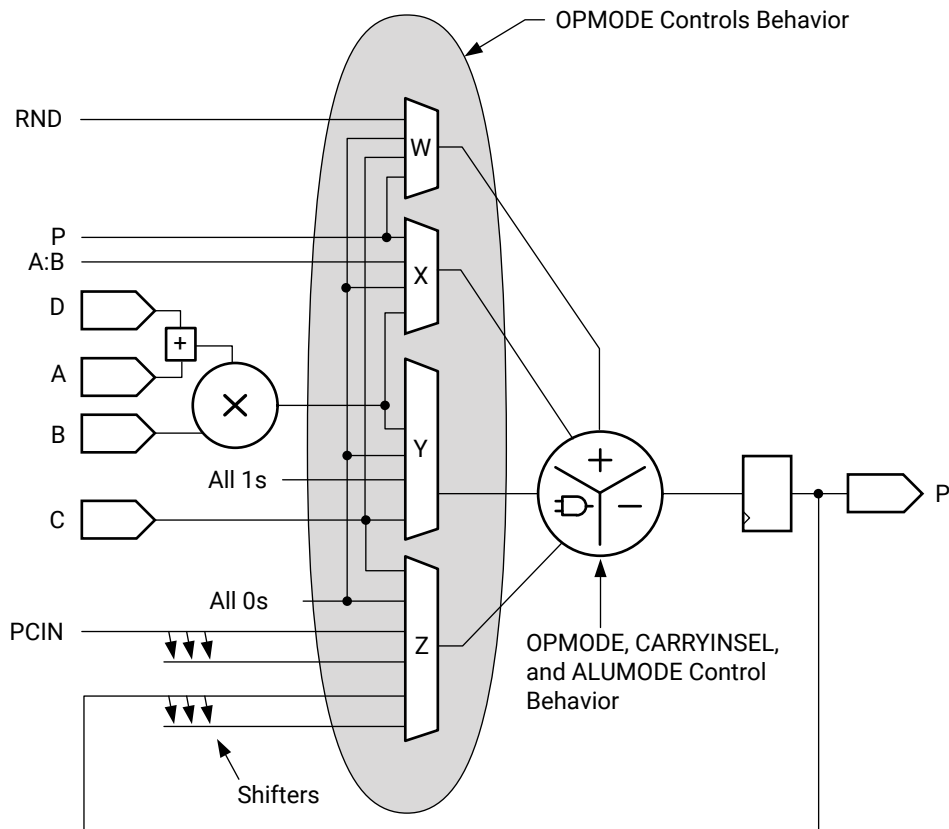
Creating wider logic operations is feasible using this cascade path because PCIN is a cascade input from a lower DSP58. A 58-bit, triple input, bit-wise XOR3 logic operation is supported when the Y multiplexer selects the C input and ALUMODE[3:0] = 0100.

The output of the adder/subtractor or logic unit feeds the pattern detector logic. The pattern detector allows DSP58 to support convergent rounding, counter autoreset when a count value has been reached, and overflow, underflow, and saturation in accumulators. In conjunction with the logic unit, the pattern detector can be extended to perform a 58-bit dynamic comparison of two 58-bit fields.

The following figure illustrates DSP58 in a simplified form. The nine OPMODE bits control the selection of the W, X, Y, and Z multiplexers, feeding the inputs to the adder, subtractor, or logic unit. In all cases, the 51-bit partial product data from the multiplier to the X and Y multiplexers is sign-extended, forming 58-bit input datapaths to the adder/subtractor. Based on 51-bit operands and a 58-bit accumulator output, the number of guard bits (that is, bits available to guard against overflow) is 7. To extend the number of MACC operations, the MACC_EXTEND feature must be

used. This feature allows the MACC to extend to 116 bits with two DSP58s. If both A and B are limited to 18 bits (sign-extended to 27 and 24), then there are 22 (58–36) guard bits for the MACC. The CARRYOUT bits are invalid during multiply operations. Combinations of OPMODE, ALUMODE, CARRYINSEL, and CARRYIN control the function of the adder/subtractor or logic unit.

Figure 10: Simplified DSP58 Operation

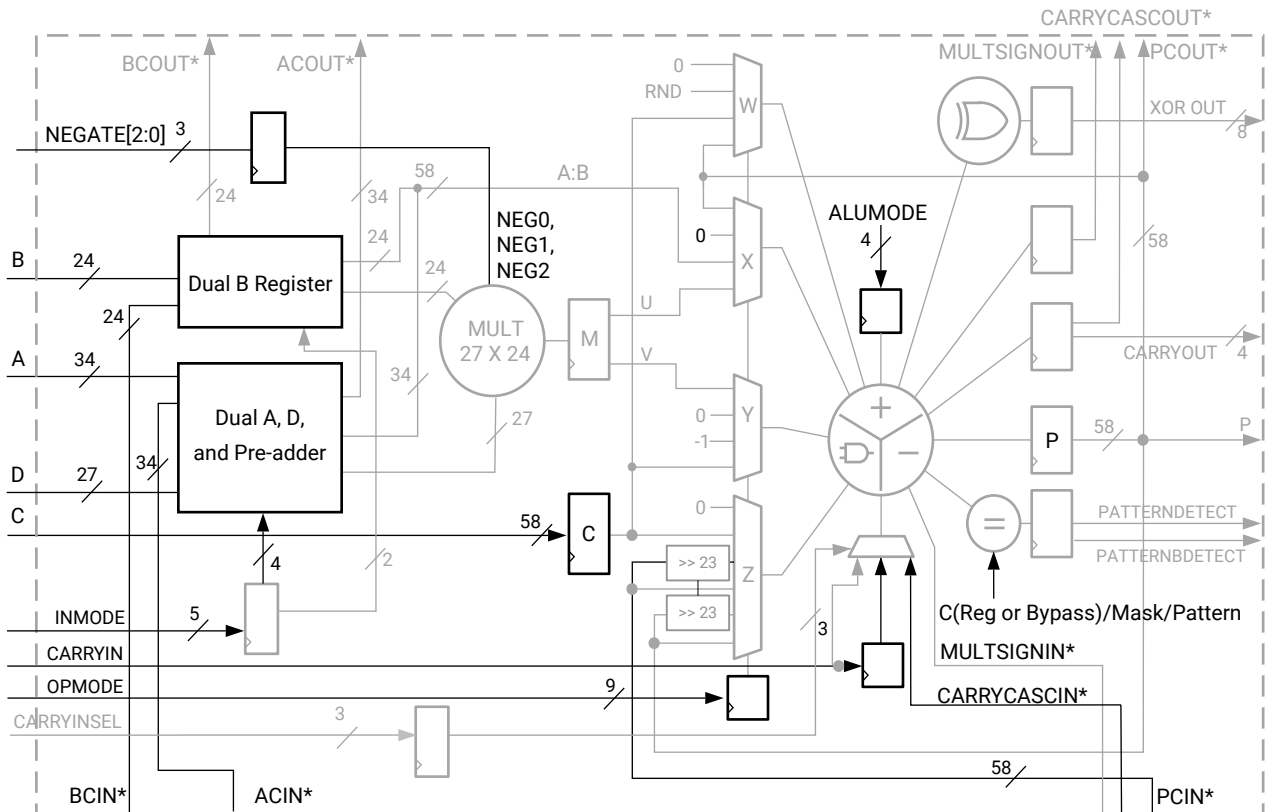


X21269-080618

Input Ports

This section describes the input ports of DSP58 in detail. The input ports of DSP58 are highlighted in the following figure.

Figure 11: Input Ports in DSP58



*These signals are dedicated routing paths internal to the DSP58 column. They are not accessible through general-purpose routing resources.

X21335-051420

A, B, C, and D Ports

The DSP58 input data ports support many common algorithms. DSP58 has four direct input data ports labeled A, B, C, and D. The A data port is 34 bits wide, the B data port is 24 bits wide, the C data port is 58 bits wide, and the pre-adder D data port is 27 bits wide.

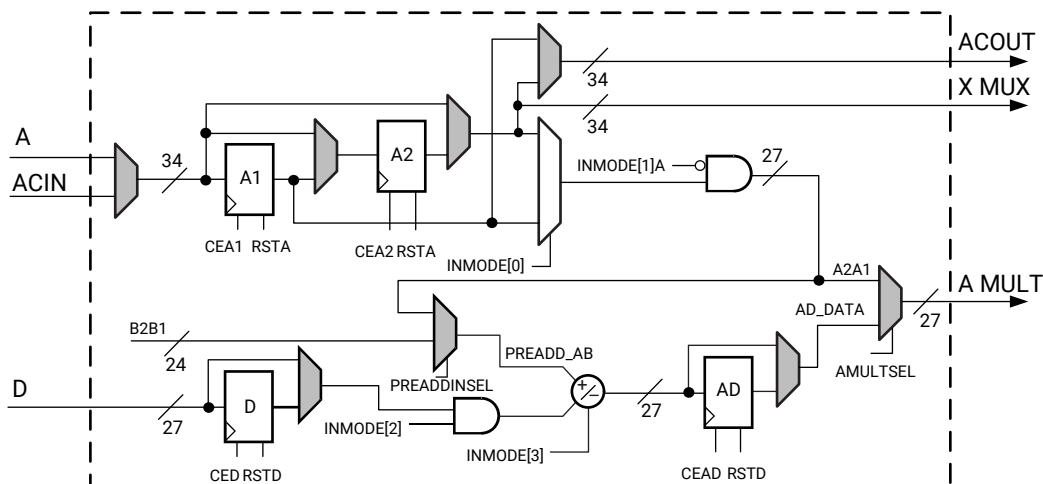
The 27-bit A (A[26:0]) and 24-bit B ports supply input data to the 27-bit by 24-bit, two's complement multiplier. With the independent C port, each DSP58 is capable of multiply-add, multiply-subtract, and multiply-round operations.

Concatenated A and B ports (A:B) bypass the multiplier and feed the X multiplexer input. The 34-bit A input port forms the upper 34 bits of A:B concatenated datapath, and the 24-bit B input port forms the lower 24 bits of the A:B datapath. The A:B datapath, together with the C input port, enables each DSP58 to implement a full 58-bit adder/subtractor provided the multiplier is not used, which is achieved by setting USE_MULT to NONE.

Each DSP58 also has two cascaded input datapaths (ACIN and BCIN) that provide a cascaded input stream between adjacent DSP58s in the same column. The cascaded path is 34 bits wide for the A input and 24 bits wide for the B input. Applications benefiting from this feature include FIR filters, complex multiplication (larger than 18×18), multi-precision multiplication, and complex MACCs (larger than 18×18).

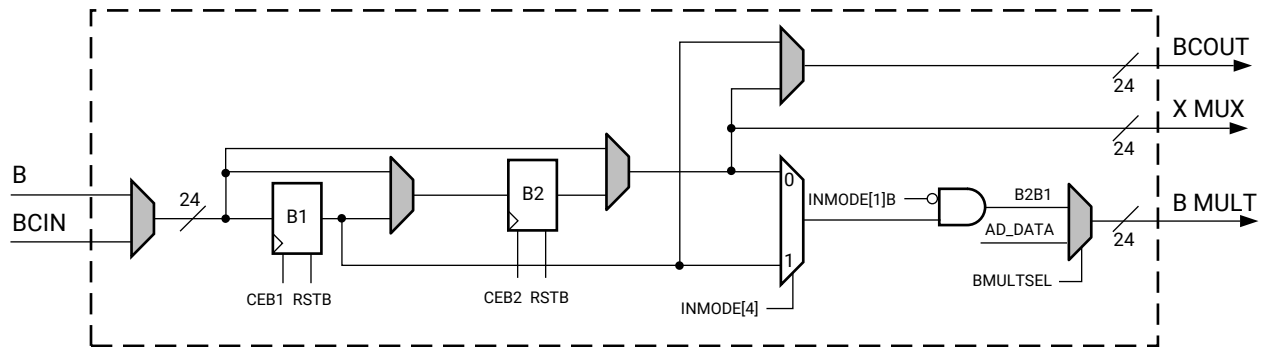
The A and B input port and the ACIN and BCIN cascade port can have 0, 1, or 2 pipeline stages in its datapath. The dual A, D, and pre-adder port logic is shown in Figure 12. The dual B register port logic is shown in Figure 13. The different pipestages are set using attributes. Attributes AREG and BREG are used to select the number of pipeline stages for A and B direct inputs to the X multiplexer to the ALU, and INMODE[0] can dynamically change the number of pipeline stages to the multiplier. Attributes ACASCREG and BCASCREG select the number of pipeline stages in the ACOU and BCOU cascade datapaths. The allowed attribute settings are shown in Table 4. Multiplexers controlled by configuration bits select flow through paths, optional registers, or cascaded inputs. The data port registers allow users to typically trade off increased clock frequency (that is, higher performance) versus data latency.

Figure 12: Dual A, D, and Pre-Adder Logic



X21337-082018

Figure 13: Dual B Register Logic



X21338-100618

The following table shows the encoding for the INMODE[4:0] dynamic control bits and AMULTSEL, BMULTSEL, and PREADDINSEL static control bits.

These bits select the functionality of the pre-adder, the A, B, and D input registers. AMULTSEL and/or BMULTSEL must be set to AD to enable the pre-adder functions described in the table. Additionally, a new dynamic control port called NEGATE, is used to conditionally negate the multiplier product.

In summary, the INMODE dynamic control signals along with AMULTSEL, BMULTSEL, and PREADDINSEL static attributes control the pre-adder functionality and A, B, and D register bus multiplexers that precede the multiplier, as well as the multiplier product negation controlled by NEGATE control signals that have an effect on the multiplier. The DSP58 supports two-deep A or B sourcing the pre-adder as well as a pre-adder squaring function.

Table 11: INMODE[4:0] Functions with Pre-Adder Options

NEGATE ⁴	INMODE[4]	INMODE[3]	INMODE[2]	INMODE[1]	INMODE[1]A ¹	INMODE[1]B ¹	INMODE[0]	PREADDINSEL	BMULTSEL	AMULTSEL	Multiplier A Port	Multiplier B Port ³	Pre-Adder/Multiplier Function
0/1	0/1	0	0	0	0	0	0/1	A/B	B	A	A2/A1	B2/B1	$\pm A[26:0] * B[23:0]$
0/1	0/1	0/1	1	0	0	0	0/1	A	B	AD	$D \pm A2/A1^2$	B2/B1	$\pm (D[25:0] \pm A[25:0]) * B[23:0]$
0/1	0/1	0	0	1	1	0	X	A	B	A	Zero	B2/B1	Zero
0/1	0/1	0	0	1	0	1	X	B	B	A	A2/A1	Zero	Zero

Table 11: INMODE[4:0] Functions with Pre-Adder Options (cont'd)

NEGATE ⁴	INMODE[4]	INMODE[3]	INMODE[2]	INMODE[1]	INMODE[1]A ¹	INMODE[1]B ¹	INMODE[0]	PREADDINSEL	BMULTSEL	AMULTSEL	Multiplier A Port	Multiplier B Port ³	Pre-Adder/Multiplier Function
0/1	X	0/1	1	0	0	0	0/1	A	AD	AD	$D \pm A2/A1^2$	$D \pm A2/A1^2$	$\pm(D[22:0] \pm A[22:0])^2$
0/1	X	0	1	1	1	0	X	A	AD	AD	D	D	$\pm D[23:0]^2$
0/1	X	0/1	0	0	0	0	0/1	A	AD	AD	$\pm A2/A1$	$\pm A2/A1$	$\pm A[23:0]^2$
0/1	X	0/1	1	0	0	0	0/1	A	AD	A	$A2/A1$	$D \pm A2/A1^2$	$\pm(D[22:0] \pm A[22:0])^* A[22:0]$
0/1	0/1	0/1	1	0	0	0	0/1	B	AD	A	$A2/A1$	$D \pm B2/B1^2$	$\pm(D[22:0] \pm B[22:0])^* A[26:0]$
0/1	X	0	1	1	0	1	0/1	B	AD	A	$A2/A1$	D	$\pm D[23:0]^* A[26:0]$
0/1	0/1	0/1	1	0	0	0	0/1	B	AD	AD	$D \pm B2/B1^2$	$D \pm B2/B1^2$	$\pm(D[22:0] \pm B[22:0])^2$
0/1	0/1	0/1	0	0	0	0	0/1	B	AD	AD	$\pm B2/B1$	$\pm B2/B1$	$\pm B[23:0]^2$
0/1	0/1	0/1	1	0	0	0	0/1	B	B	AD	$D \pm B2/B1^2$	$B2/B1$	$\pm(D[25:0] \pm B[22:0])^* B[22:0]$
A[26]	0	X	0	0	0	0	0	X	B	A	A	B = 1	A[26:0]
B[23]	0	X	0	0	0	0	0	X	B	A	A = 1	B	B[23:0]
D[26]	0/1	0	1	1	1	0	0/1	A	B	AD	D + Zerp	B = 1	D[26:0]

Notes:

1. INMODE[1]A and INMODE[1]B are internal signals defined by the user settings of PREADDINSEL and INMODE[1]. If PREADDINSEL=A, INMODE[1]A (see Figure 12) is INMODE[1] and INMODE[1]B (see Figure 13) is 0. If PREADDINSEL=B, INMODE[1]B is INMODE[1] and INMODE[1]A is 0.
2. Set the data on the D and the A or B ports so the pre-adder, which does not support saturation, does not overflow or underflow. See Pre-Adder.
3. A or D are limited to 24-bit (if only one bus is 0) when provided through the B port (if the pre-adder is configured as a multiplexer). A or D are limited to 23-bit two's complement signed extended numbers when both are non-zero, PREADDINSEL = A, BMULTSEL = AD, and the pre-adder is configured as an adder/subtractor.
4. With DSP58, set NEGATE = 1 to negate the product instead of using the pre-adder to negate one of the inputs. Doing so skips the pre-adder stage, which at high clock frequencies, reduces one cycle of latency. Furthermore, the pre-adder can only support 26-bit two's complement negation whereas the NEGATE pin directly negates the product of the 27 × 24 multiplier.

INMODE[0] selects between A1 (INMODE[0] = 1) and the A2 MUX controlled by AREG (INMODE[0] = 0).

INMODE[1] can be used to gate the A or B datapath to use the pre-adder to create a 2:1 bus multiplexer along with the INMODE[2] control signal.

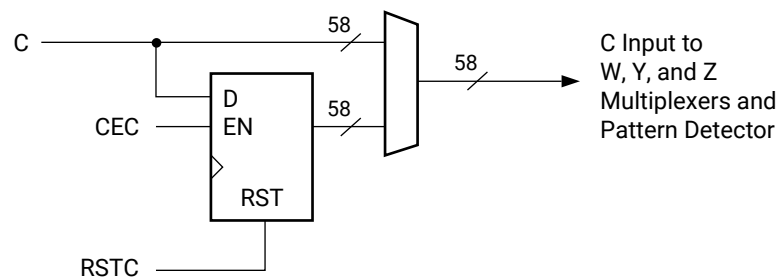
When INMODE[2] = 0, the D input to the pre-adder is 0. INMODE[1] and INMODE[2] enable multiplexing between the D register and the A or B registers, without having to use resets to force them to zero. For information on how to configure the preadder as a 2:1 multiplexer, see [Pre-Adder Block Applications](#).

INMODE[3] provides pre-adder subtract control, where INMODE[3] = 1 indicates subtract and INMODE[3] = 0 indicates add of A or B to D. When D is gated off, this dynamic inversion can provide the absolute value of A or B.

INMODE[4] selects between B1 (INMODE[4] = 1) and the B2 MUX controlled by BREG (INMODE[4] = 0).

The 58-bit C port is used as a general input to the W, Y, and Z multiplexers to perform add, subtract, four-input add/subtract, and logic functions. The C input is also connected to the pattern detector and can be used to support rounding function implementations. The C port logic is shown in the following figure. The CREG attribute selects the number of pipestages for the C input datapath.

Figure 14: C Port Logic

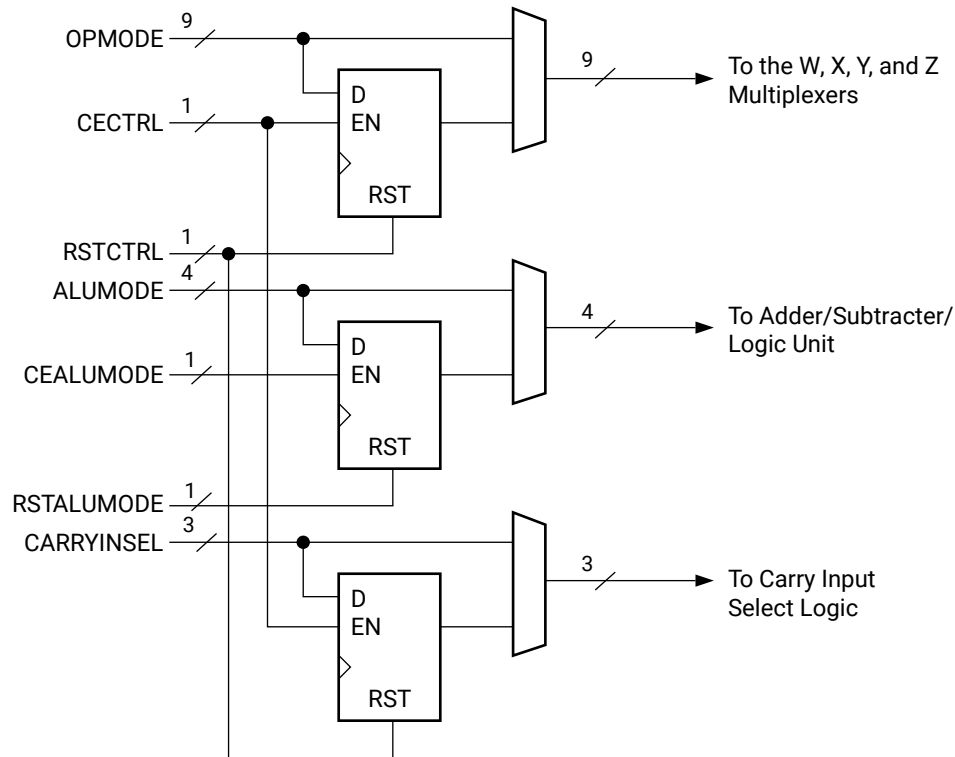


X21339-082018

OPMODE, ALUMODE, and CARRYINSEL Port Logic

The OPMODE, ALUMODE, and CARRYINSEL port logic support flow through or registered input control signals. Multiplexers controlled by configuration bits select flow through or optional registers. The control port registers allow you to trade off increased clock frequency (i.e., higher performance) versus data latency. The registers have independent clock enables and resets. The OPMODE and CARRYINSEL registers are reset by RSTCTRL. The ALUMODE is reset by RSTALUMODE. The clock enables and the OPMODE, ALUMODE, and CARRYINSEL port logic are shown in the following figure.

Figure 15: OPMODE, ALUMODE, and CARRYINSEL Port Logic



X21340-101118

W, X, Y, and Z Multiplexers

The OPMODE control input contains fields for W, X, Y, and Z multiplexer selects.

The OPMODE input provides a way for you to dynamically change DSP58 functionality from clock cycle to clock cycle (for example, when altering the internal datapath configuration of DSP58 relative to a given calculation sequence). The OPMODE bits can be optionally registered using the OPMODEREG attribute.

The following tables list the possible values of OPMODE and the resulting function at the outputs of the four multiplexers (W, X, Y, and Z multiplexers). The multiplexer outputs supply four operands to the following adder/subtractor. Not all possible combinations for the multiplexer select bits are allowed. Some are marked in the tables as *illegal selection* and give undefined results. If the multiplier output is selected, then both the X and Y multiplexers are used to supply the multiplier partial products to the adder/subtractor.

Table 12: OPMODE Control Bits Select W Multiplexer Outputs

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	W Multiplexer Output	Notes
00	xxx	xx	xx	0	Default. Must be selected for logic operations.
01	xxx	xx	xx	P	Requires PREG = 1
10	xxx	xx	xx	RND	-
11	xxx	xx	xx	C	-

Table 13: OPMODE Control Bits Select X Multiplexer Outputs

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	X Multiplexer Output	Notes
xx	xxx	xx	00	0	Default
xx	xxx	01	01	$\pm M$	Must select with OPMODE[3:2] = 01 M is selected when NEGATE = 0. -M is selected when NEGATE = 1.
xx	xxx	xx	10	P	Requires PREG = 1
xx	xxx	xx	11	A:B	58-bit wide

Table 14: OPMODE Control Bits Select Y Multiplexer Outputs

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	Y Multiplexer Output	Notes
xx	xxx	00	xx	0	Default
xx	xxx	01	01	$\pm M$	Must select with OPMODE[1:0] = 01 M is selected when NEGATE = 0. -M is selected when NEGATE = 1.
xx	xxx	10	xx	58' FFFFFFFFFF F	Used mainly for logic unit bitwise operations on the X and Z multiplexers.
xx	xxx	11	xx	C	

Table 15: OPMODE Control Bits Select Z Multiplexer Outputs

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	Z Multiplexer Output	Notes
xx	000	xx	xx	0	Default
xx	001	xx	xx	PCIN	-

Table 15: OPMODE Control Bits Select Z Multiplexer Outputs (cont'd)

W OPMODE[8:7]	Z OPMODE[6:4]	Y OPMODE[3:2]	X OPMODE[1:0]	Z Multiplexer Output	Notes
xx	010	xx	xx	P	Requires PREG = 1
xx	011	xx	xx	C	-
00	100	10	00	P	Use for MACC extend only. Requires PREG = 1
xx	101	xx	xx	17- or 23-bit Shift (PCIN)	Arithmetic right shift by 17 bits only if DSP_MODE is DSP48E2.
xx	110	xx	xx	17- or 23-bit Shift (P)	PREG = 1. Arithmetic right shift by 17 bits only if DSP_MODE is DSP48E2.
xx	111	xx	xx	xx	Illegal selection.

ALUMODE Inputs

The 4-bit ALUMODE controls the behavior of the second stage adder/subtractor/logic unit. ALUMODE = 0000 selects add operations of the form $Z + (W + X + Y + \text{CIN})$. CIN is the output of the CARRYIN MUX (see Figure 16). ALUMODE = 0011 selects subtract operations of the form $Z - (W + X + Y + \text{CIN})$. ALUMODE = 0001 can implement $-Z + (W + X + Y + \text{CIN}) - 1$. ALUMODE = 0010 can implement $-(Z + W + X + Y + \text{CIN}) - 1$, which is equivalent to $\text{not}(Z + W + X + Y + \text{CIN})$. The negative of a two's complement number is obtained by performing a bitwise inversion and adding one, for example, $-k = \text{not}(k) + 1$.

Table 16: Four-Input ALUMODE Operations

DSP Operation	OPMODE[8:0]	ALUMODE[3:0]			
		3	2	1	0
$Z + W + X + Y + \text{CIN}$	Any legal OPMODE	0	0	0	0
$Z - (W + X + Y + \text{CIN})$	Any legal OPMODE	0	0	1	1
$-Z + (W + X + Y + \text{CIN}) - 1 = \text{not}(Z) + W + X + Y + \text{CIN}$	Any legal OPMODE	0	0	0	1
$\text{not}(Z + W + X + Y + \text{CIN}) = -Z - W - X - Y - \text{CIN} - 1$	Any legal OPMODE	0	0	1	0

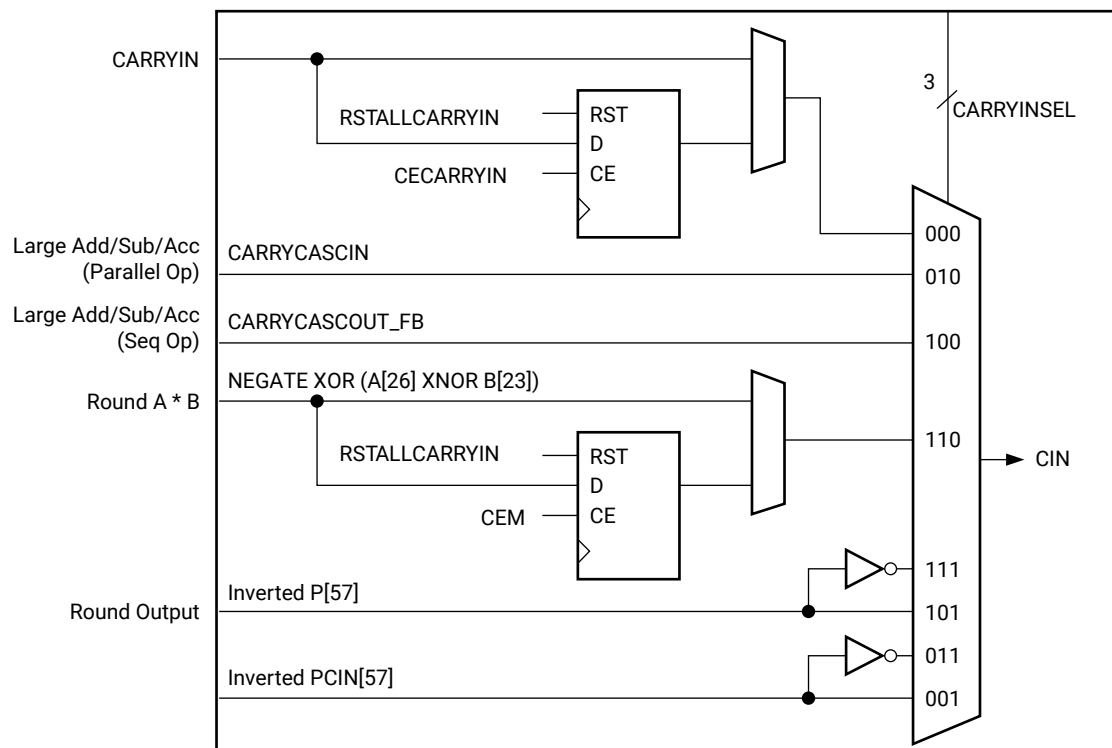
Notes:

- In two's complement, $-Z = \text{not}(Z) + 1$.
- $X + Y = -M$, the two's complement negation of the product, when NEGATE = 1 and OPMODE[3:0] = 4'b0101.

Carry Input Logic

The carry input logic result is a function of a 3-bit CARRYINSEL signal. The inputs to the carry input logic appear in the following figure. Carry inputs used to form results for adders and subtractors are always in the critical path. High performance is achieved by implementing this logic in silicon. The possible carry inputs to the carry logic are gathered prior to the outputs of the W, X, Y, and Z multiplexers. CARRYIN has no dependency on the OPMODE selection.

Figure 16: CARRYINSEL Port Logic



X21360-100618

The figure above shows eight inputs selected by the 3-bit CARRYINSEL control. The first input, CARRYIN (CARRYINSEL set to binary 000), is driven from general logic. This option allows implementation of a carry function based on user logic. CARRYIN can be optionally registered. The next input, (CARRYINSEL is equal to binary 010) is the CARRYCASCIN input from an adjacent DSP58. The third input (CARRYINSEL is equal to binary 100) is the CARRYCASCOUT from the same DSP58, fed back to itself.

The fourth input (CARRYINSEL is equal to binary 110) is the complement of the sign bit of the product (NEGATE XOR (A[26] XNOR B[23])) for symmetrically rounding multiplier outputs towards infinity. This signal can be optionally registered to match the MREG pipeline delay. The fifth and sixth inputs (CARRYINSEL is equal to binary 111 and 101) selects the true or inverted P output MSB P[57] for symmetrical rounding.

The seventh and eight inputs (CARRYINSEL is equal to binary 011 and 001) selects the true or inverted cascaded P input MSB PCIN[57] for symmetrically rounding the cascaded P input.

The following table lists the possible values of the three carry input select bits (CARRYINSEL) and the resulting carry inputs or sources.

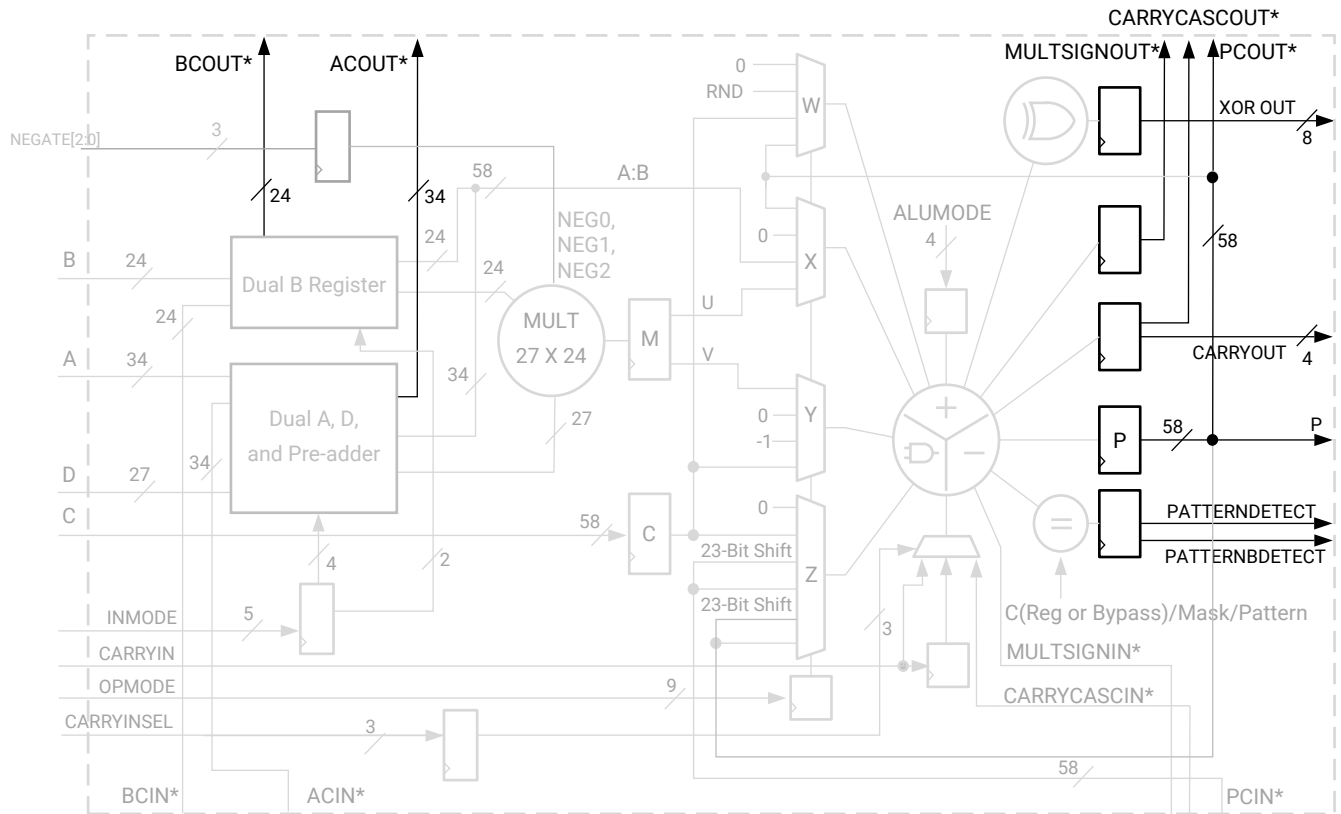
Table 17: CARRYINSEL Control Carry Source

CARRYINSEL			Select	Notes
2	1	0		
0	0	0	CARRYIN	General interconnect
0	0	1	~PCIN[57]	Rounding PCIN (round towards infinity)
0	1	0	CARRYCASCIN	Larger add/sub/acc (parallel operation)
0	1	1	PCIN[57]	Rounding PCIN (round towards zero)
1	0	0	CARRYCASCOU_FB (CARRYCASCOU in the same DSP58)	For larger add/sub/acc (sequential operation through internal feedback). Requires PREG = 1
1	0	1	~P[57]	Rounding P (round towards infinity). Requires PREG = 1
1	1	0	NEGATE XOR (A[26] XNOR B[23])	Symmetric rounding of A x B towards infinity
1	1	1	P[57]	For rounding P (round towards zero). Requires PREG = 1

Output Ports

This section describes the output ports of DSP58 in detail. The output ports in DSP58 are illustrated in the following figure.

Figure 17: Output Ports in DSP58

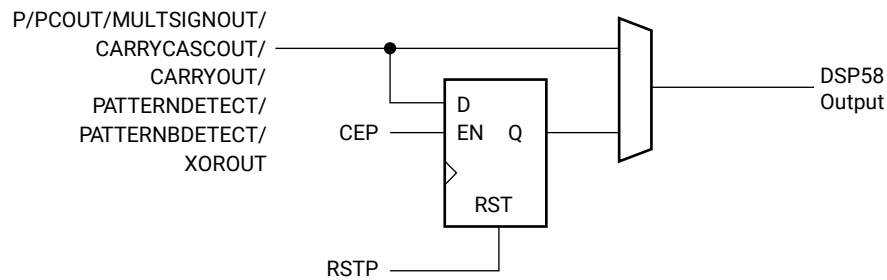


*These signals are dedicated routing paths internal to the DSP58 column. They are not accessible through general-purpose routing resources.

X21361-051420

All the output ports except ACOUT and BCOUT are reset by RSTP and enabled by CEP (see the following figure). ACOUT and BCOUT are reset by RSTA and RSTB, respectively.

Figure 18: Output Port Logic



X21362-101118

P Port

Each DSP58 has a 58-bit output port P. This output can be connected (cascaded connection) to the adjacent DSP58 internally through the PCOUT path. The PCOUT connects to the input of the Z multiplexer (PCIN) in the adjacent DSP58. This path provides an output cascade stream between adjacent DSP58s.

CARRYCASCOUT and CARRYOUT Ports

The carry out from each DSP58 can be sent to the logic resources using the CARRYOUT port. This port is 4 bits wide. CARRYOUT[3] is the valid carry output for a two-input 58-bit adder/subtractor or one-input accumulator. In this case, USE_SIMD = ONE58 is the default setting and represents a non-SIMD configuration. When a two-input adder/subtractor or one-input accumulator is used in SIMD mode, such as TWO24 or FOUR12, the valid CARRYOUT signals are listed in the following table. The CARRYOUT signals are not valid if three-input (or four-input) adder/subtractor (for example, A:B + C + PCIN) or two-input (or three-input) accumulator (for example, A:B + C + P) configurations are used or if the multiplier is used.

Table 18: CARRYOUT Bit Associated with Different SIMD Modes

SIMD Mode	Adder Bit Width	Corresponding CARRYOUT
FOUR12	P[11:0]	CARRYOUT[0]
	P[23:12]	CARRYOUT[1]
	P[35:24]	CARRYOUT[2]
	P[47:36]	CARRYOUT[3]
TWO24	P[23:0]	CARRYOUT[1]
	P[47:24]	CARRYOUT[3]
ONE58	P[57:0]	CARRYOUT[3]

The CARRYOUT signal is cascaded to the next adjacent DSP58 using the CARRYCASCOUT port. Larger add, subtract, ACC, and MACC functions can be implemented in the DSP58 using the CARRYCASCOUT output. The 1-bit CARRYCASCOUT signal corresponds to CARRYOUT[3], but is not identical. The CARRYCASCOUT signal is also fed back into the same DSP58 through the CARRYINSEL multiplexer.

The CARRYOUT[3] signal should be ignored when the multiplier or a 3-input (or 4-input) add/subtract operation is used. Because a MACC operation includes a three-input adder in the accumulator stage (feedback from the P output and two partial products from the multiplier output), the combination of MULTSIGNOUT and CARRYCASCOUT signals is required to perform a 116-bit MACC, spanning two DSP58s. The second DSP58's OPMODE must be set to MACC_EXTEND (001001000) to use both CARRYCASCOUT and MULTSIGNOUT, thereby eliminating the ternary adder carry restriction for the upper DSP58.

MULTISIGNOUT Logic

MULTISIGNOUT is a software abstraction of the hardware signal. It is modeled as the MSB of the multiplier output and used only in MACC extension applications to build a 116-bit MACC.

The MSB of the multiplier output is cascaded to the next DSP58 using the MULTSIGNIN signal and can be used only in MACC extension applications to build a 116-bit accumulator.

PATTERNDETECT and PATTERNBDETECT Logic

A pattern detector on the output of DSP58 detects if the P bus matches a specified pattern or if it exactly matches the complement of the pattern. The PATTERNDETECT output goes High if the output of the adder matches a set pattern. The PATTERNBDETECT output goes High if the output of the adder matches the complement of the set pattern.

A mask field can also be used to hide certain bit locations in the pattern detector. PATTERNDETECT computes $((P = \text{pattern}) \ll \text{mask})$ on a bitwise basis and then ANDs the results to a single output bit. Similarly, PATTERNBDETECT can detect if $((P = \sim \text{pattern}) \ll \text{mask})$. The pattern and the mask fields can each come from a distinct 58-bit configuration field or from the (registered) C input. When the C input is used as the PATTERN, the OPMODE must be set to select a 0 at the input of the Z multiplexer. If all the registers are reset, PATTERNDETECT is High for one clock cycle immediately after the RESET is deasserted.

The pattern detector allows DSP58 to support convergent rounding and counter auto reset when a count value has been reached as well as support overflow, underflow, and saturation in accumulators.

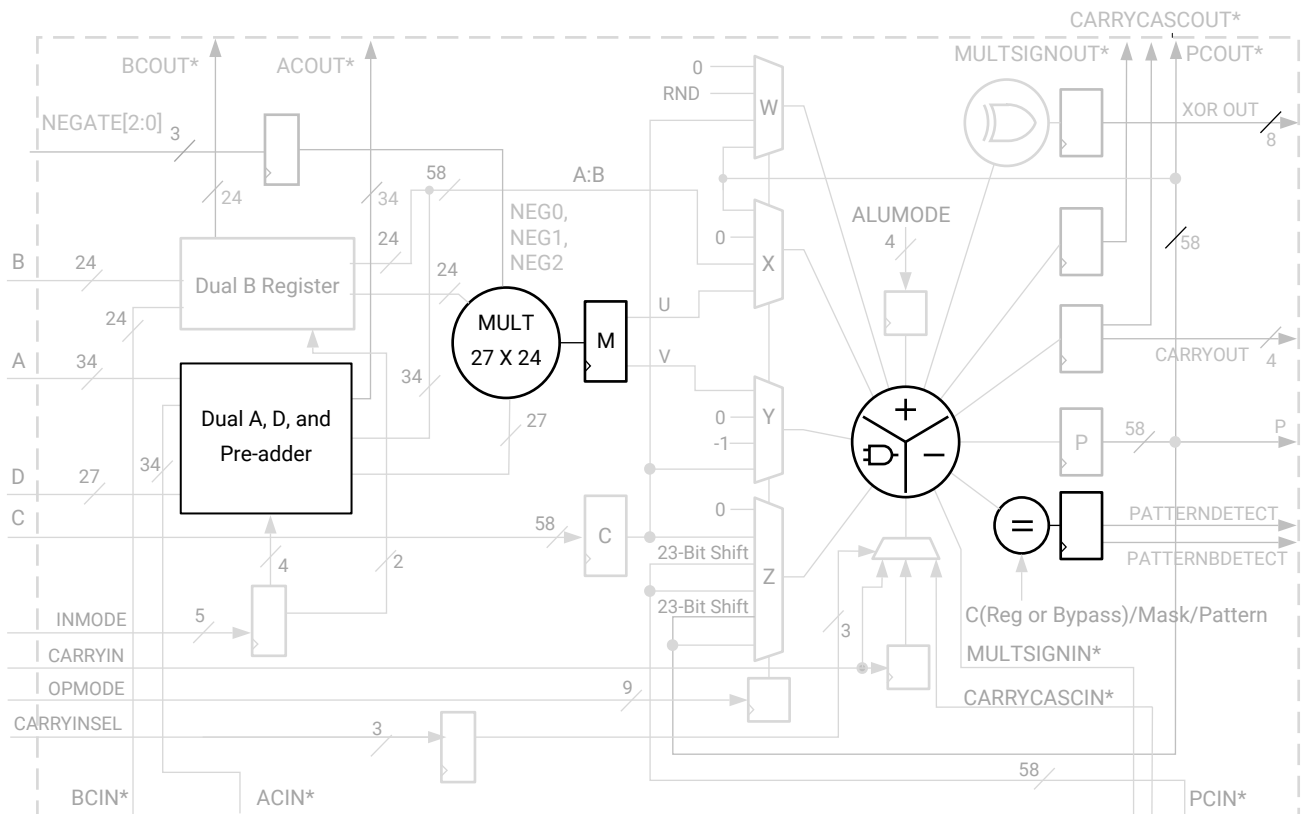
Overflow and Underflow Logic

The dedicated OVERFLOW and UNDERFLOW outputs of DSP58 use the pattern detector to determine if the operation in DSP58 has overflowed beyond the P[N] bit where N is between 1 and 56, and is applied only to a sequential accumulator. The P register must be enabled while using OVERFLOW and UNDERFLOW. This is explained further in the next section.

Embedded Functions

The embedded functions include a 27×24 multiplier, adder/subtractor/logic unit, and pattern detector logic (see the following figure).

Figure 19: Embedded Functions in DSP58



*These signals are dedicated routing paths internal to the DSP58 column. They are not accessible through general-purpose routing resources.

X21363-051420

Pre-Adder

DSP58 has a 27-bit pre-adder, which is inserted in the A or B register path (shown in Figure 19 with an expanded view in Figure 12). With the pre-adder, pre-additions or pre-subtractions are possible prior to feeding the multiplier. The pre-adder does not contain saturation logic and thus, the designers must limit input operands to 26-bit (or 23-bit for the B path) two's complement sign-extended data to avoid overflow or underflow during arithmetic operations. Optionally, the pre-adder can be bypassed, making D the new input path to the multiplier. When the D path is not used, the output of the A or B pipeline can be negated prior to driving the multiplier. There are up to 15 operating modes, including pre-adder squaring, making this pre-adder block very flexible.

In the following equations, A (or B) and D are added initially through the pre-adder/subtractor. The result of the pre-adder is then multiplied against B (or A), with the result of the multiplication being added to the C input. The following equations facilitate efficient symmetric filters.

$$\begin{aligned} \text{Final Adder / Subtractor Output} &= C \pm (\pm B \times (D \pm A) + W + C_{IN}) \text{ or } C \pm (\pm A \times (D \pm B) + W + C_{IN}) \\ \text{Final Adder / Subtractor Output} &= C \pm (\pm B \times (D \pm B) + W + C_{IN}) \text{ or } C \pm (\pm A \times (D \pm A) + W + C_{IN}) \end{aligned}$$

$$\text{Final Adder / Subtractor Output} = C \pm (\pm(D \pm B)^2 + W + C_{IN}) \text{ or } C \pm (\pm(D \pm A)^2 + W + C_{IN})$$

Two's Complement Multiplier

The two's complement multiplier in DSP58 in [Figure 19](#) accepts a 27-bit two's complement input and a 24-bit two's complement input along with a conditional product negation control input bit. The multiplier produces two 51-bit partial products. The two partial products together give an 51-bit result at the output of the multiplier, as shown in the following figure.

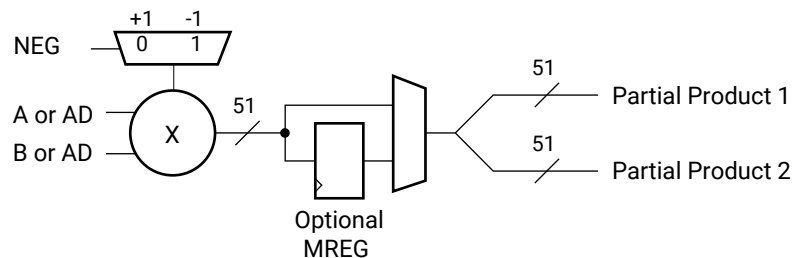
Note: The product-negation input does not cause overflow and is safe to use for all values of the multiplier inputs. Specifically, an N-bit two's complement number is in the range $[-2^{N-1}, 2^{N-1} - 1]$.

All values in this range can be negated to fit within N bits except -2^{N-1} , the most negative integer. But the DSP58 negates the product, not the inputs. The output of the multiplier has an inversion of the sign if $\text{Sign}(A) \neq \text{Sign}(B)$. The indicator makes the inputs appear to have grown but a wider multiplier is unnecessary to avoid overflow. As a result overflow can be avoided even if one or both of the multiplier inputs are the most negative numbers. The DSP58 multiplier can compute the absolute value of a two's complement number up to 27 bits. Define the DSP58 multiplier as $\text{mult}(A[26:0], [B[23:0], p) \equiv (-1)^p AB$, where A and B are the 27- and 24-bit inputs, and p is the Boolean product-negation control input. Then the absolute value of A can be computed as $\text{mult}(A[26:0], 1, A[26])$.

Cascading of multipliers to achieve larger products is supported with a 23-bit, right-shifted, cascaded output bus. The right shift is used to right justify the partial products by the correct number of bits. This cascade path feeds into the Z multiplexer, which is connected to the adder/subtractor of an adjacent DSP58. The multiplier can emulate unsigned math by setting the MSB of an input operand to zero.

The following figure shows an optional pipeline register (MREG) for the output of the multiplier. Using the register provides increased performance with an increase of one clock latency.

Figure 20: Two's Complement Multiplier Followed by Optional MREG



X21365-100618

Adder/Subtractor or Logic Unit

The adder/subtractor or logic unit output is a function of control and data inputs (see [Figure 21](#)). The data inputs to the adder/subtractor are selected by the OPMODE and the CARRYINSEL signals. The ALUMODE signals choose the function implemented in the adder/subtractor. Thus, the OPMODE, ALUMODE, and CARRYINSEL signals together determine the functionality of the embedded adder/subtractor/logic unit. When using the logic unit, the multiplier must not be used. The values of OPMODEREG and CARRYINSELREG must be identical.

As with the input multiplexers, the OPMODE bits specify a portion of this function. The symbol \pm in the table means either add or subtract and is specified by the state of the ALUMODE control signal. The symbol $:$ in the table means concatenation. The outputs of the X and Y multiplexer and CIN are always added together. For more information, refer to [ALUMODE Inputs](#).

Two-Input Logic Unit or Three-Input XOR Special Case

The capability to perform an addition, subtraction, and simple logic functions in DSP58 exists through the use of a second-stage, four-input adder. The following table lists the logic functions that can be implemented in the second stage of the four input adder/subtractor/logic unit. The table also lists the settings of the OPMODE and ALUMODE control signals.

Setting OPMODE[3:2] to 00 selects the default 0 value at the Y multiplexer output. OPMODE[3:2] set to 10 selects all 1s at the Y multiplexer output. OPMODE[1:0] selects the output of the X multiplexer, OPMODE[6:4] selects the output of the Z multiplexer. For two-input or three-input logic operations, OPMODE[8:7] must be set to 00 for the default all 0s value at the W multiplexer output.

An XOR3 can be built by setting the OPMODE[3:2] to 11, selecting the C input at the Y multiplexer output. The XOR3 is only valid for ALUMODE[3:0] = 0100, as shown in the following table.

Table 19: OPMODE and ALUMODE Control Bits Select Logic Unit Outputs

Logic Unit Mode	OPMODE[3:2]		ALUMODE[3:0]			
	3	2	3	2	1	0
X XOR Z	0	0	0	1	0	0
X XNOR Z	0	0	0	1	0	1
X XNOR Z	0	0	0	1	1	0
X XOR Z	0	0	0	1	1	1
X AND Z	0	0	1	1	0	0
X AND (NOT Z)	0	0	1	1	0	1
X NAND Z	0	0	1	1	1	0
(NOT X) OR Z	0	0	1	1	1	1
X XNOR Z	1	0	0	1	0	0

Table 19: OPMODE and ALUMODE Control Bits Select Logic Unit Outputs (cont'd)

Logic Unit Mode	OPMODE[3:2]		ALUMODE[3:0]			
	3	2	3	2	1	0
X XOR Z	1	0	0	1	0	1
X XOR Z	1	0	0	1	1	0
X XNOR Z	1	0	0	1	1	1
X OR Z	1	0	1	1	0	0
X OR (NOT Z)	1	0	1	1	0	1
X NOR Z	1	0	1	1	1	0
(NOT X) AND Z	1	0	1	1	1	1
X XOR Y XOR Z ¹	1	1	0	1	0	0

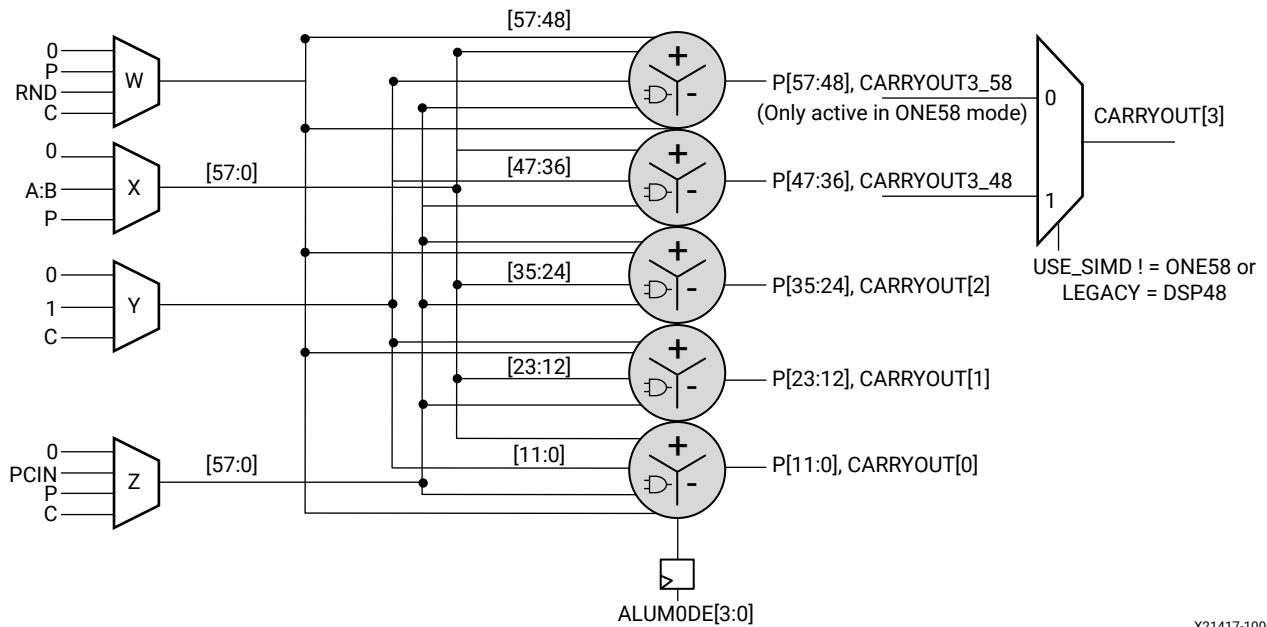
Notes:

1. Valid when Y multiplexer selects C input.

Single Instruction, Multiple Data (SIMD) Mode

The 58-bit adder/subtractor/accumulator can be split into smaller data segments where the internal carry propagation between segments is blocked to ensure independent operation for all segments. The adder/subtractor/accumulator can be split into four 12-bit adder/subtractor/accumulators or two 24-bit adder/subtractor/accumulators with carry out signal per segment. The SIMD mode segmentation is a static configuration as opposed to dynamic OPMODE type control (see the following figure). In all non-58-bit SIMD modes the upper 10-bits of the adder/subtractor/accumulator is disabled. This is done by driving the W/X/Y/Z inputs on the upper 10-bits to a static 0. This means that in TWO24 and FOUR12 modes the DSP58 operates the same as the DSP48E2, with the upper 10-bits of the P output equaling zero.

Figure 21: SIMD Adder Configuration



X21417-100618

- Four segments of dual, ternary, or quad adders with 12-bit inputs, a 12-bit output, and a carry output for each segment
- Function controlled dynamically by ALUMODE[3:0], and operand source by OPMODE[8:0]
- All four adder/subtractor/accumulators perform same function
- Two segments of dual, ternary, or quad adders with 24-bit inputs, a 24-bit output, and a carry output for each segment is also available (not pictured).

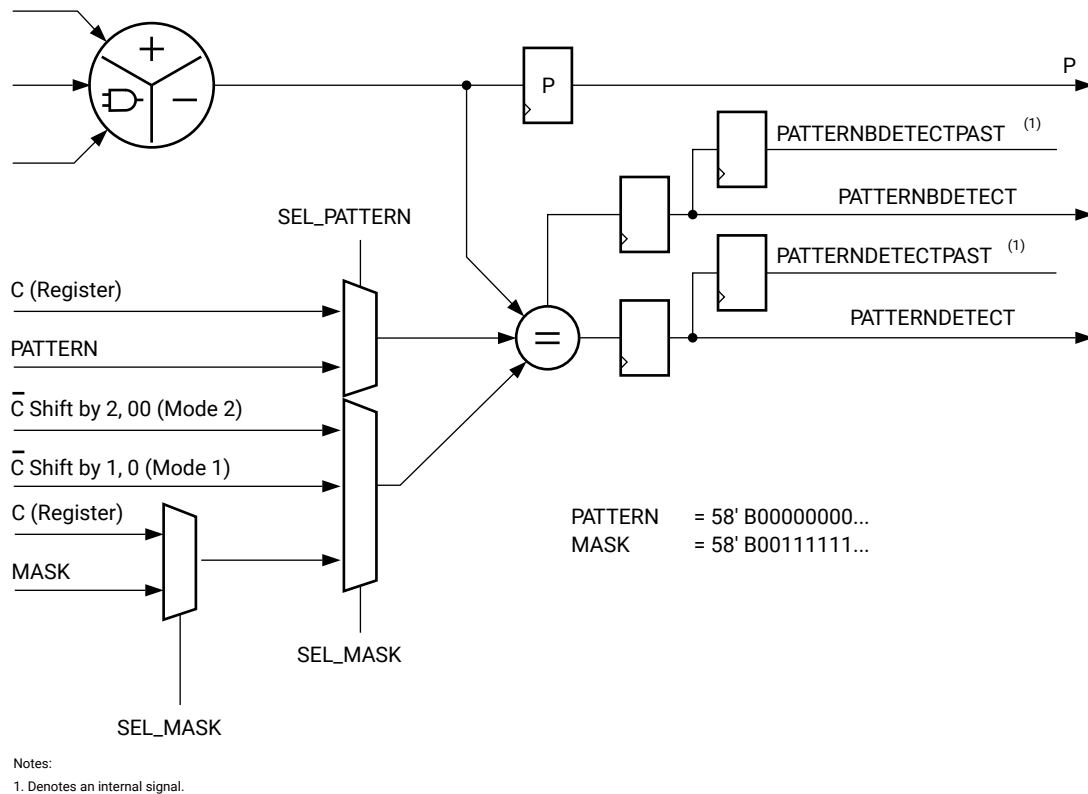
The SIMD feature allows the 58-bit logic unit to be split into multiple smaller logic units (see the figure above). Each smaller logic unit performs the same function. This function can also be changed dynamically through the ALUMODE[3:0] and opmode control inputs.

Note: The Carry-In input selected by the CARRYINSEL only propagates to the lowest-order adder in SIMD mode, that is, P[11:0] in FOUR12 and P[23:0] in TWO24 modes.

Pattern Detect Logic

The pattern detector is connected to the output of the add/subtract/logic unit in DSP58. The pattern detector is best described as an equality check on the output of the adder/subtractor/logic unit that produces its result on the same cycle as the P output. There is no extra latency between the pattern detect output and the P output of DSP58. The use of the pattern detector leads to a moderate speed reduction due to the extra logic on the pattern detect path (see the following figure).

Figure 22: Pattern Detector Logic



X21385-082818

Some of the applications that can be implemented using the pattern detector are:

- Pattern detect with optional mask
- Dynamic C input pattern match with A x B
- Overflow/underflow/saturation past P[56] (or P[46] when DSP_MODE = DSP48E2)
- A:B == C and dynamic pattern match, for example, A:B OR C == 0, A:B AND C == 1
- A:B {function} C == 0
- 58-bit (48-bit for DSP_MODE = DSP48E2) counter auto reset (terminal count detection) with option for CEP priority
- Detecting mid points for rounding operations

If the pattern detector is not being employed, it can be used for other creative design implementations. These include:

- Duplicating a pin (for example, the sign bit) to reduce fanout and thus increase speed.
- Implementing a built-in inverter on one bit (for example, the sign bit) without having to route out to the CLBs.

- Checking for sticky bits in floating-point, handling special cases, or monitoring DSP58 outputs.
- Raising a flag if a certain condition is met or if a certain condition is no longer met.

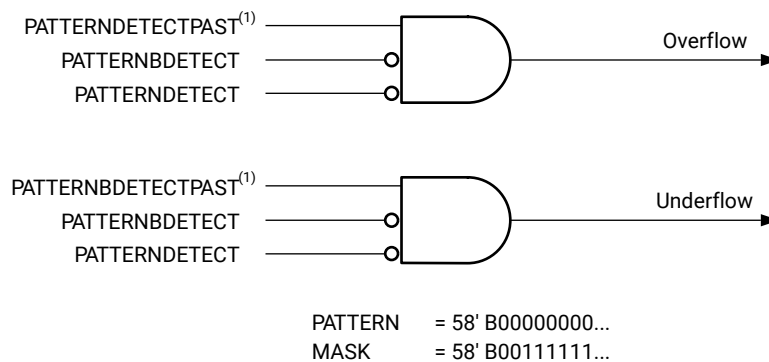
A mask field can also be used to mask out certain bit locations in the pattern detector. The pattern field and the mask field can each come from a distinct 58-bit memory cell field or from the (registered) C input.

Note: For the mask field, all 1's are not valid because it would make PATTERNDETECT always equal to 1.

Overflow and Underflow Logic

The following discussion of overflow and underflow applies to sequential accumulators (MACC or Adder-Accumulator) implemented in a single DSP58. The accumulator must have at least one guard bit. When the pattern detector is set to detect a pattern equal to 00000...0 with a mask of 0011111 ...1 (default settings), DSP58 flags overflow beyond 00111 ... 1 or underflow beyond 11000... 0. The USE_PATTERN_DETECT attribute is set to PATDET to enable the use of the pattern detect logic. This overflow/underflow implementation uses a redundant sign bit and reduces the output bit width to 57 bits.

Figure 23: Overflow/Underflow Logic in Pattern Detect



Notes:
1. Denotes an internal signal.

X21391-041119

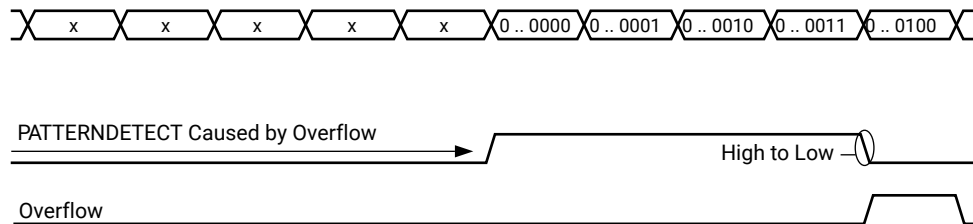
By setting the mask to other values like 0000111 ...1, the bit value P[N] at which overflow is detected can be changed. This logic supports saturation to a positive number of $2^N - 1$ and a negative number of 2^N in two's complement where N is the number of 1s in the mask field.

To check overflow/underflow condition for N = 2, the following example is used:

- Mask is set to 0 . . . 11.
- The (N) LSB bits are not considered for the comparison.
- For N = 2, the legal values (patterns) are $2^2 - 1$ to -2^2 , or 3 to -4.

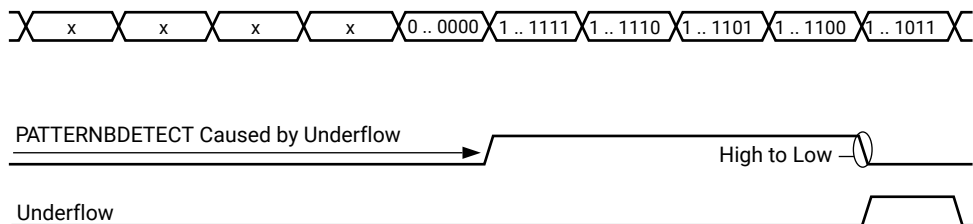
See the following figures for overflow and underflow examples. The pattern detect logic asserts the overflow/underflow signal for only one clock cycle in the same cycle in which the P output that caused the overflow/underflow is produced.

Figure 24: **Overflow Condition in the Pattern Detector**



X21392-082818

Figure 25: **Underflow Condition in the Pattern Detector**



X21393-100718

- PATTERNDETECT is 1 if $P = \text{pattern}$ or mask
- PATTERNBDETECT is a 1 if $P = \text{patternb}$ or mask

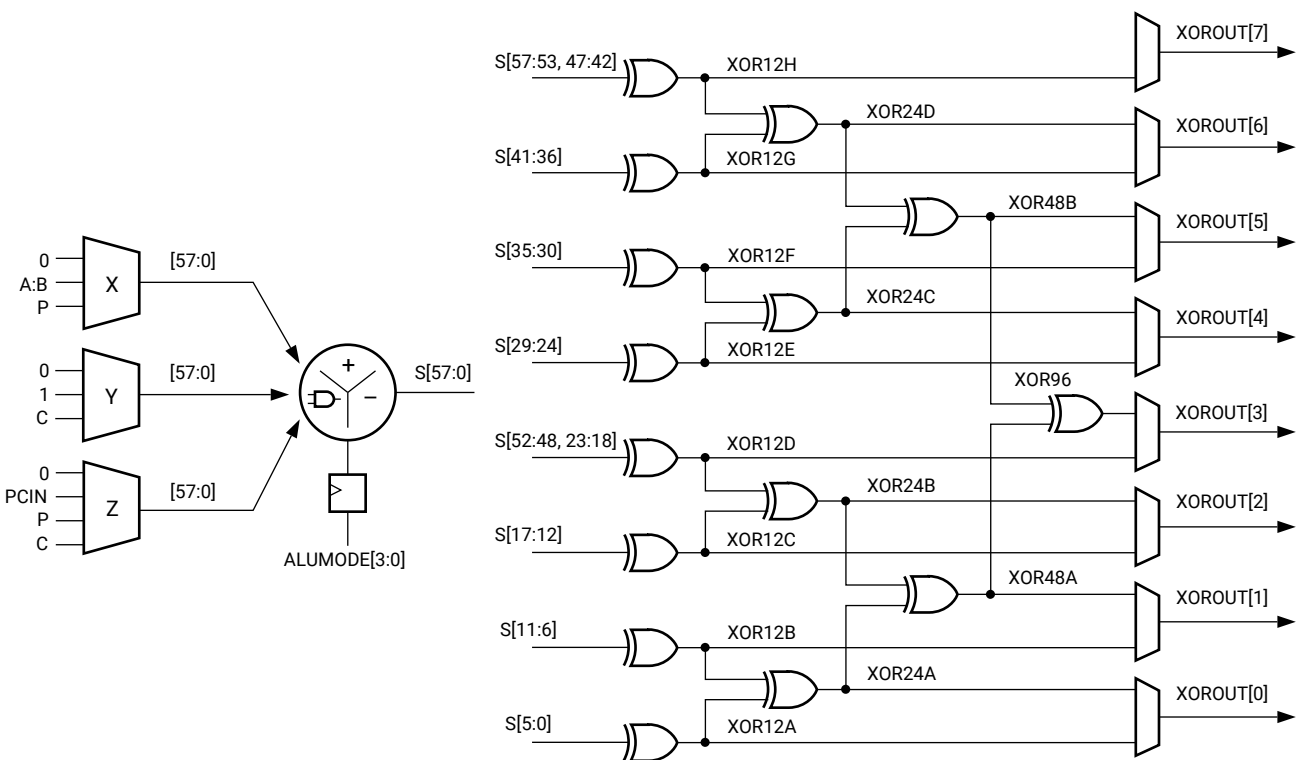
Overflow is caused by addition when the value at the output of the adder/subtractor/logic unit goes over 3. Adding 1 to the final value of 0..0011 gives 0..0100 as the result, which causes the PATTERNDETECT output to go to 0. When the PATTERNDETECT output goes from 1 to 0, an overflow is flagged.

Underflow is caused by subtraction when the value goes below -4. Subtracting 1 from 1..1100 yields 1..1011 (-5), which causes the PATTERNBDETECT output to go to 0. When the PATTERNBDETECT output goes from 1 to 0, an underflow is flagged.

Wide XOR

DSP58 has the ability to perform a 116-bit wide XOR function. The XOR uses the X, Y, and Z multiplexers as inputs. The W multiplexer selects all 0s at its output. The ALU logic is used for the first stage of the wide XOR by using the proper OPMODE and ALUMODE signals as shown in Table 19, to implement either $X \text{ XOR } Z$, or $X \text{ XOR } Y \text{ XOR } Z$. The signals then branch out to an XOR logic tree with dedicated outputs. Multiplexers allow selection as six 12-bit and two 22-bit wide XOR, two 24-bit and two 34-bit wide XOR, two 58-bit wide XOR, or one 116-bit wide XOR (see the following figure). In the following figure, the S[57:0] internal bus is not the P[57:0] output, it is one of the 4:2 compressor buses.

Figure 26: Wide XOR Function in ALU



X21333-051420

The XORSIMD attribute is used to select the width of the XOR function as either 116-bit or 12/22/24/34/58 bits, as shown in the following table.

Table 20: XOR9_XOR SIMD Mode Bits

XORSIMD Attribute	XOR Width	XOR INPUT Bits (A:B^C)	Corresponding XOROUT
XOR12_22	6 × 12-bit 2 × 22-bit	S[5:0]	XOROUT[0]
		S[11:6]	XOROUT[1]
		S[17:12]	XOROUT[2]
		S[52:48, 23:18]	XOROUT[3]
		S[29:24]	XOROUT[4]
		S[35:30]	XOROUT[5]
		S[41:36]	XOROUT[6]
		S[57:53, 47:42]	XOROUT[7]
XOR24_34_58_116	2 × 24-bit 2 × 34-bit	S[11:0]	XOROUT[0]
		S[52:48, 23:12]	XOROUT[2]
		S[35:24]	XOROUT[4]
		S[57:53, 47:36]	XOROUT[6]
	2 × 58-bit	S[52:48, 23:0]	XOROUT[1]
		S[57:53, 47:24]	XOROUT[5]
	1 × 116-bit	S[57:0]	XOROUT[3]

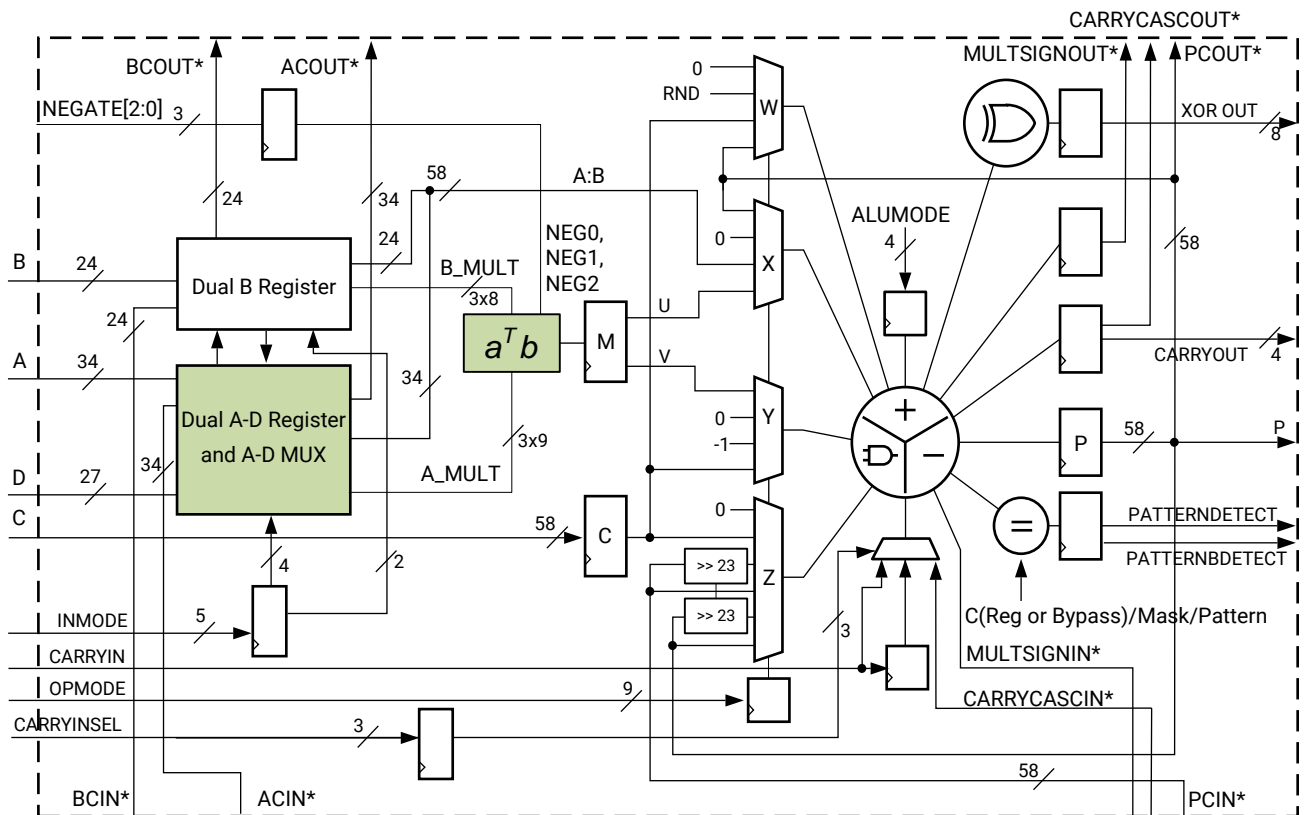
The first level XOR can either be XOR2 or XOR3. In both cases, ALUMODE[3:0] = 0100 for the XOR function in the ALU. When the Y multiplexer selects 0, an XOR2 is created. When the Y multiplexer selects the C register, an XOR3 is created, supporting up to 58 XOR3 in the ALU. The third input can come from the P output or the PCIN cascade, which provides XOR-accumulate and cascade capability for even wider XOR functions.

Vector Fixed-Point ALU

Overview

This chapter provides details of DSP58 in the vector fixed-point ALU mode. As shown in the following figure, DSP58 performs identically to the scalar fixed-point except for the three-dimensional vector dot product unit that replaces the 27×24 multiplier.

Figure 27: Detailed DSP58 Functions as a Vector Fixed-Point ALU



*These signals are dedicated routing paths internal to the DSP58 column. They are not accessible through general-purpose routing resources.

X21270-051420

Dot Product Unit

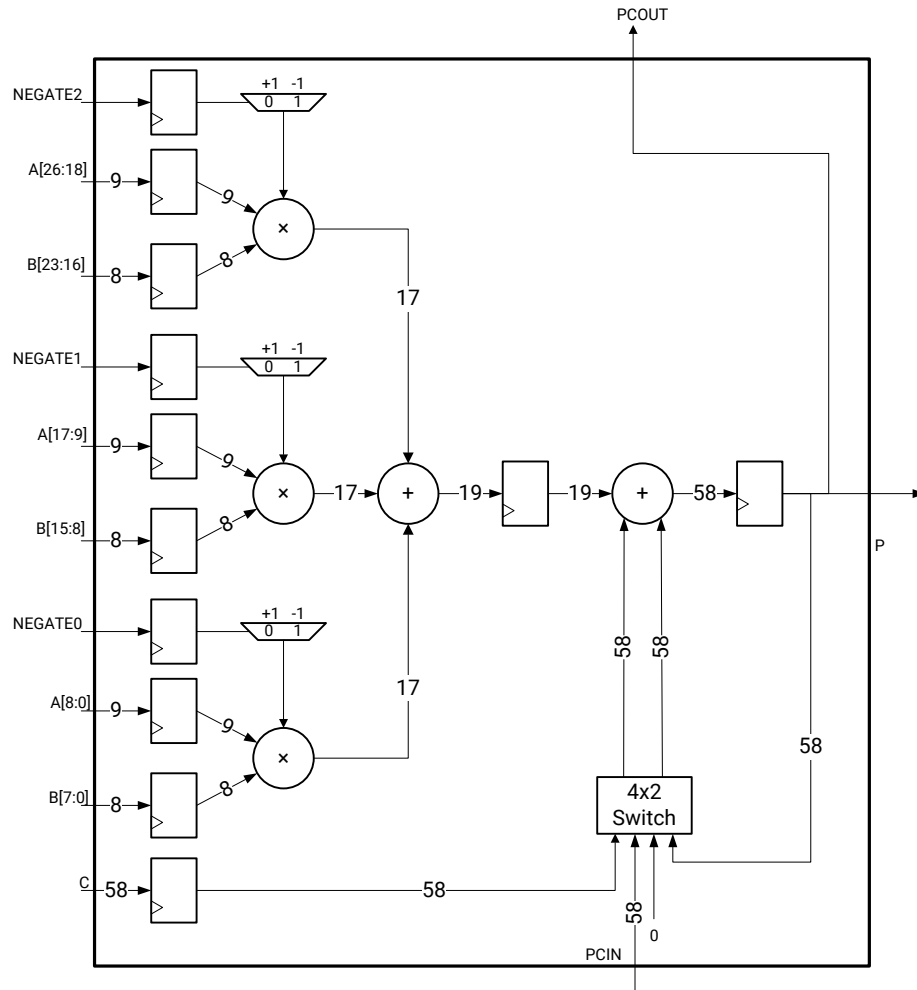
The INT8 mode is used to implement the dot product unit (see the following figure for a simplified block diagram), optimized for neural network and machine learning applications. It computes the inner product of two vectors $a = (a_0, a_1, a_2)$ and $b = (b_0, b_1, b_2)$ where a_i is a 9-bit two's complement fixed point number and b_i is an 8-bit two's complement number. The dot product also supports element-wise product negation with the pins NEGATE[2:0]. This unit computes the partial results u and v such that:

$$u + v = \pm a_0 b_0 \pm a_1 b_1 \pm a_2 b_2$$

Taking pipelining into account, the dot product unit computes the following:

$$(-1)^{NEG0} AMULT[8:0]BMULT[7:0] + (-1)^{NEG1} AMULT[17:9]BMULT[17:9] + (-1)^{NEG2} AMULT[26:18]BMULT[26:18]$$

Where NEG0, NEG1, and NEG2 are registered versions of NEGATE[2:0] or the signals themselves. AMULT and BMULT are the vector 9×8 multiplier inputs. See [Figure 27](#). Similar to the scalar fixed-point ALU, the negation pins do not cause internal overflow, even if any of the inputs is the most negative two's complement number (all zeros except for the sign bit). Because the dot-product unit is derived from the scalar ALU, other than the pre-adder and the 27×24 multiplier, all other features function identically.

Figure 28: Simplified Two's Complement Dot Product Unit (INT8 mode)


X20611-080618

Note: All input signals can bypass their input registers.

Pre-Adder Used as a Multiplexer

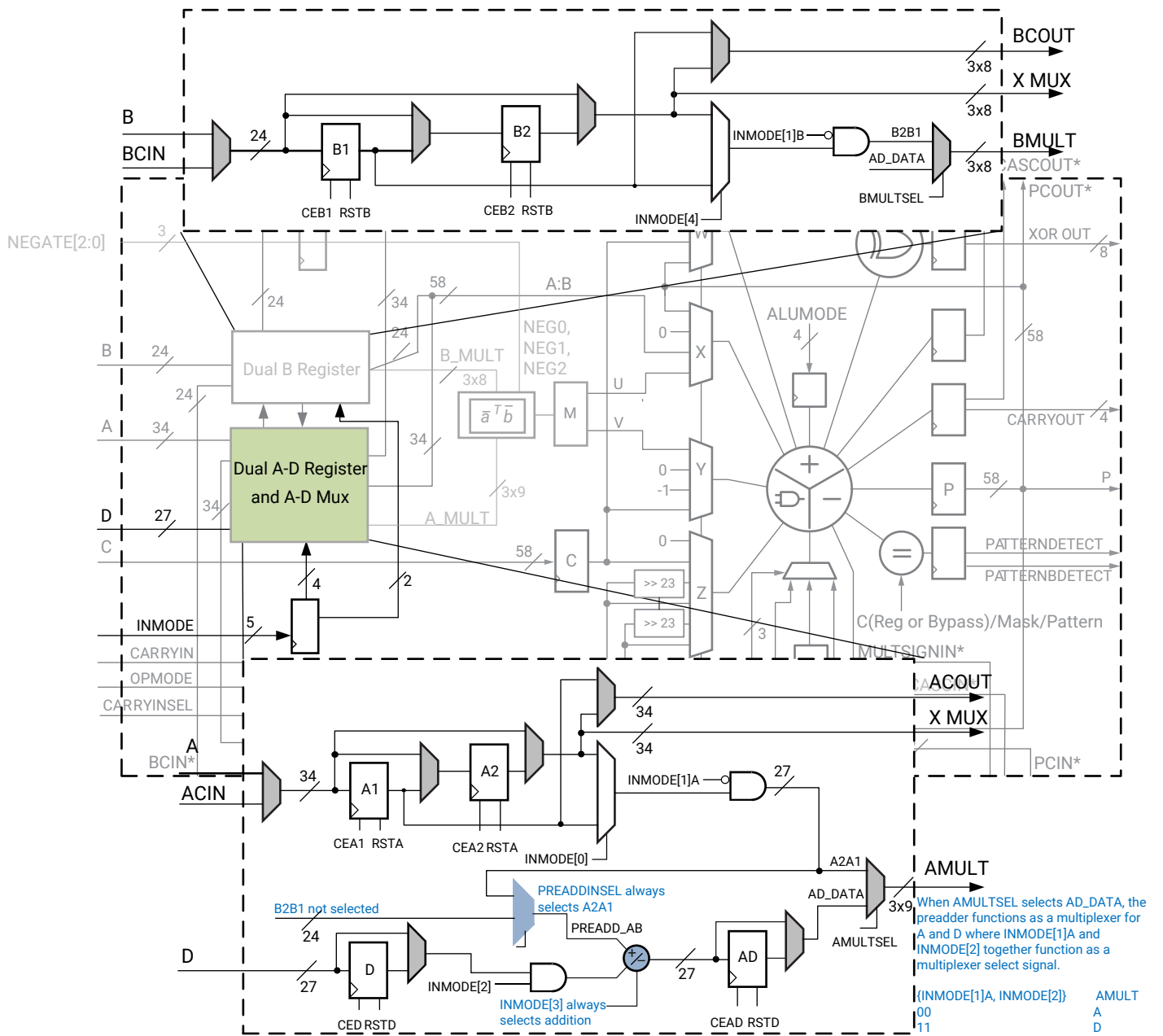
The pre-adder can be reused as a 2:1 multiplexer to select between the A[26:0] and D[26:0], useful for connecting ping-pong buffers implemented in the programmable logic (PL) memory. The idea is to make the pre-adder perform either $A + 0$, or $0 + D$, where A receives data from the ping buffer and D receives data from the pong buffer. See the following figure. The following configurations are included.

1. Configure AMULT to receive data from AD_DATA with AMULTSEL.
2. Configure the pre-adder to select A2A1 with PREADDINSEL because B is narrower than A.

3. Configure the pre-adder to the add mode with `INMODE[3]` so that the pre-adder to compute either $A + 0$, or $0 + D$.
4. Connect the A-D multiplexer select pin from the PL to both `INMODE[1]A` and `INMODE[2]`. When these signals are both 0, A is selected to go to `AMULT`, the input of the vector multiplier; when they are both 1, D is selected.

Note: The pre-adder must not be used for selecting between B and D because B only has 24 bits whereas D has 27 bits.

Figure 29: Hierarchical View of the DSP58 Input Registers and Pre-Adder as a Multiplexer



X21452-110320

Complex Arithmetic Unit

Basic Function and Complex Adder

Two back-to-back DSP58s form one complex arithmetic unit. The two DSP58s together compute,

$$\tilde{P} = \tilde{Z} \pm (\tilde{A} \times \tilde{B} + \tilde{W} + \tilde{C}_{IN})$$

where \tilde{A} and \tilde{B} are 18-bit complex two's complement numbers (18 bits real and 18 bits imaginary), \tilde{W} , \tilde{Z} , and \tilde{P} are complex two's complement 58-bit fixed-point numbers and \tilde{C}_{in} is a one-bit complex carry-in.

The right DSP58 computes (when $CONJUGATE_A = CONJUGATE_B = 0$),

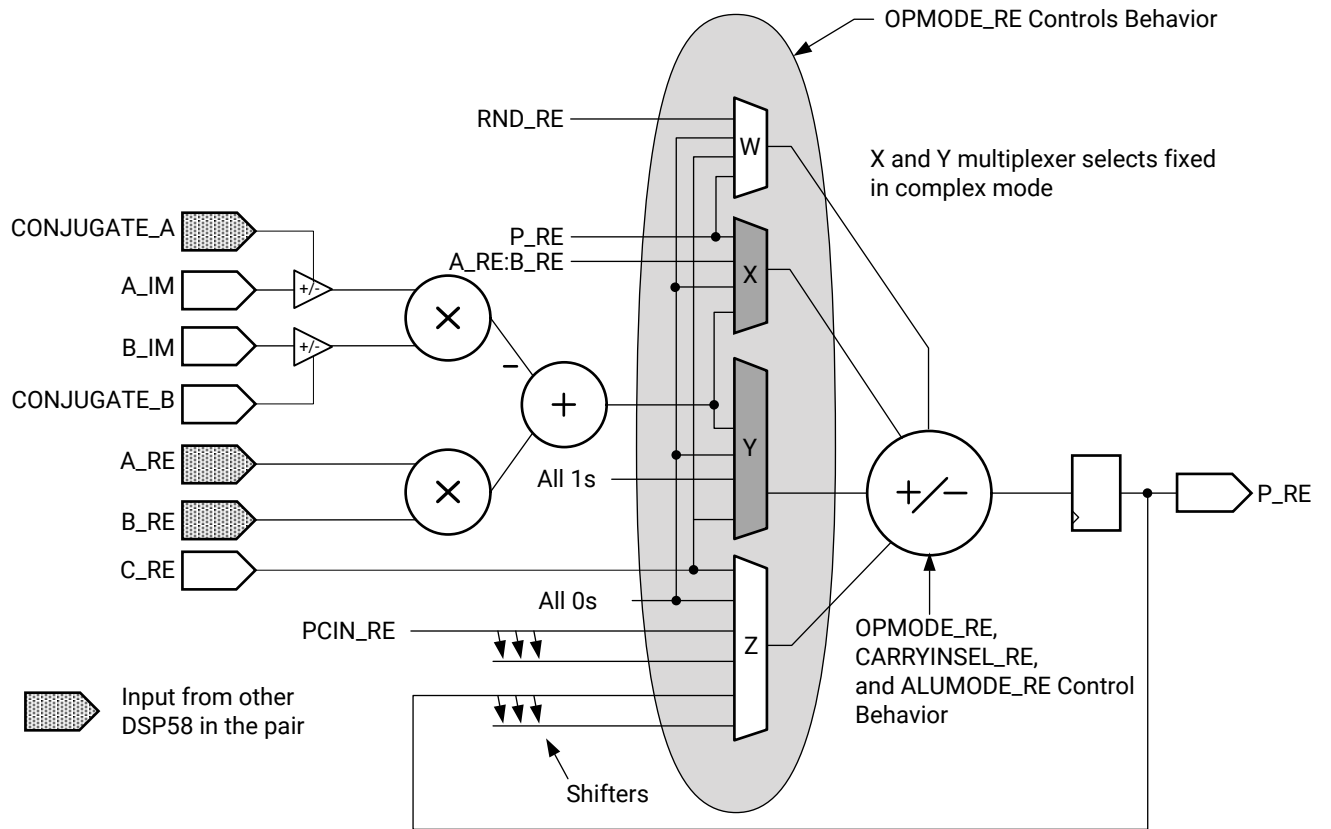
$$P_{RE}[57:0] = \pm Z_{RE}[57:0] \pm (A_{RE}[17:0] \times B_{RE}[17:0] - A_{IM}[17:0] \times B_{IM}[17:0] + W_{RE}[57:0] + C_{IN,RE})$$

while the left DSP58 computes in parallel,

$$P_{IM}[57:0] = \pm Z_{IM}[57:0] \pm (A_{RE}[17:0] \times B_{IM}[17:0] + A_{IM}[17:0] \times B_{RE}[17:0] + W_{IM}[57:0] + C_{IN,IM})$$

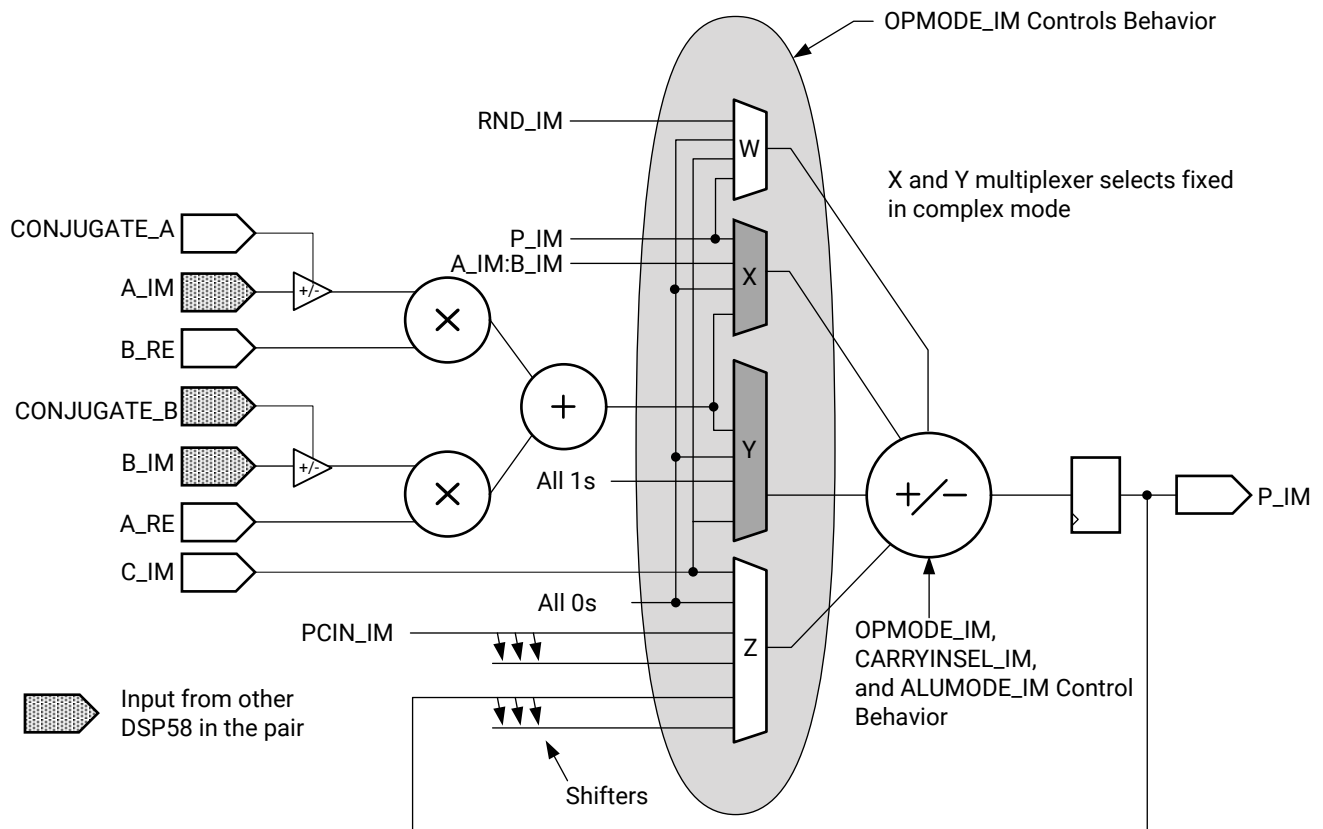
As shown in the following figures, the right DSP58 computes the real result P_{RE} and the left DSP58 computes the imaginary result P_{IM} .

Figure 30: Right DSP58 in the Complex Arithmetic Unit to Generate the Real Part of the Result



X20612-051420

Figure 31: Left DSP58 in the Complex Arithmetic Unit to Generate the Imaginary Part of the Result



X20613-052620

Complex Multiplier

- Generation of Complex Conjugates for Multiplication:** The 18-bit complex multiplier that straddles two back-to-back DSP58s produces a complex product. See the complex multiplier example in [Chapter 8: DSP58 Applications](#) for details on generation of complex conjugates and coding example.
- Multiplier Input Sign Extension:** Both \tilde{A} and \tilde{B} are 18-bit complex values. Even though both the A and B ports of DSP58 are wider than 18 bits, there is no need to sign-extend \tilde{A} and \tilde{B} .
- OPMODE Control:** The complex multiplier generates a partial product that occupies both the X and the Y multiplexers of the back-to-back DSP58s, leaving the W and the Z multiplexers for two more complex inputs to go into the complex adder. Similar to the scalar fixed-point ALU, the W and Z multiplexers in the two DSP58s can therefore accept complex inputs from two of the following four sources:
 1. The PL (the two C ports, C_{RE} and C_{IM}).

2. The complex accumulator (the two P registers, P_{RE} and P_{IM}).
3. The complex result from an upstream DSP58 pair along the cascade (from $PCIN_{RE}$ and $PCIN_{IM}$).
4. The complex constant (in the two internal RND constants, RND_{RE} and RND_{IM}).

DSPCPLX Pipeline Configuration

In the CINT18 mode, the complete implementation of the complex multiplication algorithm requires the preadders and multipliers in both the DSP58s, and the common section to work together thereby allowing only specific pipeline configuration to be valid. Programmability for each pipeline stage visible to the user in the DSPCPLX Unisim is allowed. However, invalid attribute values generate a software DRC that instructs the user to change their attribute settings. The following table specifies the valid attribute values for the complex mode. The figure that follows illustrates a simplified block diagram of the complex multiply accumulator.

Note: The conjugate inputs labeled as *_A and *_B are intended to demonstrate that these are relative to the two complex inputs rather than the real and imaginary parts. Also, the labels A_RE, B_RE, A_IM, and B_IM show the side the signals come in physically. Internally the signals go to both DSP58s.

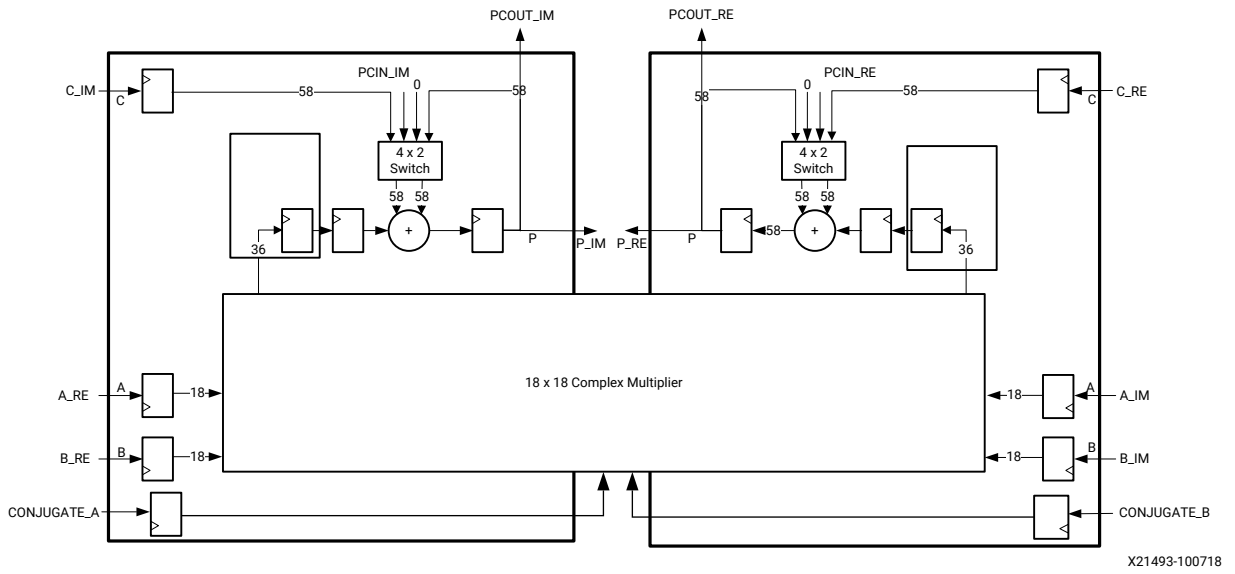
Table 21: CINT18 Mode Programmable Register Attributes

Case Number	AREG_RE	BREG_RE	DREG_RE = AREG_RE	ADREG	AREG_IM	BREG_IM	DREG_IM	Notes	Register Delay Required on CONJUGATE Inputs (RE and IM) ¹
1	0	0	0	0	0	0	0	Valid Case, ALL REG bypassed	0
2	1	1	1	0	1	1	1	Valid Case Balanced Pipeline	1
3	2	2	1	1	2	2	1	Valid Case Fully Pipeline	1

Notes:

1. The value of the register delay required on the CONJUGATE inputs (both RE and IM) can be achieved by either setting $CONJUGATEREG = 1$ or balancing the inmode input in the programmable logic (PL) by one clock cycle (adding latency in the PL and setting $CONJUGATEREG = 0$).

Figure 32: CINT18 Mode 18 × 18 Complex Multiplier and 58 + 58 Complex Accumulator Simplified Block Diagram



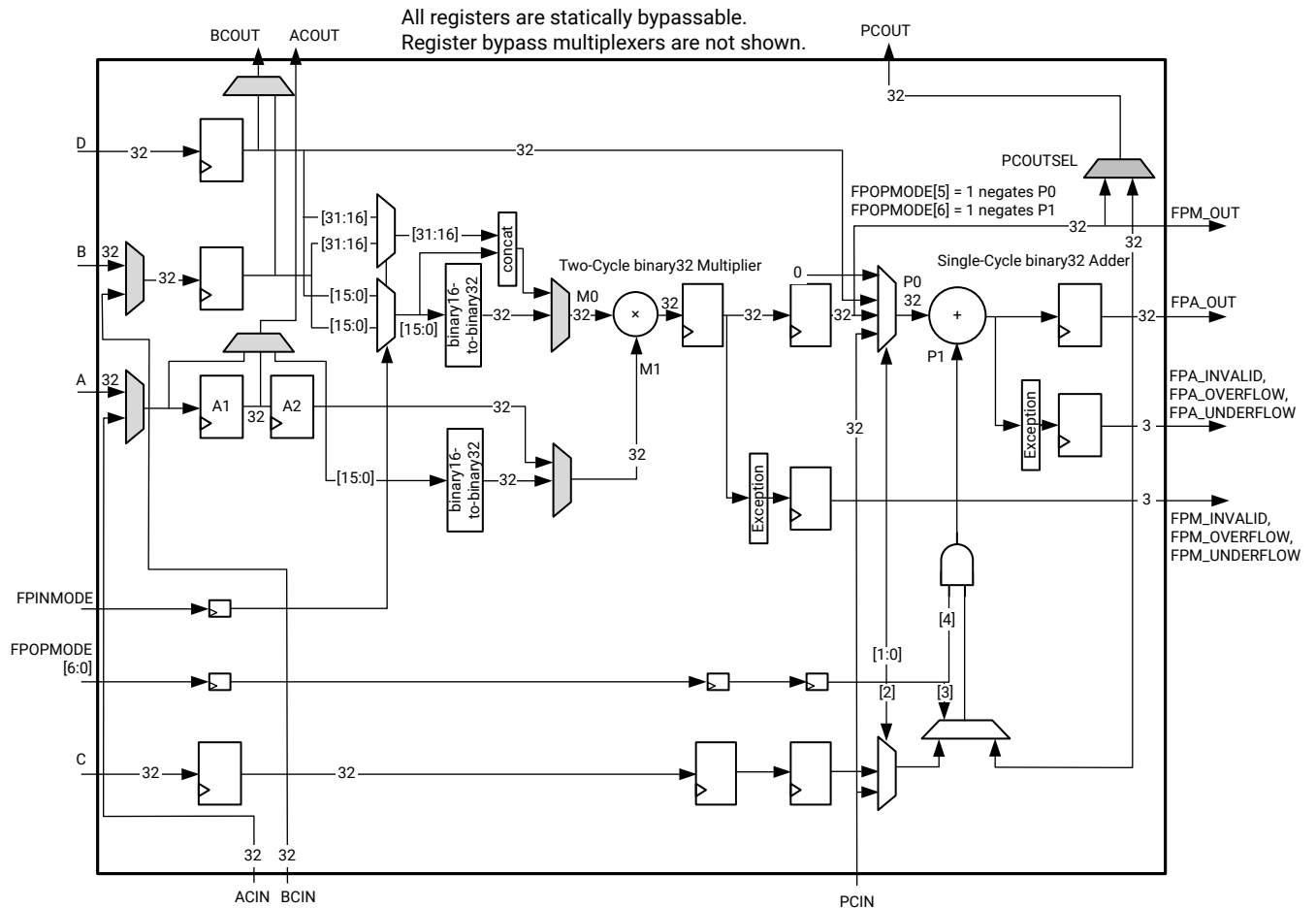
Floating-Point Arithmetic Unit

DSPFP32 Unisim Primitive

The DSPFP32 consists of a floating point multiplier and a floating point adder with separate outputs in the binary 32 format going into the internal logic. Each floating point multiplier input can be in either the IEEE binary32 (FP32 or single-precision) or binary16 (FP16 or half-precision) format, whereas the floating point adder only accepts binary32 inputs. Outputs are always in FP32 or single precision format. The adder has an internal loop-back path to form an accumulator in a single cycle. The multiplier output can also feed the adder internally without logic routing to form a multiply-add (MADD) or multiply-accumulate (MACC) unit. Alternatively, the P output of the FP adder (FPA) can feedback to a MUX with the C input with FPCREG = 3 to form an accumulator chain supporting up to four threads of MACC computation.

The two outputs of the floating point unit are $M = A \times M0$ and $P = \pm P0 \pm P1$, see the following figure. Both M and P are rounded and only the IEEE round-to-nearest-even mode is supported.

Note: M0 is an internal node that can receive input from either input B or port D controlled by FPINMODE because of the ping-pong input feature.

Figure 33: Floating Point Multiplier and Adder (DSPFP32 Mode)


X20614-041519

Operational Modes

Output and Rounding Modes

The two outputs of the DSPFP32 are $FPM_OUT = M1 \times M0$ and $FPA_OUT = \pm P0 \pm P1$ (see [Figure 33](#)). Both M and P are rounded and only the IEEE round-to-nearest-even mode, as explained in the [IEEE Standard for Floating-Point Arithmetic](#), is supported.

Mixed Numerical Formats

M, P0, P1, and P are always in the binary32 format, whereas M1 and M0 (which is either B or D) can be configured statically to be in the binary32 or the binary16 format—independently. The input cascades (ACIN-ACOUT and BCIN-BCOUT) use the same formats as A and B/D (see the following table). Because the adder can accept D as an input, when D is configured as a binary16 input for the multiplier, D cannot be used as an input to the binary32 adder.

Note: When FP data (single precision or half precision) is cascaded, the *_FPTYPE attributes for both the source and the receiving DSPs must match (for example, if A_FPTYPE on the source DSP = B16, then the receiving DSP must also have A_FPTYPE = B16).

Table 22: Multiplier Input Format Combination

A_FPTYPE Attribute	B_D_FPTYPE Attribute	A, ACIN, ACOUT	B, BCIN, BCOUT, D
B32	B32	FP32 format	FP32 format
B32	B16	FP32 format	FP16 format
B16	B32	FP16 format	FP32 format
B16	B16	FP16 format	FP16 format

Arithmetic Modes

FPINMODE and FPOPMODE[6:0] controls the behavior of the floating-point arithmetic unit.

Multiplier Input Selection

FPINMODE selects between B and D to feed the multiplier input M0. The other input of the multiplier, M1, is always a function of the primary input A.

Adder Input Selection

1. FPOPMODE[1:0] selects between the product M, PCIN, the internal input D or 0 to feed the adder input P0. When D carries binary16 data for the multiplier, it cannot be used for the binary32 adder.
2. FPOPMODE[4:2] selects between P, PCIN, C, or 0 to feed the adder input P1.
 - P is the adder output. When looped back to P1, the P register is the accumulator. Alternatively, P can be looped back to C with programmable logic (PL) resources to form an accumulator chain that is up to four registers long, enabling up to four independent threads to be time-interleaved. For instance, one input element from A can be processed by four linear filters with coefficients at the B input. The results from the four filters are shifted in the four-accumulator loop. While the four filters context switches in a round-robin fashion at input B every cycle, the input element on A changes only once every four cycles and reduces power.
 - PCIN is the P output of the downstream DSP58 in a cascade.

- C is an external input to the adder. Together with FPOPMODE[1:0], the adder can be used independently from the multiplier to compute $P = C + D$.
- The input 0 is used to pass P0 directly to P. It is used, for instance, to initialize the accumulator to the first product in a sum of products (vector dot-product) calculation. FPOPMODE[4] is a single pin to initialize the accumulator.

The following tables outline the selection of P1 and P0.

Table 23: Selecting P1 with FPOPMODE[4:2]

FPOPMODE[4]	FPOPMODE[3]	FPOPMODE[2]	P1
1	0	X	P
1	1	0	C
1	1	1	PCIN
0	X	X	0

Table 24: Selecting P0 with FPOPMODE[1:0]

FPOPMODE[1]	FPOPMODE[0]	P0
0	0	0
0	1	M
1	0	PCIN
1	1	D

Adder Input Negation

When FPOPMODE[5] (FPOPMODE[6]) is set to 1, it negates P0 (P1). Thus all four combinations of input negation are possible. See the following table.

Table 25: Adder Input Negation with FPOPMODE[6:5]

FPOPMODE[6]	FPOPMODE[5]	P
0	0	$P0 + P1$
0	1	$P1 - P0$
1	0	$P0 - P1$
1	1	$-P0 - P1$

Subnormals

The IEEE term for *denormal* is *subnormal*. The following explains how DSP58 behaves when it detects a subnormal input and when it creates a subnormal output.


- When the DSP58 detects a subnormal operand, it treats the operand as zero with the sign of the original operand.

- When the DSP58 creates a subnormal output, it flushes the output to zero and the sign is preserved as a result of the mantissa of the adder. For the multiplier, the sign is the XOR of the signs of both inputs.

DSP58 Design Considerations

Design for Performance

To achieve maximum performance when using DSP58, the design needs to be fully pipelined. For multiplier-based designs, DSP58 requires a three-stage pipeline. For non-multiplier-based designs, a two-stage pipeline must be used.

 **IMPORTANT!** *If latency is important in the design and only one or two registers can be used within DSP58, always use the M register.*

Design for Power

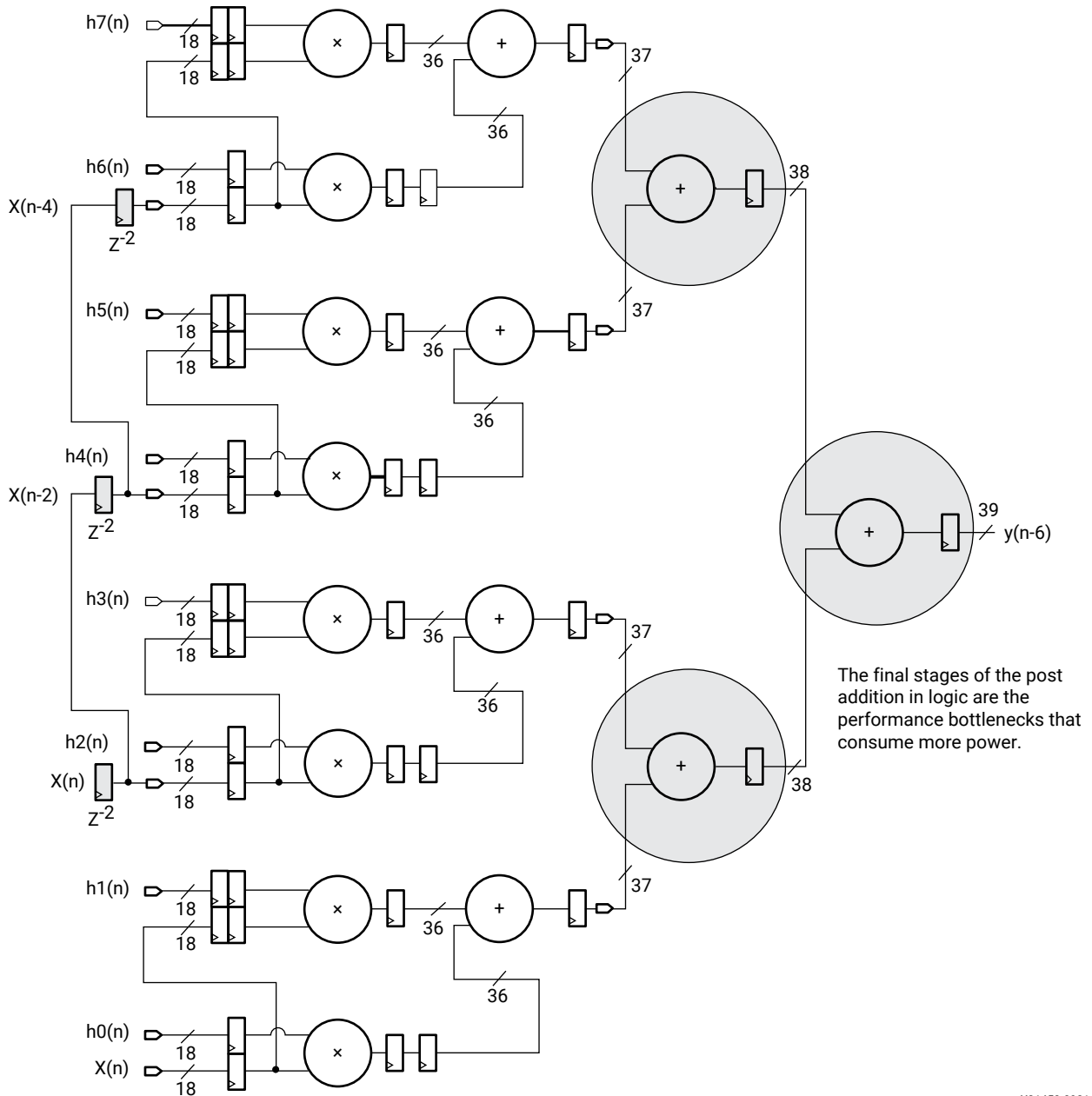
The USE_MULT attribute selects the multiplier to be used. This attribute can be set to NONE to save power when using only the Adder or Logic unit. Functions implemented in DSP58 use less power than those implemented in the programmable logic (PL). Using the cascade paths within DSP58 instead of PL routing is another way to reduce power. A multiplier with the M register in use, uses less power than one where the M register is not used. For operands less than 27×24 , PL power can be reduced by placing operands into the MSBs and zero padding unused LSBs. If one of the multiplier input operands is a constant, then assign this to the B input to reduce Booth Encoding logic power dissipation.

Adder Tree versus Adder Cascade

Adder Tree

In typical direct form FIR filters, an input stream of samples is presented to one input of the multipliers in the DSP58s. The coefficients supply the other input to the multipliers. An adder tree is used to combine the outputs from many multipliers as shown in the following figure.

Figure 34: Traditional FIR Filter Adder Tree

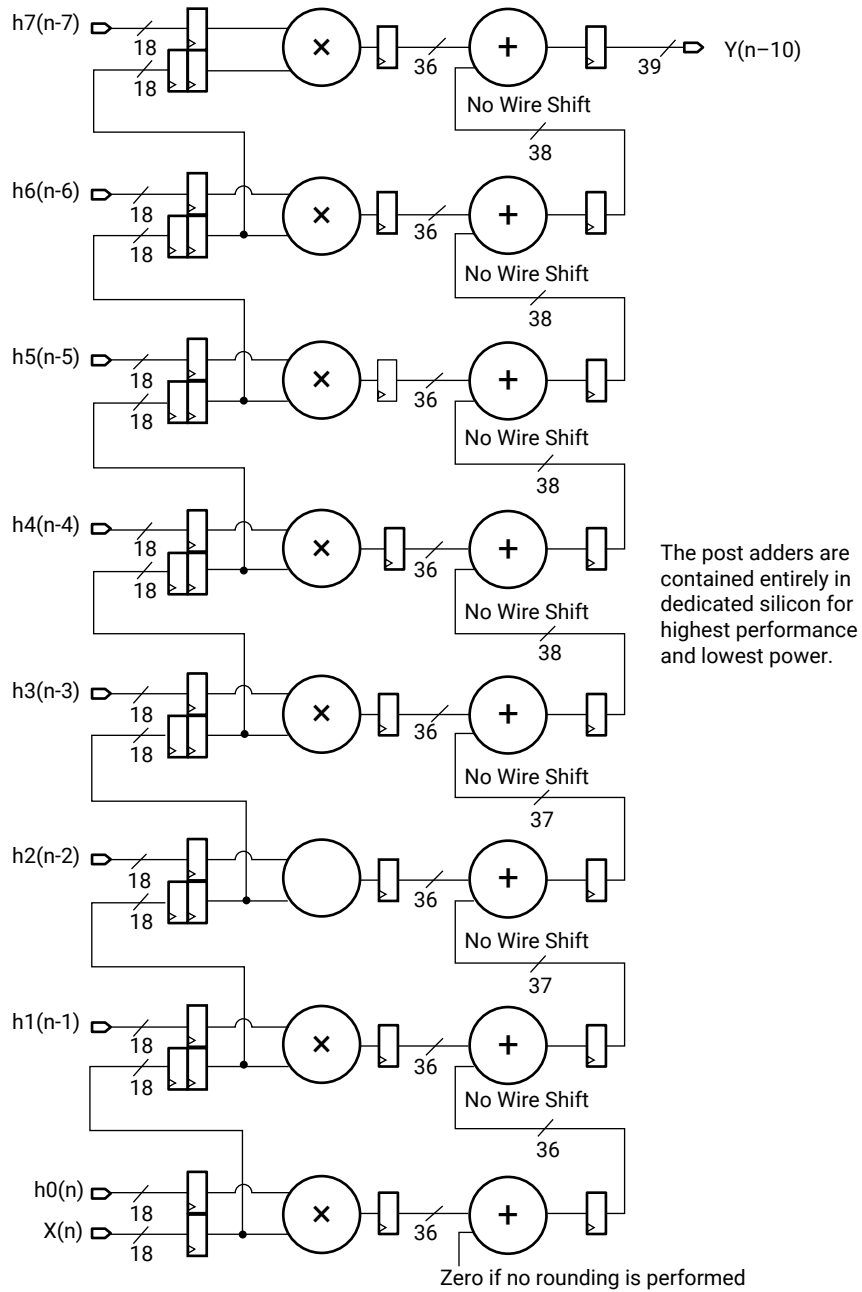


In the traditional approach, the adders in the programmable logic (PL) are usually the performance bottleneck. The number of adders needed and the associated routing depends on the size of the filter. The depth of the adder tree scales as the \log_2 of the number of taps in the filter. Using the adder tree structure shown in the figure above could also increase the cost, logic resources, and power.

Adder Cascade

The adder cascade implementation accomplishes the post addition process with minimal silicon resources by using the cascade path within the DSP58. This involves computing the additive result incrementally, using a cascaded approach as illustrated in the following figure.

Figure 35: Adder Cascade




The post adders are contained entirely in dedicated silicon for highest performance and lowest power.

X21462-101018

It is important to balance the delay of the input sample and the coefficients in the cascaded adder to achieve correct results. The coefficients are staggered in time.

Connecting DSP58s Across Columns

Using the cascade paths to implement adders significantly improves power consumption and speed. The maximum number of cascades in a path is limited only by the total number of DSP58s in one column on the chip. For more information, see [Device Resources](#).

 **IMPORTANT!** *The height of the DSP column can differ between devices and must be considered when porting designs.*

Spanning columns is possible by taking the bus output from the top of one DSP column and adding CLB slice pipeline registers to route this bus to the C input of the bottom DSP58 of the adjacent DSP column. Alignment of input operands is also necessary to span multiple DSP columns.

Time Multiplexing the DSP58

The high-speed math elements in DSP58 enable you to use time multiplexing in DSP designs. Time multiplexing is the process of implementing more than one function within a single DSP58 at different instances of time. Time multiplexing can be done easily for designs with low sample rates. The calculation to determine the number of functions (N) that can be implemented in one single DSP58 is shown in the following equation.

$$N \times \text{Channel Frequency} \leq \text{Maximum Frequency of DSP58}$$

Implementing a time-multiplexed design using DSP58 results in reduced resource usage and reduced power.

DSP58 contains the basic elements of classic FIR filters, a multiplier followed by an adder, delay, or pipeline registers, and the ability to cascade an input stream (B bus) and an output stream (P bus) without exiting to a general CLB slice.

Multichannel filtering can be viewed as time-multiplexed, single-channel filters. In a typical multichannel filtering scenario, multiple input channels are filtered using a separate digital filter for each channel. Due to the high performance of DSP58, a single digital filter can be used to filter multiple input channels. As an example, eight input channels can be handled by clocking the single filter with an 8x clock. This implementation uses 1/8th of the total resource as compared to implementing each channel separately.

Notes and Suggestions

- Implement small multiplies (for example, 4×4 multiplies) and small bit-width adders and counters using the CLB logic LUTs and carry chain. If the design has a large number of small add operations and/or counters, take advantage of the SIMD mode and implement the operation in DSP58. Factor of 2x area and power savings occur, when compared to using interconnect logic, whenever input registers are also folded into DSP58 for SIMD mode functions.
- Always sign extend the input operands when implementing smaller bit width functions. For lower power in the programmable logic (PL), push operands into MSBs and ground (GND) LSBs.
- While cascading different DSP58s, match the pipestages of the different signal paths.
- Implement a count-up-by-one counter within the DSP58 using the CARRYIN input. A count-by-N or variable-bit counter can use the C or A:B inputs.
- DSP58 counters can be used to implement control logic that runs at maximum speed.
- Use SRL16s/SRL32s in the CLB and block RAM to store filter coefficients or act as a register file or memory elements in conjunction with DSP58. The bit pitch of the input bits is designed to pitch match the CLB and block RAM.
- The block RAM can also be used as a fast, finite state machine to drive the control logic for the DSP design.
- DSP58 can also be used with a processor, for example, MicroBlaze™ or PicoBlaze™ processors, for hardware acceleration of processor functions.
- Use a pipeline register at the output of an SRL16 or block RAM before connecting it to the input of DSP58. This ensures the best performance of input operands feeding DSP58.
- The register at the output of the SRL16 in DSP58 has a reset pin and a clock-enable pin. To reset the SRL16, a zero is input into the SRL16 for 16 clock cycles while holding the reset of the output register High. This capability is particularly useful in implementing filters where the SRL16s are used to store the data inputs.

Pre-Adder Block Applications

DSP58 users can benefit from the pre-adder in several applications ranging from wireless applications (for example, in algorithms as in the Long-Term Evolution specification), in generic filtering (FIR and IIR), in video processing (for example, alpha blending), and many others. The most common use for the pre-adder is to pre-add corresponding values of a symmetric FIR filter tap delay line.

Refer to [Pre-Adder Used as a Multiplexer](#) for information on how the pre-adder can be used as a 2:1 multiplexer to dynamically select between A[26:0] and D[26:0].

Memory-Mapped I/O Register Application

To use DSP58s as memory-mapped I/O registers, you must broadcast the write data bus feeding to all the DSP58s to be used in this manner. To have random read access, a wide multiplexer is needed. Additional DSP58s can be configured as a wide bus multiplexer to help reduce routing congestion. An address decoder must be implemented in programmable logic to control individual PREG CEs to load the appropriate DSP58 output register from the write data bus.

Rounding

Arithmetic rounding is a process where a result is quantized in an *intelligent* manner. Given a choice, one would like to use an implementation that minimizes the loss of precision. However, in most cases of hardware implementation, including ones with Xilinx DSPs, one has to be aware of the overheads associated with the various rounding techniques to make appropriate design trade-offs. While the binary point placement and bit position where rounding occurs are independent of each other, it is assumed that the designer's goal is to round off the fractional bits to an integer value.

One form of rounding is simple truncation or dropping undesired LSBs from a large result to obtain a reduced number of result bits. The problem with truncation happens after the bits are dropped and the new reduced result has an undesirable DC data shift toward a more negative number. For example, if a number has the decimal value 2.8 and the fractional part of the number is truncated, then the result is two. In this example, the original number is closer to 3 than to 2 and a rounded result of 3 is more desirable than the simple truncated result of 2.

In the next few sections, other methods of quantization with a more desirable effect, including symmetric rounding and convergent rounding are discussed.

MACC

Word length of the result after the MACC operation is usually much larger than the word length of the inputs. Rounding can be applied in DSP58 before truncation. As shown in the next section, the sign of the result is needed to obtain the correct rounding. In MACC, however, it can be difficult to determine the sign of the output ahead of time, thus the rounding can cost an extra cycle.

In symmetric rounding, the extra cycle can be avoided by adding the C input on the very first cycle using dynamic OPMODE. In this case, the sign bit of the last but one cycle of the accumulator can be used for the final rounding operation done in the final accumulate cycle. There is a rare chance that the final accumulate operation can flip the sign of the output from the previously accumulated value, leading to a different result from the expected one. In the case of convergent rounding, the patterndetect result can be used. Another solution to avoid the extra cycle is to use an extra DSP58.

Symmetric

Symmetric rounding is a method of quantization that accomplishes the more desirable effect of quantizing numbers to keep them from becoming biased in the wrong direction. For example, in symmetric rounding towards infinity, the midpoint number 2.5 rounds to 3.0 and -2.5 rounds to -3 .

The C port in DSP58 is used to mark the location of the decimal point. The number of continuous ones in the C port bus plus 1 indicates the number of decimal places in the original number; for example, in the case of 4 decimal places, C is 00 ... 0111. C can be used for dynamic or static rounding whereas RND is used for static rounding only. The sign bit determines the symmetric rounding towards infinity or zero. For rounding toward infinity, the midpoint negative and positive numbers are both rounded away from zero. For example, 2.5 rounds to 3 and -2.5 rounds to -3 . Note that CARRYINSEL can select internal signals to implement the complemented sign bit and to avoid using CARRYIN and programmable logic. In case of rounding toward zero, positive and negative numbers at the midpoint are rounded towards zero. For example, 2.5 rounds to 2 and -2.5 rounds to -2 . The following tables show examples of symmetric rounding.

Table 26: Round to Zero (Decimal Place = 4)

Multiplier Output	C	Sign Bit	Output = Multiplier Out + C + Sign Bit
0010.1000 (2.5)	0000.0111	0	0010.1111 (2 after truncation)
1101.1000 (-2.5)	0000.0111	1	1110.0000 (-2 after truncation)
0011.1000 (3.5)	0000.0111	0	0011.1111 (3 after truncation)

Table 27: Round to Infinity (Decimal Place = 4)

Multiplier Output	C	Sign Bit Complement	Output = Multiplier Out + C + Sign Bit Complement
0010.1000 (2.5)	0000.0111	1	0011.1111 (3 after truncation)
1101.1000 (-2.5)	0000.0111	0	1101.1111 (-3 after truncation)
0011.1000 (3.5)	0000.0111	1	0100.0000 (4 after truncation)

Convergent

In convergent rounding, the final result is rounded to the nearest even number (or odd number). In conventional implementations, if the midpoint is detected, then the units-placed bit before the round needs to be examined to determine whether the number is going to be rounded up or down.

In convergent rounding towards even, the final result is rounded toward the closest even number, for example:

2.5 rounds to 2 and -2.5 rounds to -2, but 1.5 rounds to 2 and -1.5 rounds to -2.

In convergent rounding towards odd, the final result is rounded toward the closest odd number, for example:

2.5 rounds to 3 and -2.5 rounds to -3, but 1.5 rounds to 1 and -1.5 rounds to -1.

The convergent rounding techniques require the use of programmable logic (PL) in addition to the DSP58. The different methods of implementing a convergent rounding scheme will be covered in a future revision of this document.

Overflow/Underflow/Saturation

The pattern detector allows DSP58 to support convergent rounding and counter auto reset when a count value has been reached. It also supports overflow, underflow, and saturation in accumulators. The following discussion of overflow and underflow applies to sequential accumulators (MACC or adder-accumulator) implemented in a single DSP58. The accumulator must have at least one guard bit.

The dedicated overflow and underflow outputs of DSP58 use the pattern detector to determine if the operation in DSP58 has overflowed or underflowed beyond the P[N] bit (N = 0 to 56). If the pattern detector is set to detect a 58-bit pattern 00000 ...0, with a 58-bit mask of 00111111 ...1 (default settings), the DSP58 overflows beyond 00111 ...1 or underflows beyond 11000...0. In other words, DSP58 detects overflow past the 57th bit P[56]. The USE_PATTERN_DETECT attribute is set to PATDET to enable the use of pattern logic. This overflow/underflow implementation uses a redundant sign bit and reduces the output bit width to 57 bits.

The overflow and underflow flags remain High for only one cycle. These values must be registered in the programmable logic if a saturation value is used in the case of an overflow or underflow. The registered flags are used as multiplexer select signals. The inputs of the multiplexer are tied to the maximum positive value (0011...1) or the maximum negative value (1100...0). Depending on whether an overflow or an underflow occurred, the appropriate input is selected at the output.

By setting the 58-bit mask to other values, for example, 00001111 ...1, the bit value P(N) at which overflow is detected can be changed. This logic supports overflow/underflow detection respectively for a positive number of $2^N - 1$ and a negative number of 2^N in two's complement, where N is the number of 1s (0 to 56) in the mask field.

DSP58 Applications

Introduction

DSP58 and the associated new functional modes (for example, CINT18) can be used efficiently in video, wireless, and networking applications. High performance, dedicated logic to optimize specific functions combined with low power dissipation makes DSP58 an ideal choice for the above application segments. This chapter discusses the implementation details of the functions associated with various applications. These functions can be the building blocks for a variety of complex systems.

This chapter contains the following sections:

- New Functional Mode Applications
- Logic and Basic Math Applications
- Advanced Math Applications
- Filter Designs

New Functional Mode Applications

New functional modes are implemented in DSP58, including complex multiplier, floating point, and vector dot product. The basic operation of these modes are described in earlier chapters of this manual. This section shows coding examples of complex multiplier and dot product. A reference design for a floating point implementation is provided in [Floating Point Time-Interleaved Dot-Product Engine](#).

18 × 18 Complex Multiply using DSPCPLX

A complex multiply function is:

$$(A_{re} \pm jA_{im}) \times (B_{re} \pm jB_{im}) = Out_{re} + jOut_{im}$$

Of the four possible cases, for one $(A_{re} + jA_{im}) \times (B_{re} + jB_{im})$, the output is given as:

$$Out_re = (A_re \times B_re) - (A_im \times B_im)$$

$$Out_im = (A_re \times B_im) + (A_im \times B_re)$$

In the legacy mode (UltraScale™ architecture), the implementation uses three DSP48E2s with pre-adders. For more information, refer to the section Complex Multiplier Examples in the *Vivado Design Suite User Guide: Synthesis* (UG901).

The above solution can handle a width of upto 26×23 in the Versal™ architecture versus 26×17 in the UltraScale architecture.

A new mode is available in the Versal architecture where the 18×18 complex multiply can be implemented using one DSPCPLX module (two back-to-back DSP58s). On a per-cycle basis, a back-to-back DSP58 pair can form complex conjugates of \tilde{A} and \tilde{B} for multiplication. See the following table for the possible combinations of CONJUGATE_A and CONJUGATE_B.

Table 28: Complex Conjugates for Multiplication

CONJUGATE_A	CONJUGATE_B	Product	A	B
0	0	$\tilde{A} \times \tilde{B}$	$A_re + jA_im$	$B_re + jB_im$
1	0	$\tilde{A}^* \times \tilde{B}$	$A_re - jA_im$	$B_re + jB_im$
0	1	$\tilde{A} \times \tilde{B}^*$	$A_re + jA_im$	$B_re - jB_im$
1	1	$\tilde{A}^* \times \tilde{B}^*$	$A_re - jA_im$	$B_re - jB_im$

INT8 Integer Multiplication

As noted in [Dot Product Unit](#), u and v are the only intermediate results that go through the X and Y multiplexers to the ALU to generate output at P. The DSP58 is designed to do one multiplication and addition operation with up to 27×24 bit multiplication and up to 58-bit accumulation. The int9 \times int8 operation in DSP58 is a new feature that allows six independent inputs to generate three partial products. In addition, port A can hold up to three unsigned 8-bit values by setting the sign bits to 0 to indicate non-negative values. The sign bits are A[8], A[17], and A[26].

Coding examples will be available in Language Templates from the Vivado® Integrated Design Environment (IDE) in a subsequent release.

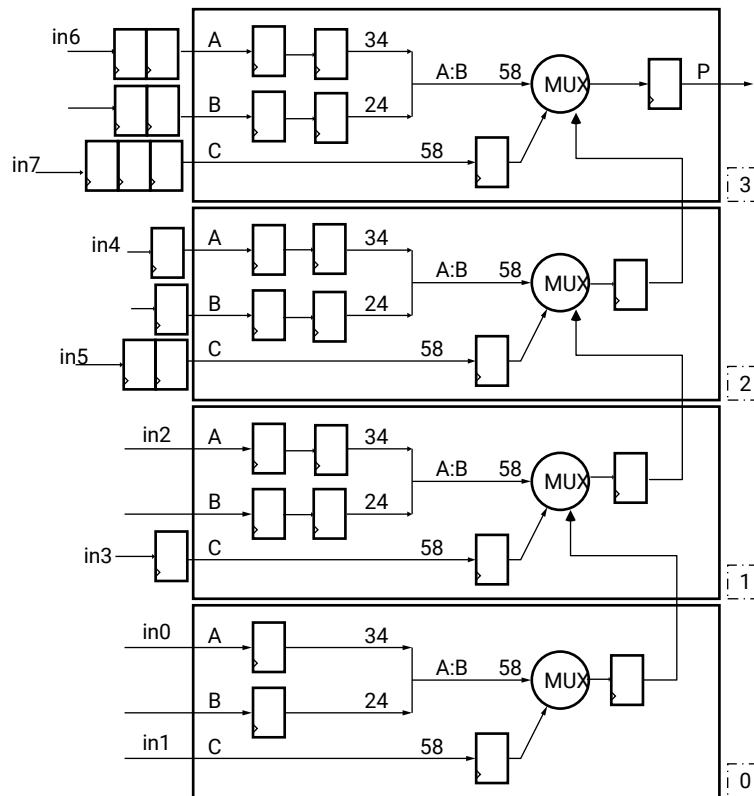
Logic and Basic Math Application

Bus Multiplexer

Wide bus multiplexers are used in many network switching applications where voice and data need to be multiplexed. In digital signal processing, a multiplexer is used to take several separate data streams and combine them into one single data stream at a higher rate. The DSP58 can be used to implement high-width (up to 58 bits) multiplexers for networking and video applications.

The OPMODE bits are used to choose between the C input and the A:B input within DSP58. Each DSP58 can multiplex between two 58-bit values. The following figure shows the implementation of a 58-bit-wide 8:1 multiplexer. Additional pipeline registers will be used to increase performance of the multiplexer.

Figure 36: 58-Bit-Wide 8:1 Multiplexer Using DSP58



X22160-100820

The ALUMODE is set to 0000. Different OPMODE settings can be used for each of the four DSP58s. The following table lists one way of setting the OPMODEs to implement the 58-bit multiplexer. To drive the OPMODE setting to each DSP, PL logic is used. To optimize the speed of this multiplexer design, pipeline registers are added. The latency of the design is equal to number of (DSPs+1), which in this case is 5 clock cycles.

Table 29: OPMODE Settings for an 8:1 Multiplexer

OPMODE				Selected Input
DSP58_3	DSP58_2	DSP58_1	DSP58_0	
000010000	000010000	000010000	000000011	in 0
000010000	000010000	000010000	000001100	in 1
000010000	000010000	000000011	000000000	in 2
000010000	000010000	000001100	000000000	in 3
000010000	000000011	000000000	000000000	in 4
000010000	000001100	000000000	000000000	in 5
000000011	000000000	000000000	000000000	in 6
000001100	000000000	000000000	000000000	in 7

Division

Binary division can be implemented in DSP58 by performing a shift and subtract or a multiply and subtract. DSP58 includes a shifter, a multiplier, and adder/subtractor unit to implement binary division. The division by subtraction and division by multiplication algorithms are shown in the following sections. The algorithms assume:

1. $N > D$
2. N and D are both positive

If either N or D is negative, use the same algorithm by taking the absolute positive values for N and D and making the appropriate sign change in the result. The terms N and D in the algorithm refer to the number to be divided (N) and the divisor (D). The terms Q and R in the algorithm refer to the quotient and the remainder, respectively.

Dividing with Subtraction

If N is an 8-bit integer and D is not more than 8-bits wide, $N/D = Q + R$.

1. Assign the value 00000000 to the 8-bit register R .
2. Shift the R register one bit to the left and fill in the LSB with $N[8-n]$.
3. Calculate $R - D$.
4. Set R and set Q .
 - If $R - D$ is positive, set $Q[8-n]$ to 1 and $R = R - D$.
 - If $R - D$ is negative, set $Q[8-n]$ to 0 and $R = R$.
5. Repeat steps 2 to 4, filling in $R[0]$ each time with $N[8-n]$, where n is the number of the iteration. $Q[8-n]$ is filled each time in Step 4. The range of n is 1 to 8.

After the eighth iteration, $Q[7:0]$ contains the quotient, and $R[7:0]$ contains the remainder.

Dividing with Multiplication

The multiply and subtract method consists of rewriting $N/D = Q$ as $N = D * (Q + R)$. The answer is calculated using the following steps for an 8-bit N/D .

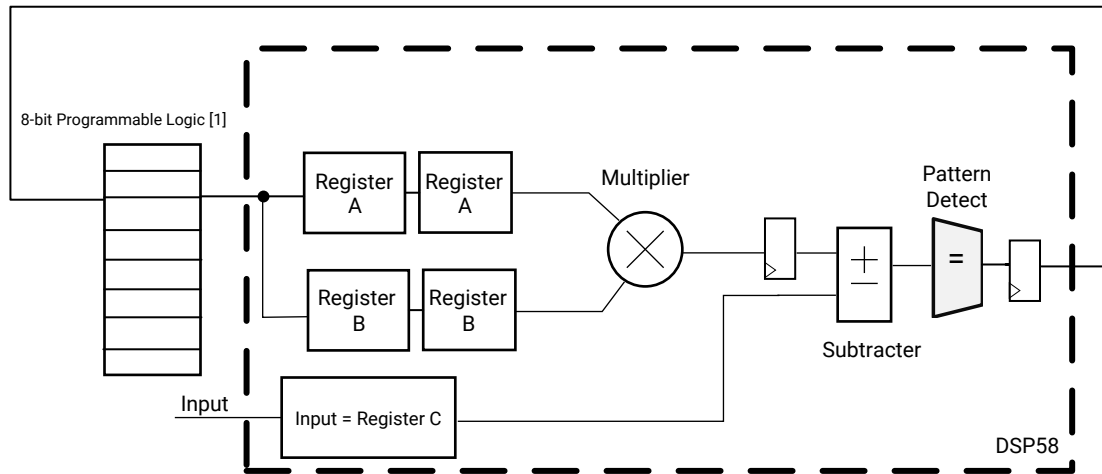
1. Set the initial value of $Q[8-n] = 1$ and the bits right of $Q[8-n]$ to 0.
2. Calculate $D*Q$.
3. Calculate $N - (D*Q)$.
 - If step 2 is positive, $N > (D*Q)$, set $Q[8-n]$ to a 1.
 - If step 2 is negative, $N < (D*Q)$, set $Q[8-n]$ to a 0.
4. Repeat steps 1 to 3.

After the eighth iteration, $Q[7:0]$ contains the quotient and $N - (D*Q)$ contains the remainder. To map to DSP58, N is applied to the C input, D is applied to the B input and Q (the whole bus) is applied to A. The initial value $Q[8-n]$ is set at the A input and after the eighth iteration, the output register P contains the remainder. Both of these division implementations are possible in one DSP58 and the latency is eight clock cycles for the fully combinational case. The latency increases if registers are used in the DSP.

Square Root

The square root of an integer number can be calculated by successive multiplication and subtraction. It is similar to the subtraction method used to divide two numbers. The square root of an N -bit number will have $N/2$ (rounded-up) bits. If the square root is a fractional number, $N/2$ clocks are needed for the integer part of the result, and every following clock gives one bit of the fractional part. The logic needed to compute the square root is illustrated in the following figure. The calculation explained here is based on the assumption that there is one stage pipelining at the input of the multiplier.

Figure 37: Square Root Logic



Note: [1] The 8 bits include 4 integer bits and 4 fractional bits. The programmable logic contains LUT and flip-flops. The flip-flops are referred to as "PL_FF" in the document.

X22157-110920

The square root can be calculated as follows:

$$\sqrt{X} = Y.Z$$

Y is the integer part of the root and Z is the fraction part. Registers A and B refer to the registers found on the A and B inputs to DSP58 respectively and Register C refers to the registers found on the C input to DSP58. The steps to calculate are listed as follows.

1. Read the number into Register C. Set the register in external programmable logic (referred to as PL_FF) to 10000000.
2. Calculate $Register\ C - (PL_FF * PL_FF)$. C is a 16-bit value in the form 0000000C00000000.
3.
 - If step 2 is positive, set $PL_FF[(8-clock)] = 1, PL_FF[(8-clock) - 1] = 1$
 - If step 2 is negative, set $PL_FF[(8-clock)] = 0, PL_FF[(8-clock) - 1] = 1$
4. Repeat steps 2 and 3 until the required precision for the fractional part is reached.

In the case where there is only 1 stage pipelining to the input of the multiplier, four clock cycles are required to calculate the integer part of the value Y. The number of clock cycles required for the fraction part, Z, depends on the precision required. For an 8-bit value that has 4 bits for the integer part and 4 bits for the fractional part, the value in PL_FF after eight clock cycles includes the integer part given by the four MSBs and the fractional part given by the four LSBs. In the use case design, four additional pipeline stages are added for every 1-bit value to improve timing.

Clock Domain Crossing and Time Division Multiplexing

For complex datapaths, multipumping is a method whereby a resource is clocked at a frequency that is a multiple of the surrounding circuit. This allows the receiver, in this case the DSP, to be shared among multiple uses in the same cycle. This concept maps to DSP engines which are capable of running at higher frequencies than designs implemented in programmable logic. Theoretically, the DSP can be fed by N memory instances in programmable logic. Assuming the DSP runs at f_{ck} , the memories can run at slower speed of f_{ck}/N . In this use case, $N=2$. The design has changed since the last release and is no longer using the handshaking mechanism. The fabric is connected directly to the DSP. The clocks connected to the two units are considered synchronous with a defined phase relationship.

Advanced Math Applications

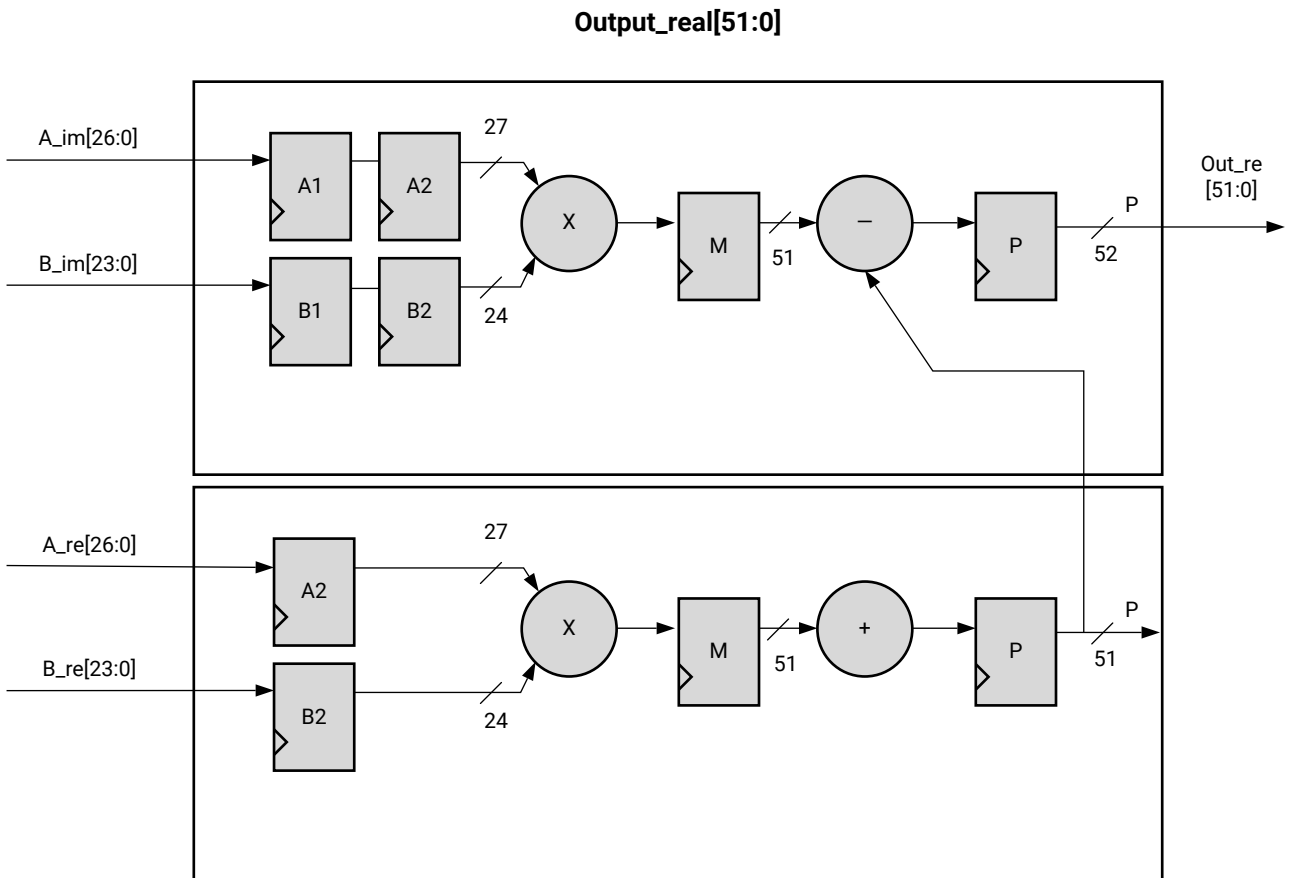
27 × 24 Complex Multiply

The three DSP58s version in the legacy mode can handle up to 26 × 23 complex multiply. For full bit width 27 × 24, four DSP58s are required. Two DSP58s implement the real part and the other two implement the imaginary part. Up until 26 × 23, the three DSP58s version is preferable because it uses one DSP less. However, note that the performance could lower while the power increases because the design has to pass through programmable interconnect. The implementation with four DSPs uses the dedicated cascade path in the DSP logic and is the better choice for performance and power dissipation. See the following block diagrams for the implementation of real and imaginary parts.

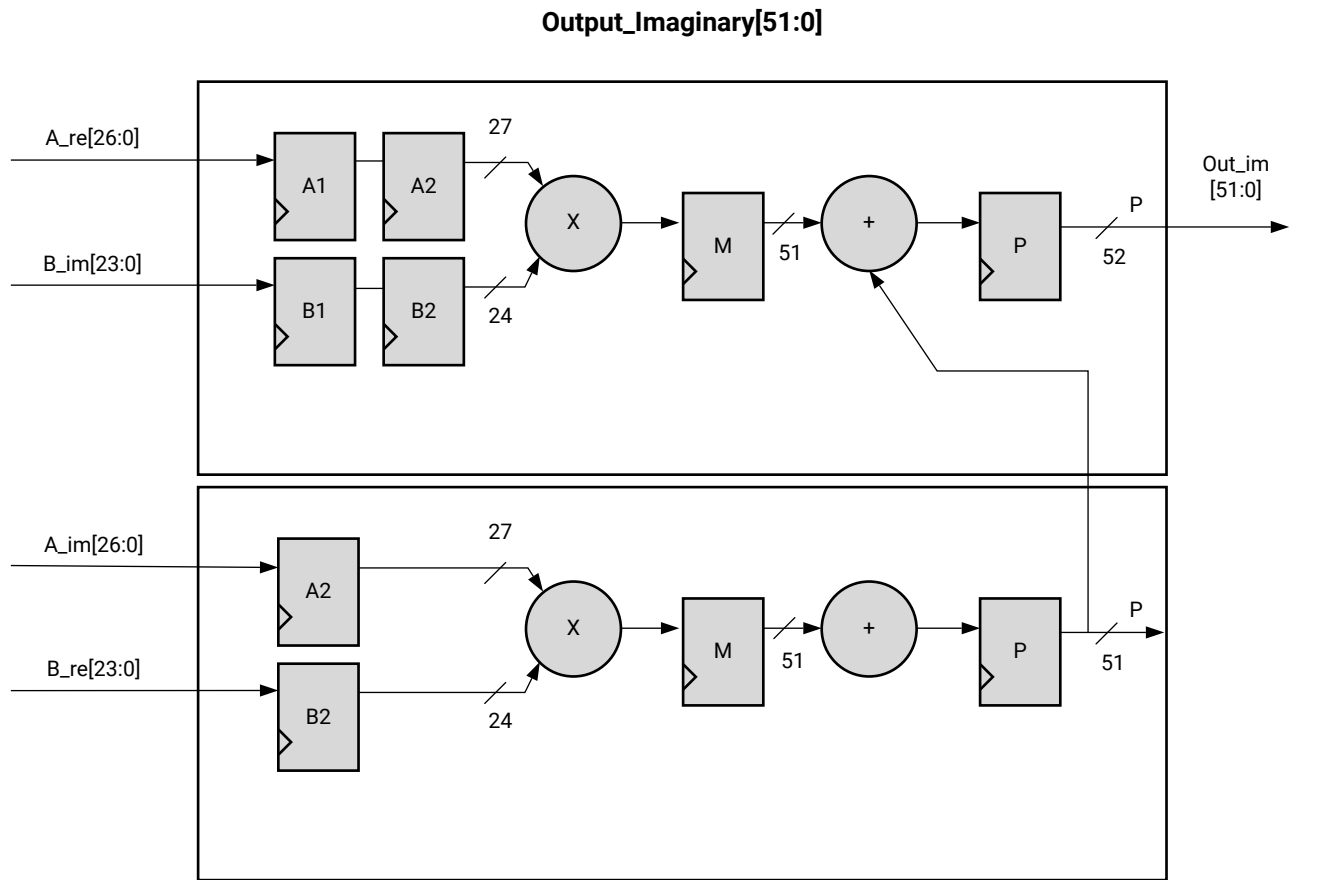
The implementation refers to the following case.

$$(a_{re} + ja_{im}) \times (b_{re} + jb_{im})$$

Figure 38: Real Part of a 27 × 24 Bits Complex Multiplier



X21470-092118

Figure 39: Imaginary Part of a 27×24 Bits Complex Multiplier


Complex Multiply with MACC and MADD Operations

MACC operation includes a 3-input adder (two from the multiplier partial product outputs) in the accumulator stage and requires guard bits to prevent overflow. For more information, refer to [MACC](#).

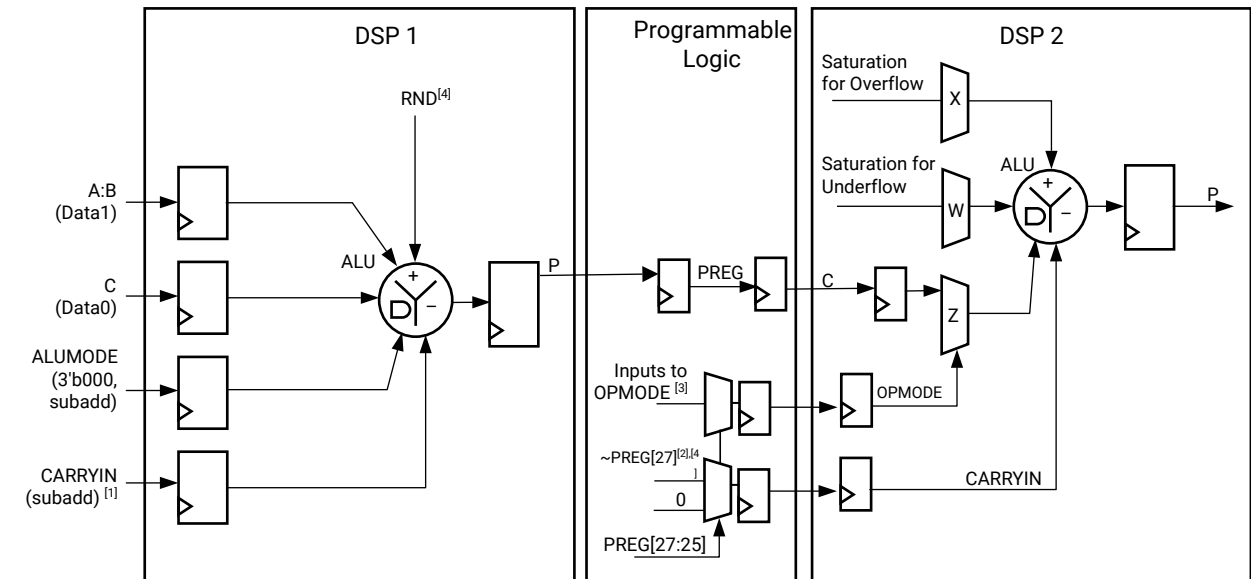
DSPCPLX

Coding examples for the implementation of the complex multiplication with attributes values corresponding to the table in [DSPCPLX Pipeline Configuration](#) are available as Language Templates with the Vivado® Integrated Design Environment (IDE) 2020.2 version.

Rounding and Saturation for Adder

A saturate (SAT) and round (RND) function for addition/subtraction operation can be provided by adding a second DSP58 and by using some P output bits from the previous DSP58 to detect the SAT condition. The following figure shows a simplified block diagram of the implementation.

Figure 40: Rounding and Saturation for Adder



[1] subadd is the input port that defines whether it is an addition or subtraction
 Subadd = 1'b0 -> Data1 + Data0
 Subadd = 1'b1 -> Data1 - Data0
 [2] MUX selection for CARRYIN to DSP2
 PREG[27:25]=000 or 111 -> CARRYIN=~PREG[27]
 Otherwise -> CARRYIN = 0

[3] MUX selection (PREG[27:25]) for OPMODE in DSP2
 110 -> OPMODE = 10_000_00_00
 001 -> OPMODE = 00_000_00_11
 010 -> OPMODE = 00_000_00_11
 Otherwise -> 00_011_00_00
 [4] Refer to Rounding Applications in UG193 for the correct rounding value.

X22274-100820

Programmable logic is used to implement multiplexers controlled by the P output bits of the first DSP to generate OPMODE and CARRYIN for the second DSP58. The operands for the addition/subtraction are fed to the first DSP58 through the concatenation of A:B and C. The rounding value is applied to the WMUX (static rounding to infinity is implemented). In the use case design, there is a dedicated input pin (subadd) that decides whether the inputs are added or subtracted. Considering that the output is represented as 26 bits (8 bits of the integer part and 18 bits of the fractional part), the saturation conditions (positive or negative magnitude) are detected by checking bits 27, 26, and 25 of P output from the first DSP. Bit 25 is the sign bit. Bits 26 and 27 are guard bits. The second guard bit is needed because in the worst case extra overflow due to the RND addition can occur. In the case of saturation, a unique value at the input of the X and W MUXes is selected respectively as ALU output. The output (8 bits of integer part) will be in a decimal representation of +127 for the positive range and -128 for the negative range.

Operations on W, X, Y, Z, and CIN are controlled by setting ALUMODE as follows:

$$ALUMODE = 0000 \text{ ALU output} = Z + (W + X + Y + CIN)$$

$$ALUMODE = 0001 \text{ ALU output} = (W + X + Y + CIN) - Z - 1$$

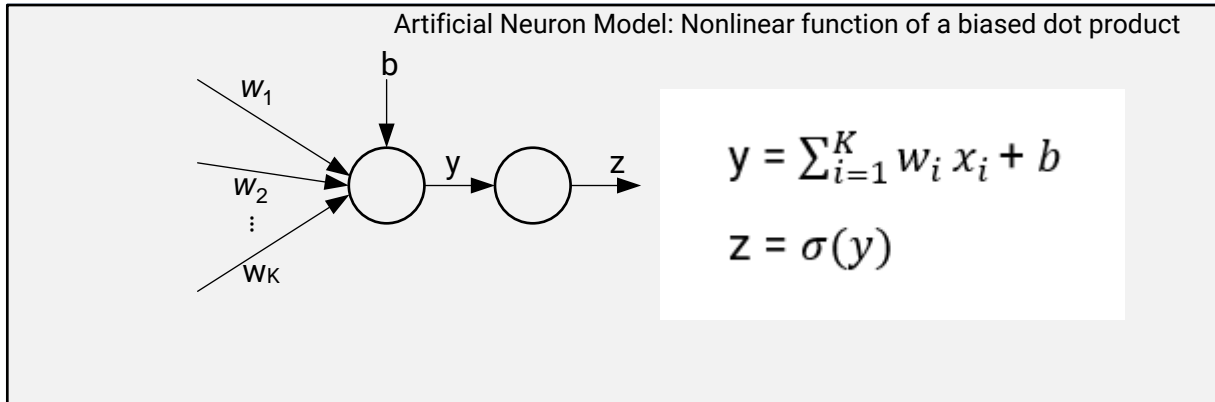
For more information, refer to the [ALUMODE Inputs](#) section.

To offset the -1 for the case ALUMODE = 0001, the input pin which selects between addition and subtraction is connected to CARRYIN port of the first DSP.

Floating Point Time-Interleaved Dot-Product Engine

A neuron model can be represented as a nonlinear function of a biased dot product of a weight vector. A simple representation is shown in the following figure.

Figure 41: Artificial Neuron Broadcast Model



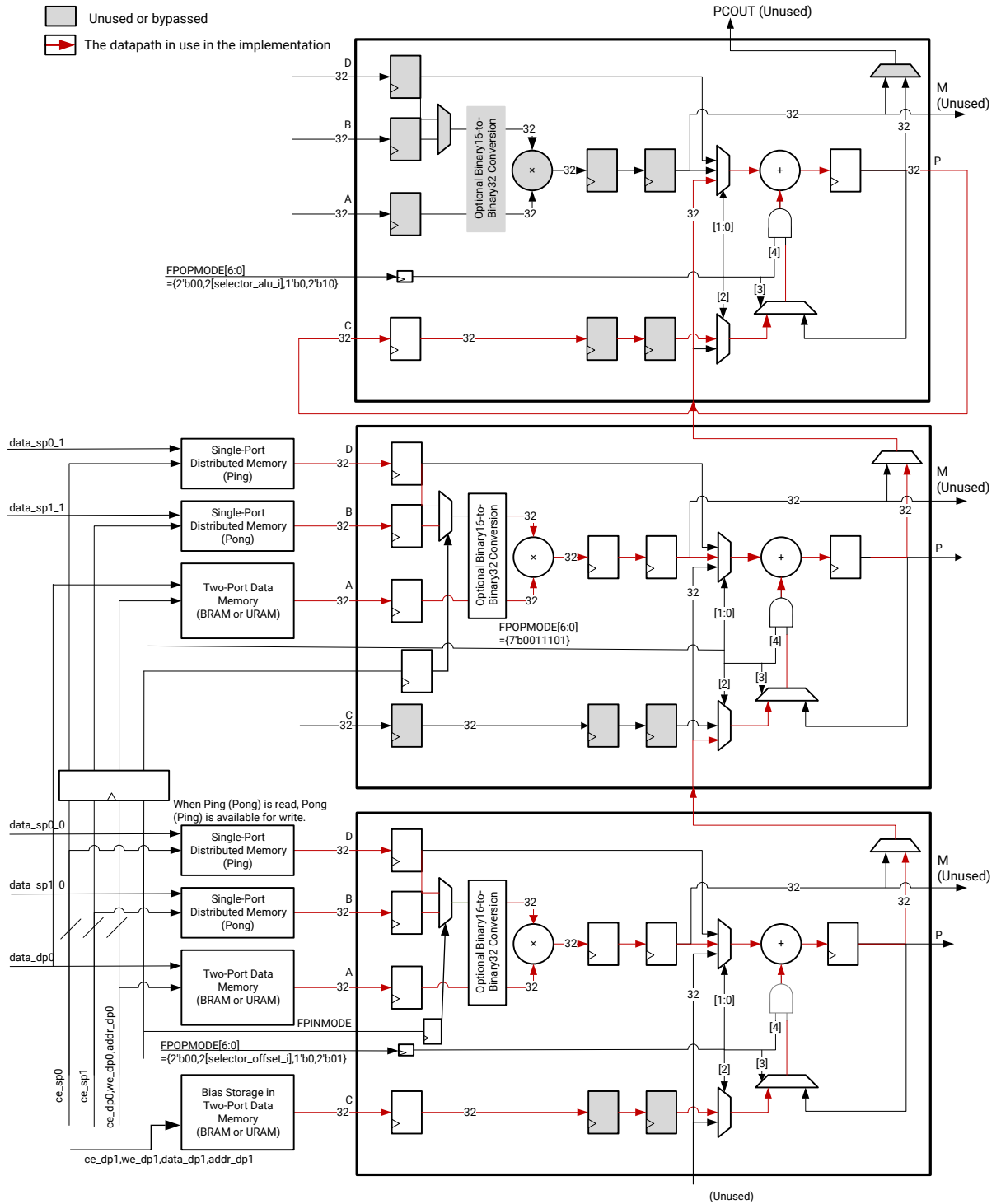
X22451-102119

Multiple artificial neurons are modeled as a matrix-vector multiplication. The weights are in an $M \times K$ matrix W , the biases are in an M -dimensional vector b , and the input data (activations) are in a K -dimensional vector x . The pre-activations are therefore in an M dimensional vector represented as follows.

$$y = Wx + b$$

The following figure implements the biased vector dot products with a cascade of DSPs. For $N=64$, in the bottom two DSPs, each computes a dot product for two 32-D input vectors and the bottom DSP also adds the bias term. The top DSP accumulates the results using a two threads accumulator loop between P and C.

Figure 42: Floating-Point Time-Interleaved Dot-Product Engine



Note the following in the use case design.

- The input and bias vectors are implemented as dual-port memories (Block RAM) where as the weight vectors are single port memories implemented in programmable logic using ping-pong scheme.
- The block diagram shows one column of cascading DSPs. In the design, there are two columns that share the input vectors in the same cycle in both the middle and bottom rows of DSPs.
- There is a separate memory for bias in each column. The memories share the same controls (CE, WE, and Addr) but separate data inputs.
- The control signals to data/weight memories and FPINMODE to the middle DSPs are delayed (registered) versions of the ones to the bottom DSPs except the input data which are dedicated to each memory.

Filter Designs

Introduction

DSP58 filter applications include, but are not limited to, the following:

- Wireless communications
- Image processing
- Video filtering
- Multimedia applications
- Portable electrocardiogram (ECG) displays
- Global Positioning Systems (GPS)

The main components used to implement a digital filter algorithm include adders, multipliers, storage, and delay elements. DSP58 includes all of the above elements, making it ideal to implement digital filter functions. For example, in the parallel FIR filter, all of the input samples from the set of n samples are present at the input of each DSP58. Each DSP58 multiplies the samples with the corresponding coefficients within that DSP58. The outputs of the multipliers are combined in the cascaded adders.

Benchmarks and Requirements

A wide variety of filter architectures are available to design engineers. The type of architecture chosen is typically determined by the amount of processing required in the available number of clock cycles. The two most important factors are:

- Sample rate (F_s)
- Number of coefficients (N)

On one end, there are sequential processing FIR filters, including the single-multiplier MACC FIR filter, that usually offer the best solution when the sample rate is low. The MACC structure uses a single multiplier with an accumulator to implement an FIR filter sequentially. At the other end, as the sample rate increases, the architecture selected for a desired FIR filter becomes a more parallel structure (that is, semi-parallel and parallel FIR filters) involving more multiply and add elements. Between a single multiplier MACC FIR filter and a fully parallel full FIR filter, the trade-off with the MACC FIR filter is that not only does it reduce hardware by a factor of N but it also reduces filter throughput by the same factor.

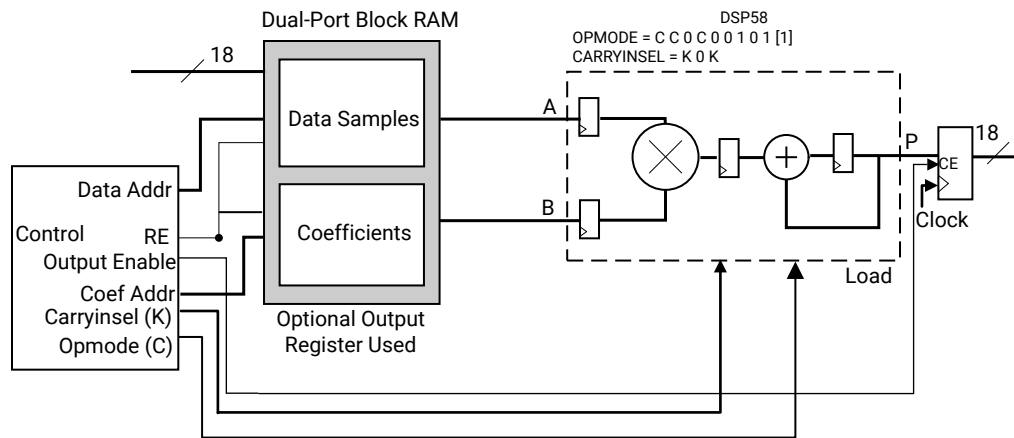
Resource Utilization Guideline

In the case of slow sample rate and small number of coefficients, the single MACC FIR filter is well suited. In the case of high sample rate and/or large number of coefficients, consider using a semi-parallel or parallel FIR filter. As for coefficients, if the number is large and/or the width is high, dual port block RAM is the preferred choice for the memory buffer. A high level implementation example of this design is provided in the following figure. If the number of coefficients and/or their width is small¹, distributed memory (LUTRAM) can be used as coefficient buffer instead of block RAM. If the data width is small¹, SRL16 can be used as data buffer instead of block RAM.

Note:

1. Based on the size, synthesis tools in Vivado Design Suite automatically maps to block RAM or SRL16/LUTRAM. To choose between SRL16/LUTRAM and block RAM, users must compare the timing and resource utilization in both cases to find the optimal solution.

Figure 43: Single-Multiplier MACC FIR Filter



Note: [1] To save inverter, set IS_OPMODE_INVERTED to 000100000.

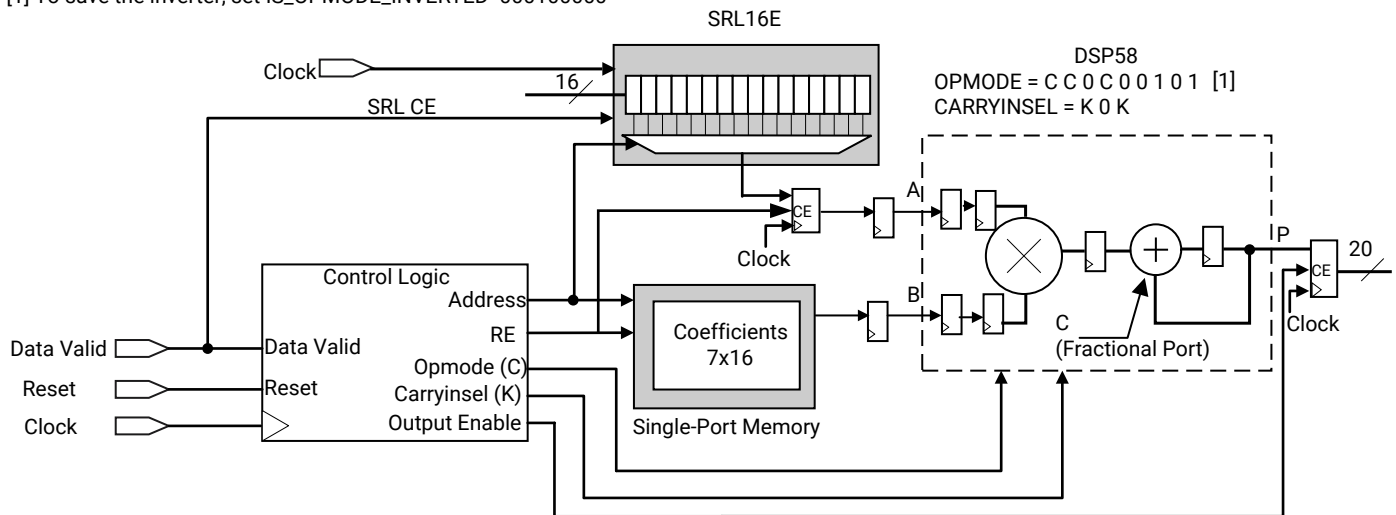
X21473-100820

For block RAM implementation of the data buffer, the cyclic RAM buffer is used. For small-sized FIR filters (typically those under 32 taps), block RAM can be underutilized as a means to store filter input samples and coefficients. Block RAMs are not as abundant as the smaller distributed RAMs found in a nearby DSP58, making them an excellent option for smaller FIR filters. The following figure illustrates the actual single-multiplier MACC FIR filter implementation using distributed RAM for the coefficient bank and an SRL16 for the data buffer.

Figure 44: 16-Tap Distributed RAM MACC FIR Filter

Note:

[1] To save the inverter, set IS_OPMODE_INVERTED 000100000



X21476-100820

MACC FIR Filter

A common filter implementation uses the multiply-accumulate (MACC) finite impulse response (FIR) filter. This section describes the implementation using DSP58. The following cases are covered:

- Single-multiplier MACC FIR filter
- Symmetric MACC FIR filter

Single Multiplier MACC FIR Filter

The single-multiplier MACC FIR is one of the simplest DSP filter structures. The MACC structure uses a single multiplier with an accumulator to implement a FIR filter sequentially versus a full parallel FIR filter. This trade-off not only reduces hardware by a factor of N, but also reduces filter throughput by the same factor. The general FIR filter equation is a summation of products (also known as an inner product), defined as follows.

$$y_n = \sum_{i=0}^{N-1} X_{n-i} h_i$$

In this equation, a set of N coefficients is multiplied by N respective data samples, and the inner products are summed together to form an individual result. The values of the coefficients determine the characteristics of the filter (for example, low-pass filter, band-pass filter, and high-pass filter). The equation can be mapped to many different implementations (for example, sequential, semi-parallel, or parallel) in the different available architectures.

Refer to the [Resource Utilization Guideline](#) for an implementation of the MACC FIR filter and the guideline on choosing the resource to implement the data buffer. In the example design, the implementation uses SRL16 for the data buffer. To support rounding toward infinity, apply the value of $(2^{(\text{fractional_part}-1)} - 1)$ to the C input. The multiplier followed by the accumulator sums the products over the same number of cycles as there are coefficients. With this relationship, the performance of the MACC FIR filter is calculated using the following equation.

$$\text{Maximum Input Sample Rate} = \frac{\text{Clock Speed}}{\text{Number of Taps}}$$

If the coefficients possess a symmetric shape, a slightly costlier structure is available (see [Symmetric MACC FIR Filter](#)), however, the maximum sampled rate is doubled. The sample rate of the costlier structure is defined as follows.

$$\text{Sample Rate} = \frac{\text{Clock Speed}}{\frac{1}{2} \text{Number of Taps}}$$

The nature of the FIR filter, with numerous MACC operations, outputs a larger number of bits from the filter than are present on the filters input. This effect is the bit growth or the gain of a filter. Due to the large output width, the full precision result is typically rounded and quantized. However, it is important to calculate the full precision output to select the correct bits from the output of the MACC.

One technique assumes every value in the filter could be the worst possible for the size of the two's complement numbers specified. Using the generic saturation level is a good starting point when the coefficients are unknown, but the number of bits required to represent them is known, for example, if the coefficients are re-loadable, as in adaptive filters. The equation of the output width in this case is given as follows.

$$\text{Output Width} = \text{ceil} \left(\log_2 \left(1 + \left(2^{(b-1)} \right) \times \left(2^{(c-1)} \right) \times N \right) + 1 \right)$$

where:

ceil: Rounds up to the nearest integer

b: Number of bits in the data samples

c: Number of bits in the coefficients

It is possible to use two clocks in the MACC FIR implementation. The faster clock goes to the DSP and the coefficient memory; the slow clock goes to the PL. It is therefore possible to avoid the condition in which the input has to be held for the number of taps cycles (the input throughput in the implementation using one clock only is equal to the number of taps).

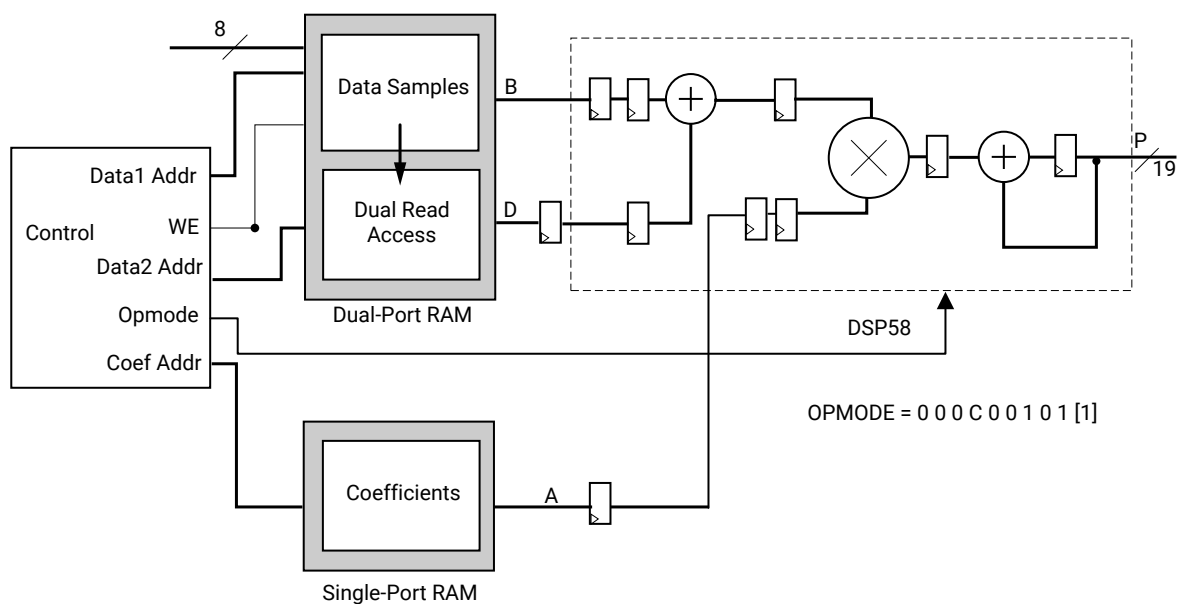
Symmetric MACC FIR Filter

For symmetric FIR filter coefficients, the capable sample rate performance of a MACC FIR filter can be doubled (assuming the same clock speed). By rearranging the following FIR filter equation in the case of even number of taps, the coefficients are exploited as follows.

$$(x_0 * c_0) + \dots + (x_n * c_n) = [(x_0 + x_n) \times c_0] + \dots \left(\text{if } c_i = c_{(n-i)}, \text{ with } i = 0, 1, \dots, \left(\frac{n+1}{2}\right) - 1 \right)$$

The following figure shows the architecture for a symmetric MACC FIR filter.

Figure 45: Symmetric MACC FIR Filter



X21477-100820

There are limitations to using the symmetric MACC FIR filter. The data (to A and D ports) and coefficients (to B port) are limited to 27 and 24 bits to fit into one DSP58.

Along with the three memory ports, additional filter resources are required to support symmetry. The control portion increases in resource utilization because the data is read out of one port in a forward direction and in reverse on the second port. This technique must only be used when extra sample rate performance is required.

Semi-Parallel FIR Filter

A common filter implementation to exploit available clock cycles, while still achieving moderate to high sample rates, is the semi-parallel FIR filter (also known as folded-hardware). The DSP58 allows creation of optimum filter structures of semi-parallel nature that in turn save resources and potential clock cycles.

In terms of the range of sample rate covered and number of coefficients, the semi-parallel FIR structure fills in the region between parallel FIR filters and sequential FIR filters. The structure implements the general FIR filter equation of a summation of products similar to the one described in the single multiplier MACC FIR filter.

$$y_n = \sum_{i=0}^{N-1} X_{n-i} h_i$$

Along with achievable clock speed and the number of coefficients (N), the number of multipliers (M) is also a factor in calculating semi-parallel FIR filter performance. The following equation demonstrates that the more multipliers used, the greater the achievable performance of the filter.

$$\text{Maximum Input Sample Rate} = \frac{\text{Clock Speed}}{\text{Number of Coefficients}} \times \text{Number of Multipliers}$$

The maximum input sample rate equation can be rearranged as follows to determine the number of multipliers to use for a particular semi-parallel architecture.

$$\text{Number of Multipliers} = \frac{\text{Maximum Input Sample Rate} \times \text{Number of Coefficients}}{\text{Clock Speed}}$$

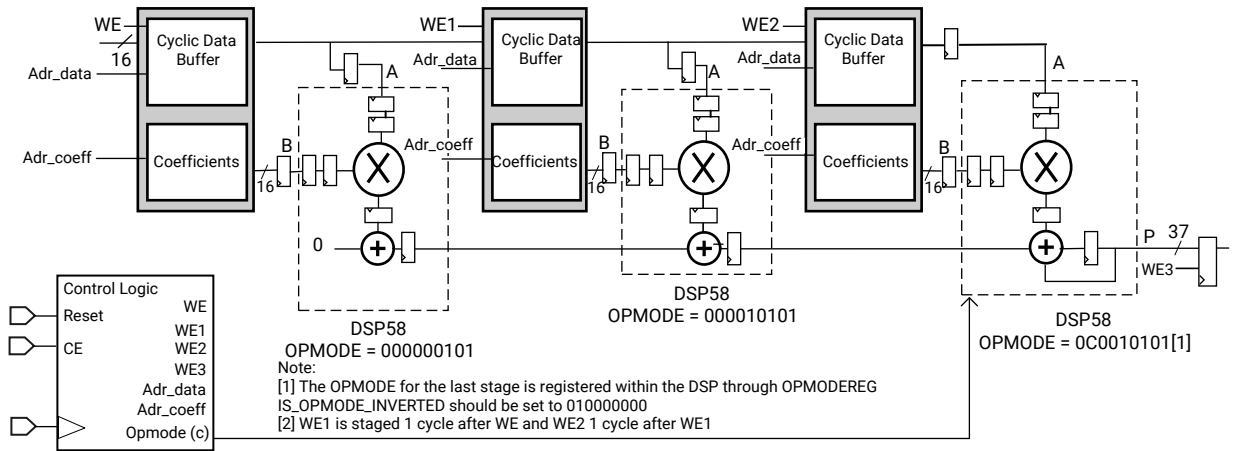
The number of clock cycles between each result of the FIR filter is determined by the following equation.

$$\text{Number of Clock Cycles per Result} = \frac{\text{Number of Coefficients}}{\text{Number of Multipliers}}$$

Three Multiplier Semi-Parallel FIR Filter

For the semi-parallel FIR case with moderate to high sample rate and large number of coefficients, the filter structure is chosen to be the three-multiplier, block RAM based, semi-parallel FIR filter. The following figure is a block diagram of the filter.

Figure 46: Three-Multiplier, Block RAM Based, Semi-Parallel FIR Filter

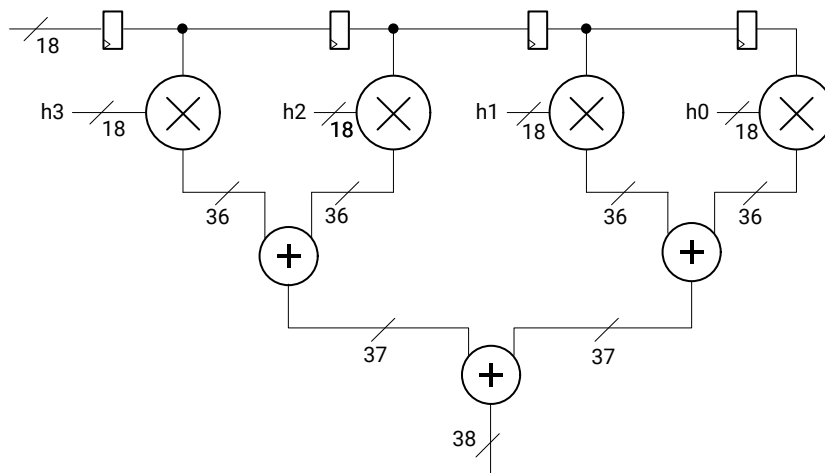


In this implementation, one memory buffer is required to hold the coefficients and also the input data history values. The block RAM can be used in dual-port mode with a cyclic data buffer established in the first half of the memory to serve the shifting input data series. There are 16 taps for each DSP and the overall number of taps in this design is 48.

Parallel FIR Filter

The basic parallel architecture, illustrated in the following figure, is referred to as the direct form type 1 filter. The final stages of the adder tree structure is usually where the performance bottleneck could increase cost, logic, and power. The adder cascade implementation accomplishes the post addition process with minimal silicon resources by using the cascade path within DSP58.

Figure 47: Direct Form Type 1 FIR Filter



This structure implements the general FIR filter equation of a summation of products as defined in the following equation.

$$y_n = \sum_{i=0}^{N-1} X_{n-i} h_i$$

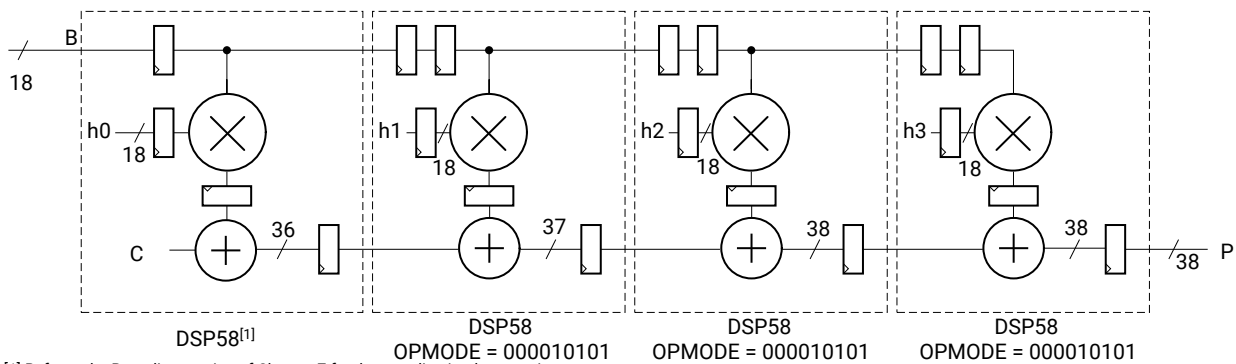
In the above equation, a set of N coefficients is multiplied by N respective data samples. The results are summed together to form an individual result. The values of the coefficients determine the characteristics of the filter (for example, a low-pass filter).

Other more optimal solutions for parallel filter architectures are covered in the following sections.

Systolic FIR Filter

The systolic FIR filter is considered an optimal solution for parallel filter architectures. The systolic FIR filter also uses adder chains to be able to take full advantage of the DSP58 architecture (see the following figure).

Figure 48: Systolic FIR Filter



[1] Refer to the Rounding section of Chapter 7 for the rounding implementation.

X21482-111820

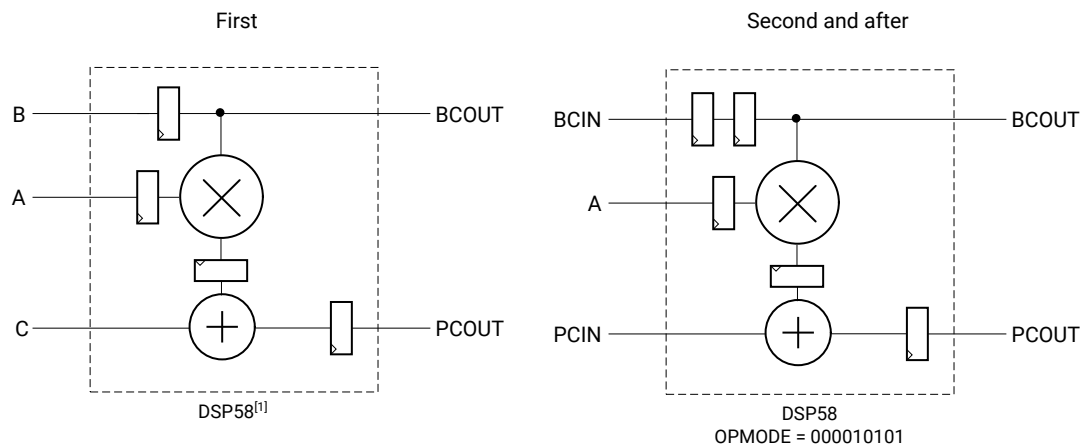
The input data is fed into a cascade of registers acting as a data buffer. Each register delivers a sample to a multiplier where it is multiplied by the respective coefficient. The coefficients are aligned from left to right with the first coefficients on the left side of the structure. The adder chain stores the gradually combined inner products to form the final result. No external logic is required to support the filter and the structure is extendable to support any number of coefficients.

Note: Dedicated cascade connections (PCOUT and PCIN) are leveraged to achieve maximum performance (adder chain structure versus adder tree).

The configuration of DSP58 for each segment of the systolic FIR filter is shown in the following figure. Apart from the very first segment, all processing elements have the same structure. If rounding is performed, the ALU in the first segment must be driven by the C input (dynamic/static rounding) or RND attribute (static rounding) with the correct value. For all DSP instances, except the first instance, OPMODE is set to feed the ALU with the multiplier result of the same instance and the result from the previous DSP in the chain through the dedicated cascade path (PCOUT → PCIN). Notice that the two leftmost bits of OPMODE (through the WMUX) can be used if rounding is implemented. The dedicated cascade input in the first DSP instance (BCIN) and dedicated cascade output (BCOUT) are used to create the necessary input data buffer cascade.

Note: This design is supported by inference, therefore, the A and B inputs can be swapped depending on the tool choice. This means that ACIN and ACOUT (instead of BCIN and BCOUT) can be used to create the cascade.

Figure 49: Systolic Multiply-Add Processing Element



[1] Refer to the Rounding section of Chapter 7 for the rounding implementation.

X21483-041719

The advantages of using the systolic FIR filter are as follows.

- **Highest Performance:** Maximum performance can be achieved with this structure because there is no high fanout input signal. Dedicated cascading avoids the need to pass through programmable interconnect. Larger filters can be routing-limited if the number of coefficients exceeds the number of DSP Engines in a column on a device.
- **Efficient mapping to the DSP58:** Mapping is enabled by the adder chain structure of the systolic FIR filter. This extendable structure supports large and small FIR filters.
- **No External Logic:** No external programmable logic is required, thus enabling the highest possible performance.

The disadvantages of using the systolic FIR filter are as follows.

- **Higher Latency:** The latency of the filter is a function of the number of coefficients present in the filter. The larger the filter, the higher the latency.

- **More Resource Usage:** Larger number of DSPs are used compared to the MACC FIR filter.

Coding examples will be available in Language Templates from the Vivado® Integrated Design Environment (IDE) in a subsequent release.

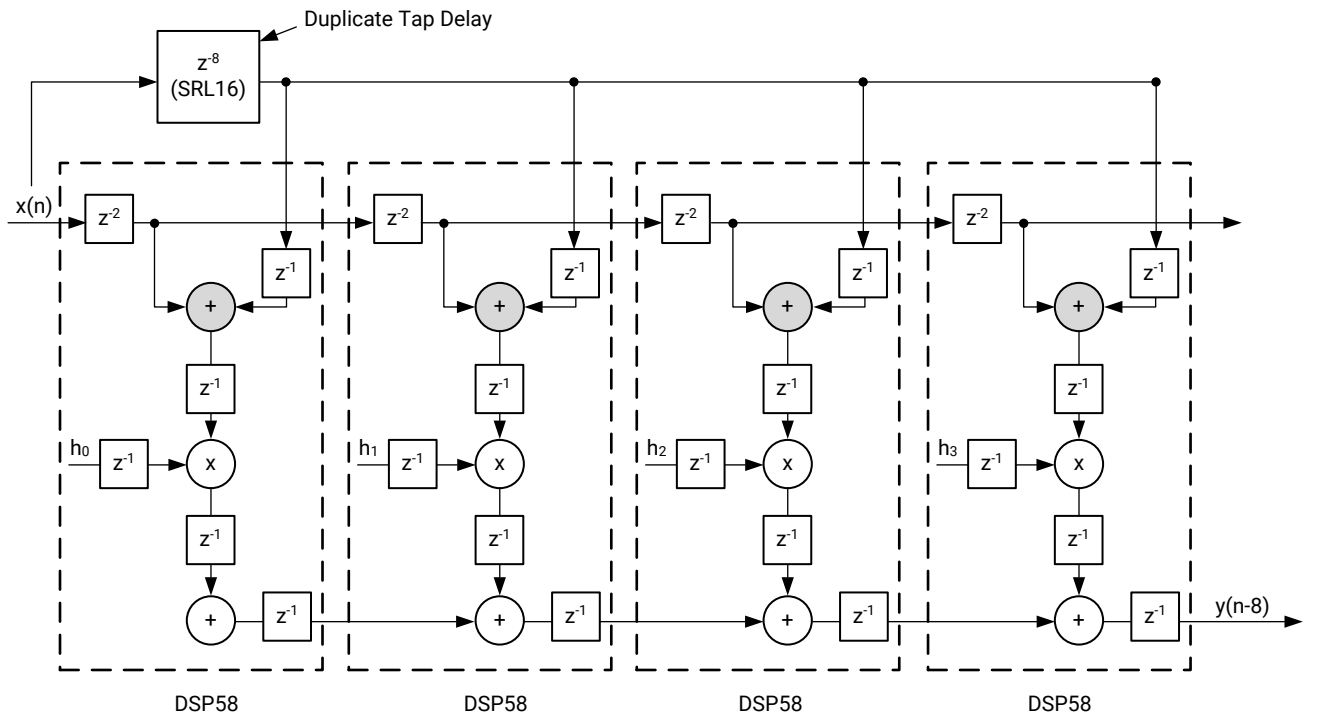
Symmetric Systolic FIR Filter

Similar to the MACC FIR filters where symmetry was examined, exploiting symmetry is extremely powerful in parallel FIR filters because it halves the required number of multipliers, which is advantageous due to the finite number of DSP58s. The following equation is valid in the case of even number of taps and demonstrates how the data is pre-added before being multiplied by the single coefficient.

$$(x_0 * c_0) + \dots + (x_n * c_n) = [(x_0 + x_n) \times c_0] + \dots \left(\text{if } c_i = c_{(n-i)}, \text{ with } i = 0, 1, \dots, \left(\frac{n+1}{2}\right) - 1 \right)$$

The following figure shows the implementation of a symmetric systolic FIR filter.

Figure 50: Symmetric Systolic FIR Filter



Note: The pre-adders (shaded in gray) in DSP58 are used instead of implementing them in the programmable logic (PL). The register delays in the input buffer time series are implemented as SRL16 and are spread evenly across the DSP58s.

Coding examples are available in Language Templates with the Vivado® Integrated Design Environment (IDE) 2020.2 version.

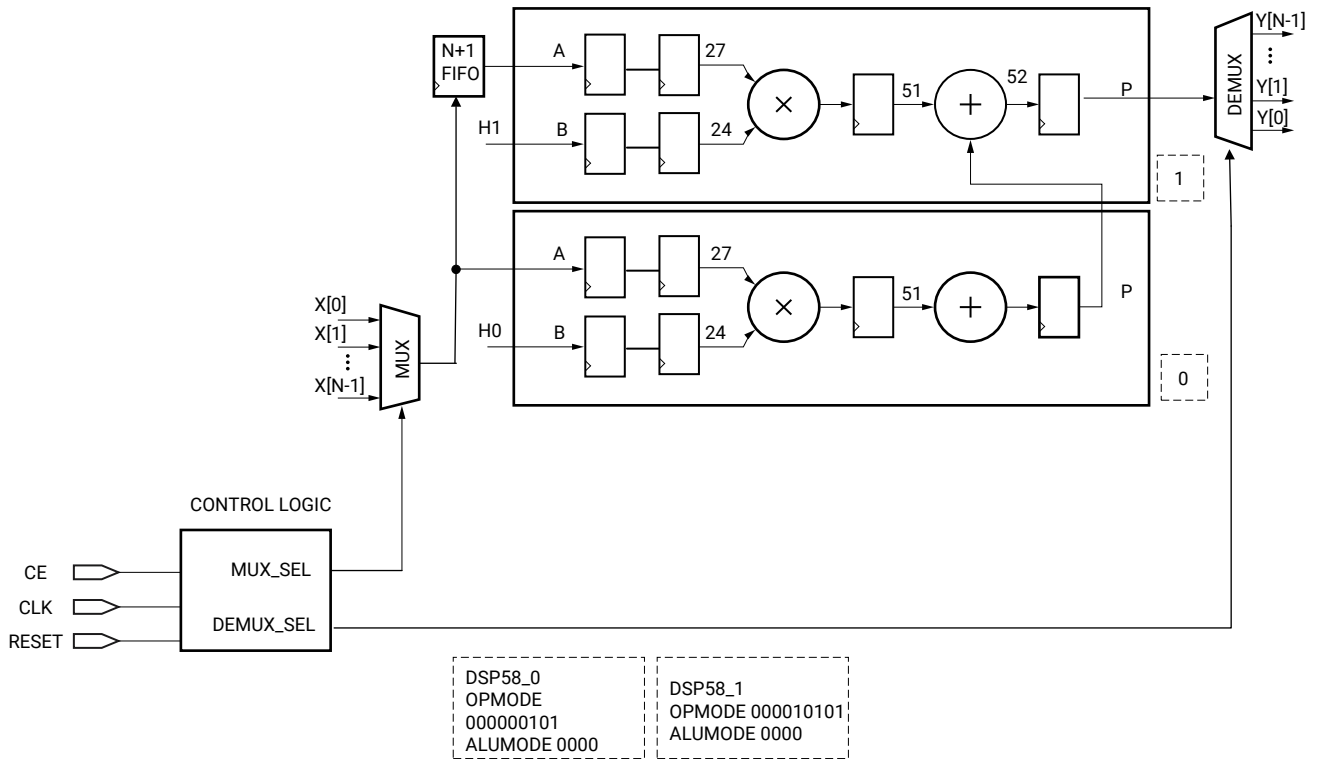
Multichannel FIR Filter

Multichannel filtering is used in applications such as wireless communication, image processing, and multimedia applications. The main advantage of using a multichannel filter is leveraging very fast math elements across multiple input streams (that is, channels) with much lower sample rates. This technique increases silicon efficiency by a factor almost equal to the number of channels. In a typical multichannel filtering scenario, multiple input channels are filtered using a separate digital filter for each channel. Due to the high performance of DSP58, time division multiplexing can be used to filter up to N separate channels using one DSP58. The number of channels N is calculated using the following equation.

$$N \times \text{Channel Frequency} \leq \text{Maximum Frequency of DSP58}$$

Each DSP58 is clocked using an NX clock (DSP clock that is N times faster than the one driving the samples of each channel). The N input streams are converted to one serial, interleaved stream using the N to one multiplexer. To ensure functionality, the multiplexer and demultiplexer selectors must run at the same clock speed of DSP58. The N parallel interleaved streams are stored in an N + 1 FIFO (implemented as SRLs). This implementation uses 1/Nth of the total Versal™ ACAP resource as compared to implementing each channel separately. The following figure shows a two-tap, N-channel filter implementation.

Figure 51: Multichannel FIR Filter



X21487-102620

Multirate FIR Filter

Multirate filtering is used to change the rate or frequency of sampling of an input signal to an arbitrary rate or frequency at the output. Multirate filtering is widely used in video applications. DSP58 is ideally suited to implement multirate sampling because of its high speed and filter-like structure. The cascaded data input and output paths, pipeline registers, high precision two's complement multiplier followed by an adder/subtractor, and accumulation capability provide needed elements for multirate filtering.

Polyphase interpolating and decimating FIR filters are used to implement multirate filters using up-sampling and down-sampling techniques. They are covered in the following sections.

Interpolating

Increasing the number of samples representing a signal is called interpolating or up-sampling. Interpolation is used in applications like medical imaging and SDTV-to-HDTV conversions. A 12-tap 1:4 interpolator is illustrated in the following figure. A 1:4 interpolation results in four output samples for every one input sample. The 12 taps are the 12 coefficients that are used to calculate the four output samples. The following equations describe the relation between the input samples (x), coefficients (h), and the output samples (y).

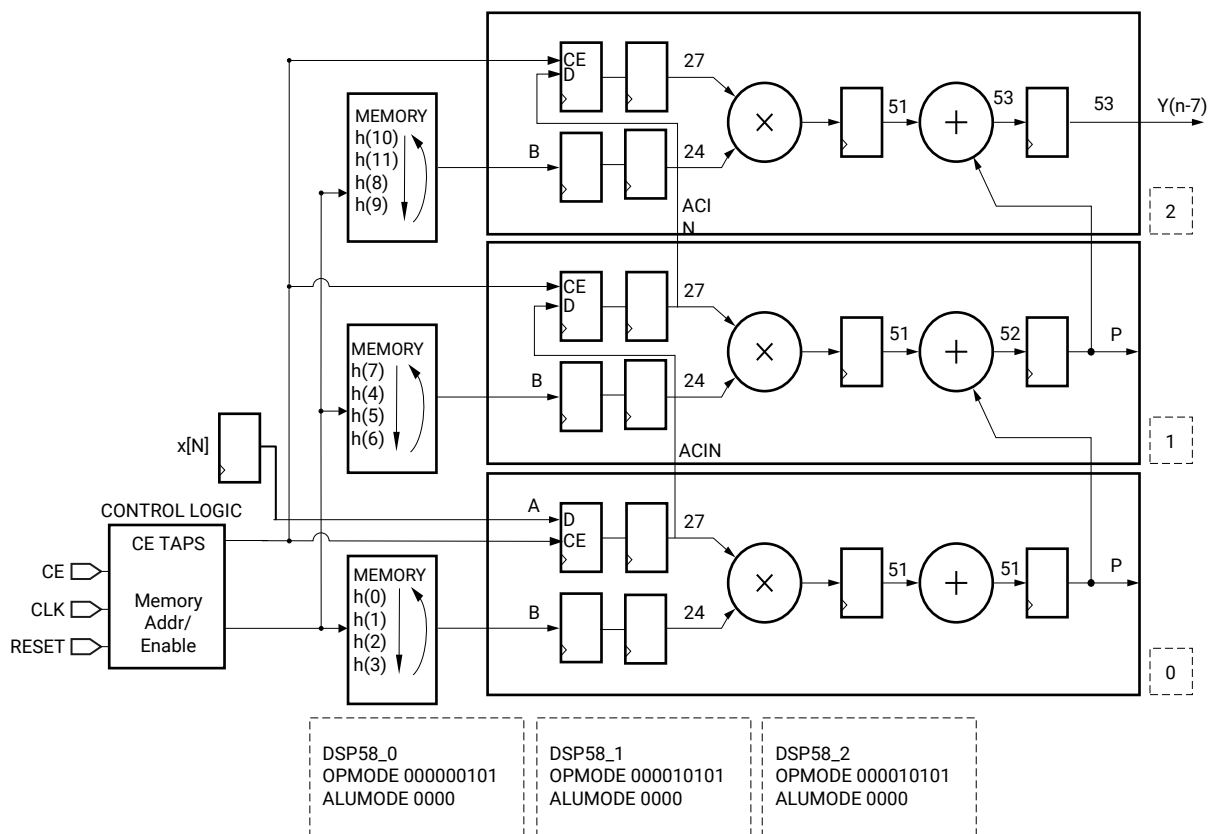
$$y_0 = (h_0 \times x_n) + (h_4 \times x_{n-1}) + (h_8 \times x_{n-2})$$

$$y_1 = (h_1 \times x_n) + (h_5 \times x_{n-1}) + (h_9 \times x_{n-2})$$

$$y_2 = (h_2 \times x_n) + (h_6 \times x_{n-1}) + (h_{10} \times x_{n-2})$$

$$y_3 = (h_3 \times x_n) + (h_7 \times x_{n-1}) + (h_{11} \times x_{n-2})$$

Figure 52: 12-Tap 1:4 Interpolator



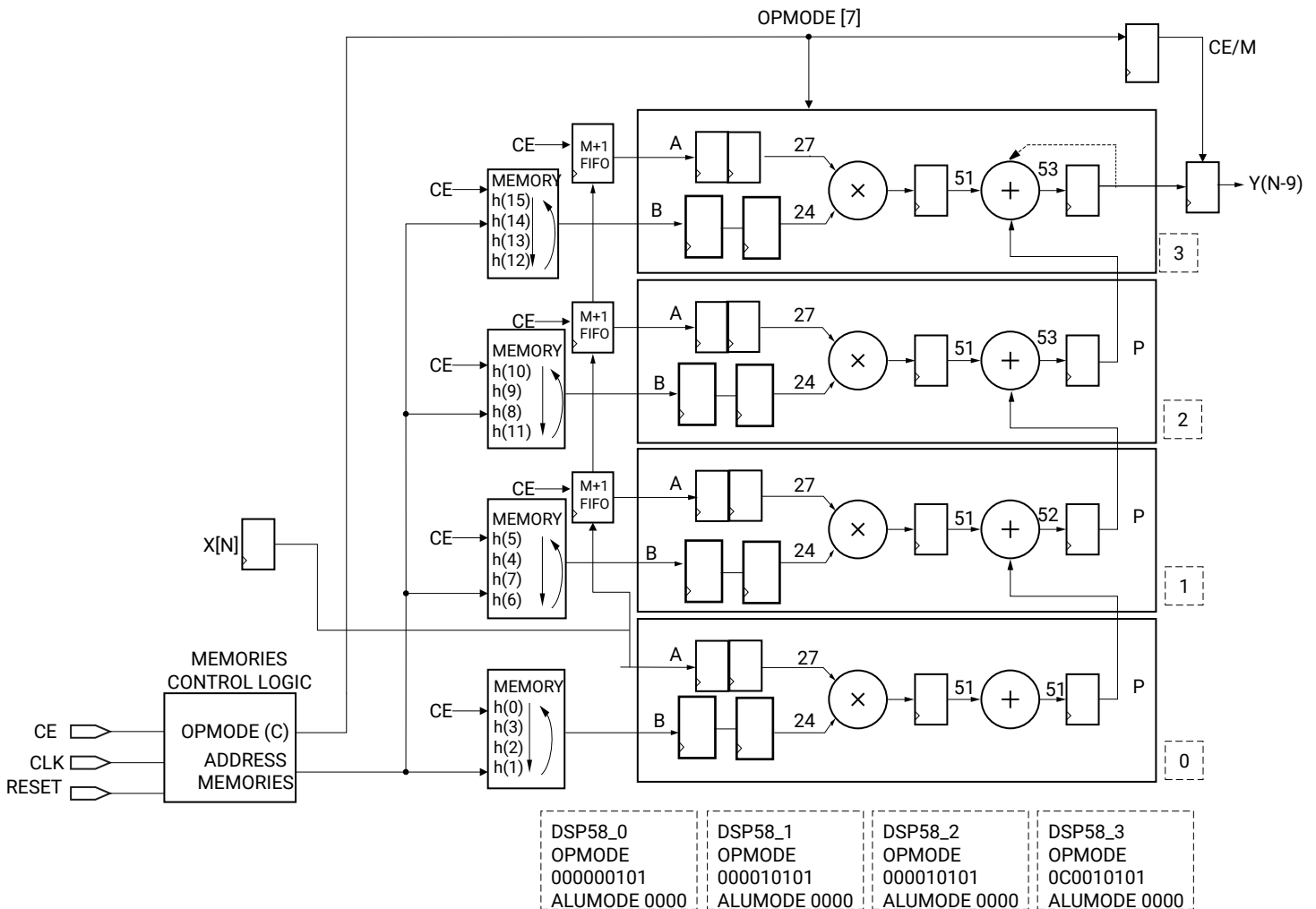
X21492-102620

In the interpolating FIR filter design, the phases are related to the number of DSP58s, and the number of coefficients is related to the memory depth (if RAM is used) or number of registers (SRL16). Referring to the figure above, the 12 coefficients are cycled through the DSP58s. These coefficients can be stored in shift registers (SRL16) or memories and grouped in each DSP58.

Decimating

Decimating is the process of reducing the number of samples representing a signal. Decimation is used in video applications like 4:4:4-to-4:2:2 conversions and HDTV-to-SDTV conversions. The following figure illustrates a 16-tap 4:1 decimator implemented using four parallel filters. The phases are related to the number of DSP58s and the number of coefficients is related to the memory depth (if RAM is used) or number of registers (SRL16). For every four-input sample, the decimator produces one output sample.

Figure 53: 16-Tap 1:4 Decimator



X21486-100820

For the 16-tap filter, the output sample (y) is the weighted average of 16 input samples (x) multiplied by 16 coefficients (h) as described in the following equation.

$$y_{[n]} = (h_0 \times x_n) + (h_1 \times x_{n-1}) + (h_2 \times x_{n-2}) + (h_3 \times x_{n-3}) + (h_4 \times x_{n-4}) + (h_5 \times x_{n-5}) + (h_6 \times x_{n-6}) + (h_7 \times x_{n-7}) + (h_8 \times x_{n-8}) + (h_9 \times x_{n-9}) + (h_{10} \times x_{n-10}) + (h_{11} \times x_{n-11}) + (h_{12} \times x_{n-12}) + (h_{13} \times x_{n-13}) + (h_{14} \times x_{n-14}) + (h_{15} \times x_{n-15})$$

The input signals to each DSP58 are delayed by $(M + 1)$ clocks from the previous DSP58. Shift registers are used to achieve this delay. An initial latency is given by the following equation.

$$(Number\ of\ Flop\ Stages\ in\ Each\ DSP58) + (Number\ of\ Phases - 1) + 2\ (One\ Stage\ Before\ and\ After\ the\ Filter)$$

The 16-tap filter output is obtained at the fourth DSP58. The OPMODE must be changed dynamically to ensure functionality.

Floating Point FIR Filter

The design in this use case essentially implements the single multiplier MACC FIR filter in a floating point data format. Refer to the figure in [Resource Utilization Guideline](#) for a simplified block diagram of the design. Note the following:

- The DSPFP32 replaces the DSP58 primitive and consists of a floating-point multiplier and a floating-point adder.
- The input and output are in IEEE binary32 format.
- The single-port block RAM is partitioned into two: one half is dedicated to the incoming data, the other half is used as ROM for the coefficients.
- FPOPMODE[6:0] controls the behavior of the floating-point arithmetic unit and is configured to be {00c0001}. The OPMODE bit 'c' is sent to the DSP from the control logic and dynamically selects the input to the ALU to enable or disable the P output feedback path. This is done to reset the accumulator from one output sequence to another. This approach means avoiding a reset to the output register, saving a clock cycle.

Complex FIR Filter

The complex FIR equation is represented as follows.

$$\tilde{y}[n] = \sum_{i=0}^{L-1} \tilde{x}[n-i] \tilde{a}_i$$

where:

the i th complex coefficient is:

$$\tilde{a}_i = a_{l,i} + ja_{Q,i}$$

the complex input sample is:

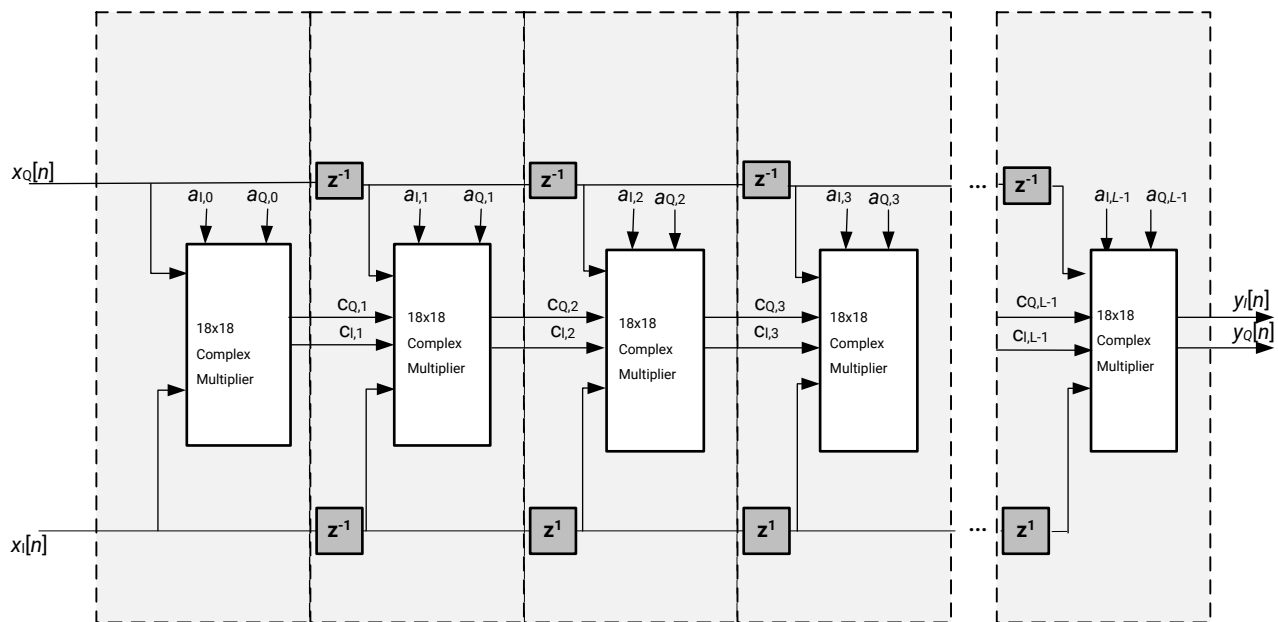
$$\tilde{x}[n] = x_I[n] + jx_Q[n]$$

the complex output sample is:

$$\tilde{y}[n] = y_I[n] + jy_Q[n]$$

The following figure is a simplified block diagram of the filter design.

Figure 54: Simplified Diagram of Complex FIR Filter



X22369-051420

Note:

- 'x' corresponds to complex input and maps to B_RE/B_IM ports.
- 'a' corresponds to complex coefficient and maps to A_RE/A_IM ports. The design is provided as an inference example therefore, the inputs can be swapped depending on the tool choice.
- 'c' corresponds to cascade path from PCOUT_RE/IM ports of one DSP58 to PCIN_RE/IM ports of the next.

In the DSPCPLX configuration, each tap in the complex FIR filter is comprised of two DSP58s (provided they are 18 bits wide complex numbers or fewer) configured as a complex multiply-add unit. The PCOUT/PCIN ports (real and imaginary portion) are used to cascade in the output of the final adder from the previous tap to the next (shown as c_Q and c_I in the diagram, except the first tap). The z^{-1} blocks represent delay stages for pipelining.

Coding examples are available in Language Templates with the Vivado® Integrated Design Environment (IDE) 2020.2 version.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Documentation Navigator and Design Hubs

Xilinx[®] Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado[®] IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, select **Start** → **All Programs** → **Xilinx Design Tools** → **DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on DocNav, see the [Documentation Navigator](#) page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. *UltraScale Architecture DSP Slice User Guide* ([UG579](#))
2. [IEEE Standard for Floating-Point Arithmetic](#)
3. *Vivado Design Suite User Guide: Synthesis* ([UG901](#))
4. *Versal Architecture and Product Data Sheet: Overview* ([DS950](#))

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2020 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.