DSP Macro v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG323 (v1.0) January 25, 2021

Table of Contents

Introduction

The Xilinx® LogiCORE™ DSP Macro provides an easy-to-use interface that abstracts the DSP Slice and simplifies its dynamic operation by enabling the specification of multiple operations using a set of user-defined arithmetic expressions. The specified operations are enumerated and can be selected through a single port on the generated core.

Features

- Simplified and abstracted interface to DSP Slice enhances ease of use, code readability and portability
- Define DSP Slice operation using a list of user-defined arithmetic expressions
- Support for up to 64 instructions
- Supports the DSP Slice pre-adder
- Configurable pipelining
- Choose between DSP Slice or device logic implementation

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Versal™ ACAP, UltraScale+™, UltraScale™, Zynq®-7000 SoC, 7 series
Supported User Interfaces	N/A
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Not Provided
Simulation Model	Encrypted VHDL
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite System Generator for DSP
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: N/A
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado® IP catalog.
2. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.

XILINX

Chapter 2

Overview

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:

- **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:
  - Port Descriptions
  - Clocking
  - Resets
  - Customizing and Generating the Core

Core Overview

The DSP Macro core allows straightforward configuration of the DSP Slice by specifying user-defined instructions. Multiple instructions can be specified, and the instruction being performed can be changed dynamically at run time.

The DSP Macro core supports up to 64 separate instructions, which can be selected dynamically though a single control port. The pipeline stages in the core can be allocated automatically in the user interface in the Vivado® Integrated Design Environment (IDE), specified in tiers, or you can individually specify them. The option to enable and specify the use of DSP Slice cascade ports allows multiple DSP Macro cores to be efficiently connected to construct a larger circuit.

Applications

The DSP Macro core is used in the following applications:

- Basic DSP Slice operations such as accumulator, multiplier, adder
- Time-shared math operations using a single DSP Slice such as complex multiplication
- DSP engine / co-processor
- MAC FIR

Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

***Note*:** To verify that you need a license, check the License column of the IP Catalog. Included means that a license is included with the Vivado® Design Suite; Purchase means that you have to purchase a license to use the core.

Information about other Xilinx® LogiCORE™ IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

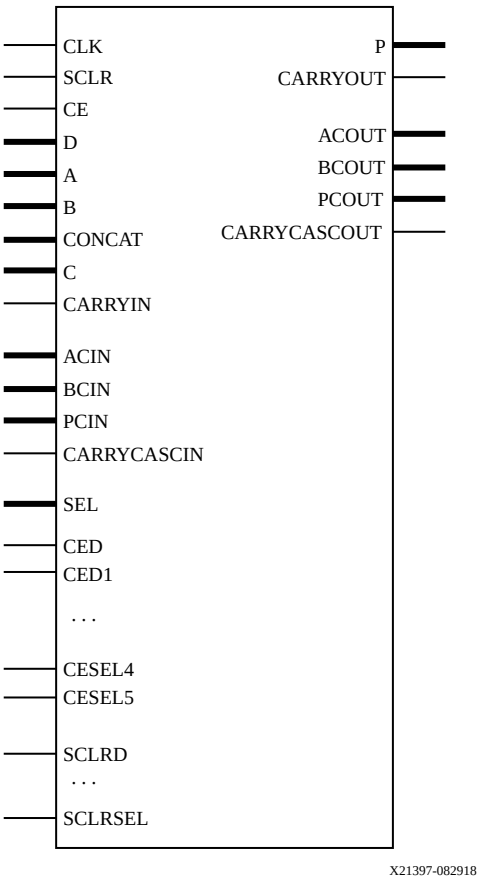
- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT! *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

The functional block diagram of the core is shown in the following figure.

Figure 1: **Core Schematic Symbol**



Port Descriptions

The core signal ports are shown in the following table.

Table 1: **Core Signal Pinout**

Port Name	I/O	Description
CLK[1]	I	Clock – active rising edge.
CE	I	Clock Enable – core clock enable (active-High).
SCLR	I	Synchronous Clear – synchronous reset (active-High). Asserting SCLR synchronously with CLK resets all registers. SCLR has priority over CE.
D [d_width-1:0]	I	D port – primary input to DSP Slice pre-adder. The maximum d_width is 25 bits for 7 series devices and 27 bits for UltraScale™ and Versal™ devices.
A [a_width-1:0]	I	A port – input to DSP Slice multiplier and secondary input (subtrahend) to pre-adder. The maximum a_width is 25 bits for 7 series devices and 27 bits for UltraScale and Versal devices.
ACIN [ac_width:0]	I	Cascaded A port – used as per the A port but must be driven by the ACOUT of the previous DSP Slice, avoids routing and logic. Static selection between A and ACIN is made by the specified DSP Macro instructions. The maximum ac_width is 25 bits for 7 series devices and 27 bits for UltraScale and Versal devices, signed-extended to 30 bits for UltraScale and to 34 bits for Versal.
B [b_width-1:0]	I	B port – second input to multiplier. Maximum b_width: 18 bits for 7 series and UltraScale devices and 24 bits for Versal devices.
BCIN [bcin_width-1:0]	I	Cascaded B port. Must be driven by the BCOUT of the previous DSP Slice. Static selection between B and BCIN. Fixed bcin_width: 18 bits for 7 series and UltraScale devices and 24 bits for Versal devices.
CONCAT [concat_width-1:0]	I	CONCAT port – concatenation of the A and B DSP Slice inputs. The D input is included in the concatenation. Input to second stage add/sub. Mutually exclusive to the A,B and D ports. The maximum concat_width is 48 bits for 7 series and UltraScale devices and 58 bits for Versal devices. **Note**: The A port, when specified, can be passed directly to the add/sub.
C [c_width-1:0]	I	C port – input to DSP Slice add/sub. Maximum c_width: 48 bits for 7 series and UltraScale devices and 58 bits for Versal devices.
PCIN [pcin_width-1:0]	I	PCIN port – cascaded P input from the previous DSP Slice. Input to add/sub. Avoids device routing and provides a low latency path. Fixed pcin_width: 48 bits for 7 series and UltraScale devices and 58 bits for Versal devices.
CARRYIN	I	CARRYIN port – carry input from device logic, single bit.
CARRYCASCIN	I	CARRYCASCIN port - cascaded carry input from the previous DSP Slice. Can be used to construct large adders.
SEL [sel_width-1:0]	I	SEL port - selects from the enumerated list of instructions specified in the core user interface. Unsigned. Fixed sel_width: ceil(log2(num_instructions))
CED1..3	I	Clock enables for D path registers (active-High)
CEA1..4	I	Clock enables for A path registers (active-High)
CEB1..4	I	Clock enables for B path registers (active-High)
CECONCAT3..5	I	Clock enables for CONCAT path registers (active-High)
CEC1..5	I	Clock enables for C path registers (active-High)
CEM	I	Clock enables for M path registers (active-High)
CEP	I	Clock enables for P path registers (active-High)
CESEL1..5	I	Clock enables for SEL path registers (active-High)

Table 1: **Core Signal Pinout** *(cont'd)*

Port Name	I/O	Description
SCLRD	I	Synchronous reset for D path registers (active-High)
SCLRA	I	Synchronous reset for A path registers (active-High)
SCLRB	I	Synchronous reset for B path registers (active-High)
SCLRCONCAT	I	Synchronous reset for CONCAT path registers (active-High)
SCLRC	I	Synchronous reset for C path registers (active-High)
SCLRM	I	Synchronous reset for M path registers (active-High)
SCLRP	I	Synchronous reset for P path registers (active-High)
SCLRSEL	I	Synchronous reset for SEL path registers (active-High)
P [p_msb-p_lsb:0][1]	O	P port - output from DSP Slice add/sub, provides the result of the selected instruction. Maximum p_msb: 47 bits for 7 series and UltraScale devices and 57 bits for Versal devices ***Note*:** p_msb and p_lsb are derived from the Output Port Properties; Full Precision Width and Width. See Implementation for more details.
CARRYOUT	O	CARRYOUT port - carryout to device logic from add/sub.
ACOUT [ac_width-1:0]	O	ACOUT port - optional cascade A output, ac_width defined for ACIN.
BCOUT [bc_width-1:0]	O	BCOUT port - optional cascade B output, bc_width defined for BCIN.
PCOUT [pc_width-1:0]	O	PCOUT port - optional cascade P output, pc_width defined for PCIN.
CARRYCASCOUT	O	CARRYCASCOUT port - optional cascade carryout output.

Notes:

1. All signals are optional with the exception of CLK and P[p_msb-p_lsb:0].

Related Information

Implementation Page

Designing with the Core

This section includes guidelines and additional information to facilitate designing with the core.

Clocking

The DSP Macro requires a single clock, CLK, and is active-High triggered. Active-High clock enables can be selected on either a global or a per-register basis.

Resets

The DSP Macro has the option of selecting a global synchronous reset, or a reset per tier. Asserting an SCLR pin for a single cycle resets the relevant registers. SCLR always overrides CE for all registers.

Using the DSP Macro IP Core

The core user interface performs error-checking on all input parameters. Several files are produced when a core is generated. For detailed instructions, see the*Vivado Design Suite User Guide: Designing with IP* (UG896).

Instruction Format

The instructions are case insensitive and ignore spaces between operands. The left side of the arithmetic expression, P=, is implicitly declared and should not be specified.

Valid Operands	D, A, ACIN, B, BCIN, CONCAT, C, PCIN, CARRYIN, CARRYCASCIN, 0	
Valid Operators	+, -, *, >>17 | >>23	The >>17 operator targets the DSP48 17-bit wire shift; valid only for P and PCIN operators and not valid for Versal™ devices. In Versal devices, this becomes a 23-bit wire shift and the operator changes to >>23, targeting the DSP58 23-bit wire shift.

Valid Functions	rndsimple, rndsym, rndmacc	Rounding functions require that the P output width is less than full precision. See Supported Functions for further details.

Related Information

Supported Functions

Examples

Accumulator

- **C:** Load
- **C+P:** Accumulate (add)
- **P-C:** Accumulate (subtract)

Three-input add

- C+CONCAT+P

Multiply Accumulate

- **A*B:** Load
- **P+A*B:** Multiply accumulate

15-tap symmetric filter, where data is provided sequentially on the A,D and B inputs. A and D are used for the data values and B is used for the filter coefficients.

- **(A+D)*B:** Taps 1 & 15
- **(A+D)*B+P:** Taps 2...7 & 14...9
- **A*B+P:** Tap 8
- **rndsym(P):** Round result

Construction of Supported Instructions

The following sections illustrate how the list of supported instructions are constructed. The notation is vaname = [I1,I2,...].

This indicates that `varname` supports the list of operator/operand combinations I1, I2, ...

7 Series Devices

preadder = [(A+D) , (D+A) , (D-A), D, -A , A, (ACIN+D) , (D+ACIN) , (D-ACIN), -ACIN , ACIN]
mult_ip1 = [ACIN , A , preadder]

mult_ip2 = [BCIN , B , 1]
mult = [mult_ip1 * mult_ip2 , mult_ip2 * mult_ip1]

The 1 is not explicitly required when defining an instruction; the 1 and corresponding * operator are ignored.

xmux = [CONCAT , P , 0]
ymux = [C , 0]
zmux = [C , PCIN , P , P>>17 , PCIN>>17, 0]
cinmux = [CARRYIN , CARRYCASCIN , 0]

Similarly, 0 is not explicitly required when defining an instruction. The 0 and corresponding operator are ignored.

xycomb = [xmux + ymux , ymux + xmux , mult]

valid instructions = [

xycomb + zmux + cinmux ,
xycomb - zmux,
zmux + xycomb + cinmux ,
zmux - (xycomb + cinmux) ,
zmux - xmux - ymux - cinmux ,
zmux - ymux - xmux - cinmux ,
-zmux + xmux + ymux,
-zmux + ymux + xmux,
-zmux - xmux,
rndsimple(zmux + xmux + cinmux)
rndsimple(mult + cinmux)
rndsym([P , PCIN])]
rndsym(mult)
rndmacc(mult + P)

Note that for the rndsimple functions, operand C is not supported for zmux.

UltraScale Devices

preadder = [(A+D) , (D+A) , (D-A), D, -A , A, (ACIN+D) , (D+ACIN) , (D-ACIN), -ACIN , ACIN]
mult_ip1 = [ACIN , A , preadder]
mult_ip2 = [BCIN , B , 1 , preadder]
mult = [mult_ip1 * mult_ip2 , mult_ip2 * mult_ip1]

The 1 is not explicitly required when defining an instruction; the 1 and corresponding * operator are ignored.

Note: The preadder option in mult_ip2 enables a squaring operation. The operator selected from the preadder for mult_ip1 must match the preadder instruction in mult_ip2.

wmux = [C , P , 0]
xmux = [CONCAT , P , 0]
ymux = [C , 0]
zmux = [C , PCIN , P , P>>17 , PCIN>>17, 0]
cinmux = [CARRYIN , CARRYCASCIN , 0]

Similarly, 0 is not explicitly required when defining an instruction. The 0 and corresponding operator are ignored.

xycomb = [wmux + xmux + ymux , wmux + ymux + xmux , mult]

valid instructions = [

xycomb + zmux + cinmux ,
zmux + xycomb + cinmux ,
zmux - (xycomb + cinmux) ,
zmux - wmux - xmux - ymux - cinmux ,
zmux - wmux - ymux - xmux - cinmux ,
-zmux +wmux + xmux + ymux,
-zmux +wmux + ymux + xmux,
-zmux - wmux - xmux,
-zmux - xmux - wmux,
-zmux - mult,
rndsimple(zmux + xmux + ymux + cinmux)
rndsimple(mult + zmux + cinmux)
rndsym([P , PCIN])]
rndsym(mult)
rndmacc(mult +wmux + P)

Versal Devices

preadder = [(A+D) , (D+A) , (D-A), D, -A , A, (ACIN+D) , (D+ACIN) , (D-ACIN), -ACIN , ACIN]
mult_ip1 = [ACIN , A , preadder]
mult_ip2 = [BCIN , B , 1 , preadder]
mult = [mult_ip1 * mult_ip2 , mult_ip2 * mult_ip1]

The 1 is not explicitly required when defining an instruction; the 1 and corresponding * operator are ignored.

Note: The preadder option in mult_ip2 enables a squaring operation. The operator selected from the preadder for mult_ip1 must match the preadder instruction in mult_ip2.

wmux = [C , P , 0]
xmux = [CONCAT , P , 0]
ymux = [C , 0]
zmux = [C , PCIN , P , P>>23 , PCIN>>23, 0]
cinmux = [CARRYIN , CARRYCASCIN , 0]

Similarly, 0 is not explicitly required when defining an instruction. The 0 and corresponding operator are ignored.

xycomb = [wmux + xmux + ymux , wmux + ymux + xmux , mult]

valid instructions = [

xycomb + zmux + cinmux ,
zmux + xycomb + cinmux ,
zmux - (xycomb + cinmux) ,
zmux - wmux - xmux - ymux - cinmux ,
zmux - wmux - ymux - xmux - cinmux ,
-zmux +wmux + xmux + ymux,
-zmux +wmux + ymux + xmux,
-zmux - wmux - xmux,
-zmux - xmux - wmux,
-zmux - mult,
rndsimple(zmux + xmux + ymux + cinmux)
rndsimple(mult + zmux + cinmux)
rndsym([P , PCIN])]
rndsym(mult)
rndmacc(mult +wmux + P)

Restrictions

- CONCAT operand is mutually exclusive to `mult` operands. When any input to the multiplier is specified, the CONCAT operand is restricted for all instructions, or vice versa. The inclusion of 1 in `mult_ip2` enables all `mult_ip1` combinations to be used as direct inputs to the second stage add/sub.
- The choice between A and ACIN is static; after one is specified the other is restricted. Similarly for B and BCIN.
- The use of CARRYCASCIN is restricted to a subset of instructions.
- The inclusion of a squaring instruction (`mult_ip1` = `mult_ip2`) also has static implications. Only squaring operators can be used through `mult_ip1` and `mult_ip2`, that is, previously valid `x_mux`, `w_mux`, `z_mux`, and `y_mux` operators can still be used.

Supported Functions

Related Information

Instruction Format

RNDSIMPLE(arg)

RNDSIMPLE implements a non-symmetric round to negative; this is equivalent to the MATLAB® function `ceil(arg-0.5)`.

Table 3: **Illustration of RNDSIMPLE Return Values**

arg	RNDSIMPLE(`arg`)
<x.5	x
x.5	x
>x.5	x+1
>-x.5	-x
-x.5	-x-1
<-x.5	-x-1

The binary point of `arg` is defined by the full precision output width and the specified core output width; the core output is taken from the upper MSBs of the DSP Slice output with the remaining LSBs considered as the fractional portion. The binary point is taken as `full precision p_width - p_width`.

A rounding constant with a binary value of 0.0111...(or 0.499...) is added to `arg` and the LSBs removed by the process of reinterpreting the full precision output to the specified core output width. The LSBs remain on the accumulator.

`Arg` can include CARRYIN. This enables CARRYIN to determine the rounding direction. The assertion of CARRYIN is equivalent to adding 0.00...01. This modifies the rounding constant to 0.100.. (or 0.5) and therefore the rounding direction. This can be used to implement random rounding.

RNDSYM(arg)

RNDSYM implements a symmetric round to highest magnitude; this is equivalent to the MATLAB function `round(arg)`.

Table 4: **Illustration of RNDSYM Return Values**

arg	RNDSYM(arg)
<x.5	x
x.5	x+1
>x.5	x+1
>-x.5	-x
-x.5	-x-1
<-x.5	-x-1

The binary point of `arg` is defined in the same manner as for the RNDSIMPLE(arg) function.

A rounding constant with a binary value of 0.0111.... (or 0.499...) is added to `arg`along with a carryin, defined as the inverse sign bit of `arg`. The LSBs are removed by the process of reinterpreting the full precision output to the specified core output width. The LSBs remain on the accumulator. For multiply operations, the XNOR of the multiplier inputs is used instead of the inverse sign bit of arg, performing multiply rounding toward infinity.

When carryin is 1, this is equivalent to adding 0.00....01. This modifies the rounding constant to 0.100.. (or 0.5) and therefore the rounding direction.

RNDMACC(arg)

When performing symmetric rounding towards infinity of MACC and add accumulate operations, it is difficult to determine the sign of the output ahead of time, so the round might cost an extra clock cycle. This extra cycle can be eliminated by adding the C input rounding constant (typically a binary value of 0.0111...) on the very first cycle. 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. This implementation is a practical way to save a clock cycle. There is a rare chance that the final accumulate operation can flip the sign of the output from the previous accumulated value.

To perform this for a MACC operation, the following DSP Macro instructions are used:

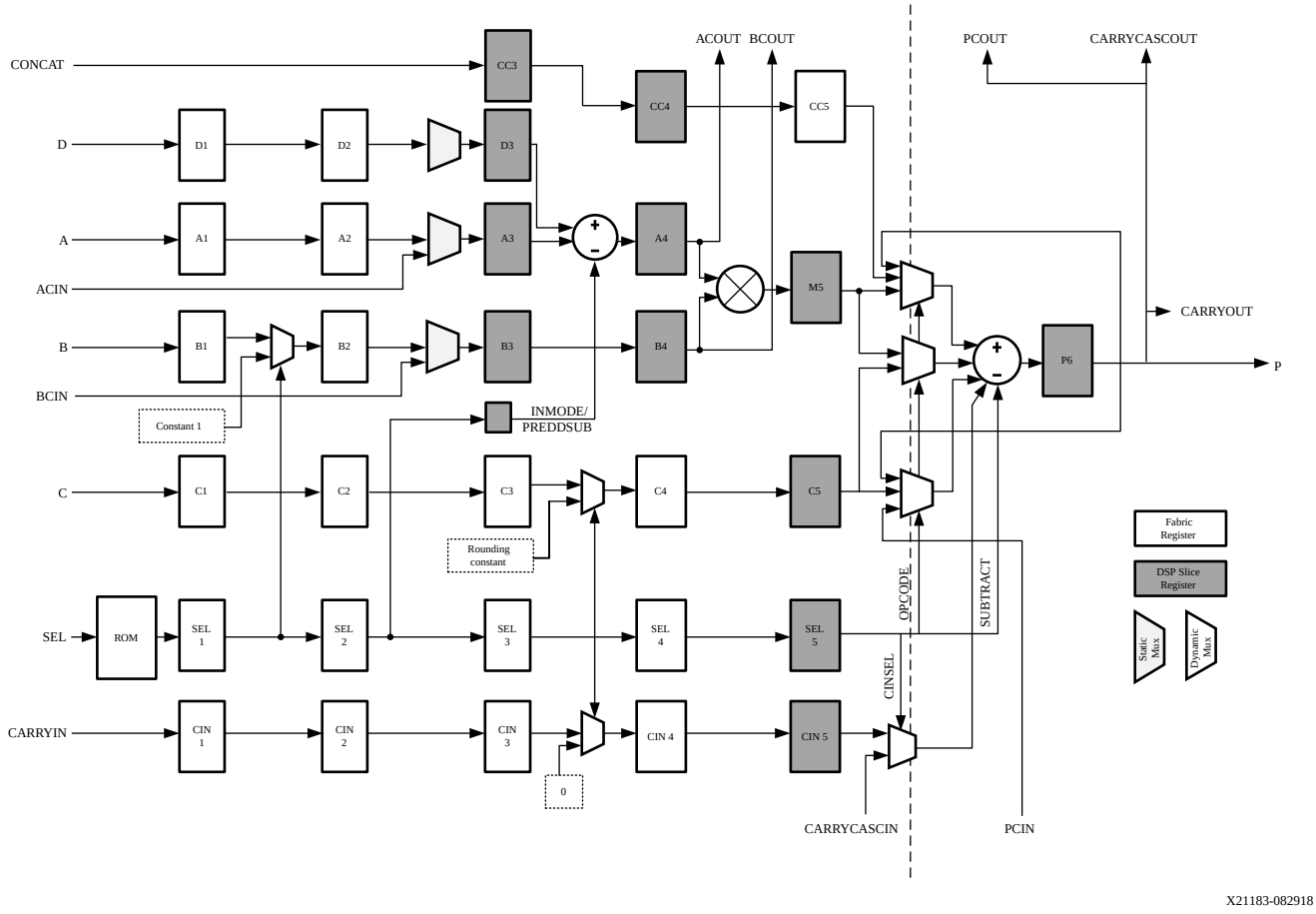
P = (A+D)*B+C (first cycle only)

P = (A+D)*B+P (middle cycles)

P = RNDMACC((A+D)*B+P) (last cycle only)

Detailed Pipeline Implementation

Figure 2: **DSP Macro Detailed Implementation**



The previous figure iillustrates the generalized DSP Macro implementation.

***Note*:**

- The second stage add/sub input mux implementations vary depending on the selected device.
- A static mux is resolved at core generation, whereas a dynamic mux is implemented in the generated core.

Users requiring a specific mapping to registers in the DSP48 or DSP58 primitive can use the following table to determine the corresponding DSP Macro registers. See the *Versal ACAP DSP Engine Architecture Manual* (AM004), *UltraScale Architecture DSP Slice User Guide* (UG579), or the *7 Series DSP48E1 Slice User Guide* (UG479).

Table 5: **Register Mapping for DSP Macro**

DSP Macro	Using Pre-adder[1]	No Pre-adder
D3	D	N/A
A3	A1	A1
A4	AD	A2
B3	B1	B1
B4	B2	B2

Notes:

1. The mapping of A4 changes depending on whether the pre-adder is used so the tiered latency model is maintained.

Design Flow Steps

This section describes customizing and generating the core, constraining the core, and the simulation, synthesis, and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

The DSP Macro core has three pages used to configure the core plus two informational tabs.

IP Symbol

The IP Symbol tab illustrates the core pinout.

Instruction Summary

The Instruction Summary tab displays the complete list of specified instructions and the SEL value that is required to select each instruction.

Instructions Page

This page is used to specify the instructions that the core is to implement.

- **Component Name:** The name of the core component to be instantiated. The name must begin with a letter and be composed of the following characters: a to z, 0 to 9, and "_".
- **Available instructions:** Informational parameter. When the Show Filtered check box is selected, the available instruction list dynamically updates to show the remaining valid instructions given the instruction that is currently being entered into the user interface. Instructions can be selected in the Available instruction panel and drag and dropped into an Instruction parameter.
- **Instructions 0 to 7:** Specifies the operations the core is to implement. Text entry of the desired arithmetic operation to be generated on the P output port. The left side of the expression, P=, is implicitly declared and should not be specified. Instructions are case insensitive.See Instruction Format for further details on the instruction format and supported operations.
- **Instructions 8 to 63:** Specifies instructions 8 to 63 using a comma-delimited list of instructions.

Related Information

Instruction Format

Pipeline Options Page

The pipeline depths of the various input paths are specified on this page.

- **Pipeline Options:** Specifies the pipeline method to be used: Automatic, By Tier and Expert.
- **Custom Pipeline options:** Specifies the pipeline depth of the various input paths.
  - **Tier 1 to 6:** When By Tier has been selected for Pipeline Options, these parameters are used to enable/disable the registers across all the input paths for a given pipeline stage. Some restrictions are enforced.When P has been specified in an expression, tier 6 is forced, as asynchronous feedback is not supported.

- **Individual registers:** When Expert has been selected for the Pipeline Options, these parameters are used to enable/disable individual register stages. Some restrictions are enforced.The P register is forced when P has been specified in an expression. Asynchronous feedback is not supported.

See Detailed Pipeline Implementation for further details on how the various pipeline stages relate to the core implementation.

- **Control Ports:**
  - **CE:** Global check box: enables a single CE pin for all registers in the core.D, A, B, CONCAT, C, M, P, SEL/CARRYIN check boxes: enable individual CE pins for all enabled registers in the core.
  - **SCLR:** Global check box: enables a single SCLR pin for all registers in the core.D, A, B, CONCAT, C, M, P, SEL/CARRYIN checkboxes: enable an individual SCLR pin for each datapath.

Related Information

Detailed Pipeline Implementation

Implementation Page

- **Input Port Properties:** Specifies the bit-width of the D, A, B, CONCAT and C input ports.
- **Output Port Properties:** Specifies the precision of the P output port; Full Precision and User Defined.

The Vivado® IDE automatically calculates the full precision output width given the width of the specified input ports. When P has been used as an operand, the full precision output width is set to the full DSP Slice width of 48 bits for 7 series and UltraScale™ devices or 58 bits for Versal™ devices. When Full Precision is selected, the output width is set to the full precision value. When User Defined is selected, the output width can be set to any value up to 48 bits and 58 bits for Versal™. When the specified value is less than the full precision width, the output is truncated, that is, the LSBs are removed. This option should be used when a rounding function has been specified.

- **Width:** Specifies the actual output width of the P output port. When specified to be less than Full Precision, the DSP Slice output is truncated.
- **Additional Ports:** Specifies if the core has a CARRYOUT output port or the ACOUT, BCOUT, PCOUT or CARRYCASCOUT cascaded output ports.
- **Use DSP Slice:** Specifies if the core implementation uses an DSP Slice or device logic equivalent. When a device logic implementation is specified, the core is unlikely to achieve the same F_{max} as DSP Slice.

Related Information

Port Descriptions

User Parameters

The following table shows the relationship between the fields in the Vivado® IDE and the user parameters (which can be viewed in the Tcl Console).

Table 6: **User Parameters**

Vivado IDE Parameter	User Parameter	Default Value
Show Filtered	show_filtered	False
Instructions: 0	instruction1	A*B+C
Instructions: 1	instruction2	#
Instructions: 2	instruction3	#
Instructions: 3	instruction4	#
Instructions: 4	instruction5	#
Instructions: 5	instruction6	#
Instructions: 6	instruction7	#
Instructions: 7	instruction8	#
Instructions: 8-63	instruction_list	#
Pipeline Options	pipeline_options	Automatic
Custom Pipeline options: Tier: 1	tier_1	False
Custom Pipeline options: Tier: 2	tier_2	False
Custom Pipeline options: Tier: 3	tier_3	False
Custom Pipeline options: Tier: 4	tier_4	False
Custom Pipeline options: Tier: 5	tier_5	False
Custom Pipeline options: Tier: 6	tier_6	False
Custom Pipeline options: Tier: D: 1	dreg_1	False
Custom Pipeline options: Tier: D: 2	dreg_2	False
Custom Pipeline options: Tier: D: 3	dreg_3	True
Custom Pipeline options: Tier: A: 1	areg_1	False
Custom Pipeline options: Tier: A: 2	areg_2	False
Custom Pipeline options: Tier: A: 3	areg_3	True
Custom Pipeline options: Tier: A: 4	areg_4	True
Custom Pipeline options: Tier: B: 1	breg_1	False
Custom Pipeline options: Tier: B: 2	breg_2	False
Custom Pipeline options: Tier: B: 3	breg_3	True
Custom Pipeline options: Tier: B: 4	breg_4	True
Custom Pipeline options: Tier: C: 1	creg_1	False
Custom Pipeline options: Tier: C: 2	creg_2	False
Custom Pipeline options: Tier: C: 3	creg_3	True

Table 6: **User Parameters** *(cont'd)*

Vivado IDE Parameter	User Parameter	Default Value
Custom Pipeline options: Tier: C: 4	creg_4	True
Custom Pipeline options: Tier: C: 5	creg_5	True
Custom Pipeline options: Tier: CONCAT: 3	concatreg_3	False
Custom Pipeline options: Tier: CONCAT: 4	concatreg_4	False
Custom Pipeline options: Tier: CONCAT: 5	concatreg_5	False
Custom Pipeline options: Tier: SEL: 1	opreg_1	False
Custom Pipeline options: Tier: SEL: 1	opreg_2	False
Custom Pipeline options: Tier: SEL: 1	opreg_3	False
Custom Pipeline options: Tier: SEL: 1	opreg_4	False
Custom Pipeline options: Tier: SEL: 1	opreg_5	False
Custom Pipeline options: Tier: CARRYIN: 1	cinreg_1	False
Custom Pipeline options: Tier: CARRYIN : 2	cinreg_2	False
Custom Pipeline options: Tier: CARRYIN : 3	cinreg_3	False
Custom Pipeline options: Tier: CARRYIN : 4	cinreg_4	False
Custom Pipeline options: Tier: CARRYIN : 5	cinreg_5	False
Custom Pipeline options: Tier: CARRYIN : 5	mreg_5	True
Custom Pipeline options: Tier: B: 5	preg_6	True
Input Port Properties: D	d_width	18
Input Port Properties: A	a_width	18
Input Port Properties: B	b_width	18
Input Port Properties: CONCAT	concat_width	48
Input Port Properties: C	c_width	48
Output Properties	output_properties	Full_Precision
Output Port Properties: Width	p_width	48
Use CARRYOUT	has_carryout	False
Use ACOUT	has_acout	False
Use BCOUT	has_bcout	False
Use Squaring	has_square	False
Use CARRYCASCOUT	has_carrycascout	False
Use PCOUT	has_pcout	False
Control Ports: CE : Global	has_ce	False
Control Ports: CE : D	has_d_ce	False
Control Ports: CE : A	has_a_ce	False
Control Ports: CE : B	has_b_ce	False
Control Ports: CE : C	has_c_ce	False
Control Ports: CE : CONCAT	has_concat_ce	False
Control Ports: CE : M	has_m_ce	False
Control Ports: CE : P	has_p_ce	False
Control Ports: CE : SEL/CARRYIN	has_sel_ce	False

Table 6: **User Parameters** *(cont'd)*

Vivado IDE Parameter	User Parameter	Default Value
Control Ports: SCLR : Global	has_sclr	False
Control Ports: SCLR : D	has_d_sclr	False
Control Ports: SCLR : A	has_a_sclr	False
Control Ports: SCLR : B	has_b_sclr	False
Control Ports: SCLR : C	has_c_sclr	False
Control Ports: SCLR : CONCAT	has_concat_sclr	False
Control Ports: SCLR : M	has_m_sclr	False
Control Ports: SCLR : P	has_p_sclr	False
Control Ports: SCLR : SEL/CARRYIN	has_sel_sclr	False

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

System Generator for DSP

The DSP Macro core is available through Xilinx® System Generator for DSP, a design tool that enables the use of the model-based design environment Simulink® for device design. The DSP Macro core is one of the DSP building blocks provided in the Xilinx blockset for Simulink®. The core can be found in the Xilinx Blockset in the DSP section. The block is called “DSP Macro v1.0.” See the *Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator* (UG897) for more information.

System Generator for DSP Graphical User Interface

This section describes each tab of the System Generator for DSP GUI and details the parameters that differ from the Vivado® IDE.

Tab 1: Instructions

The Instruction tab is used to define the operations that the core is to implement. Each instruction can be entered on a new line, or in a comma delimited list, and are enumerated from the top-down. A maximum of 64 instructions can be specified. See Instructions Page and Instruction Format for details on supported instructions and their format.

Related Information

Instructions Page
Instruction Format

Tab 2: Pipeline Options

The Pipeline Options tab is used to define the pipeline depth of the various input paths. See the Pipeline Options Page for details of all the core parameters on this tab.

Related Information

Pipeline Options Page

Tab 3: Implementation

The Implementation tab is used to define implementation options. See the Implementation Page for details of all the core parameters on this tab.

- **Output Port Properties:** Specifies the precision of the P output port: Full Precision and User Defined.

  The core automatically calculates the full precision output width and binary point position given the width and binary point of the specified input ports. When P has been used as an operand, the full precision output width is set to the full DSP Slice width of 48 bits for 7 series and UltraScale™ devices and 58 bits for Versal™ devices.

  When Full Precision is selected, the output is set to full precision width and binary point.

  When User Defined is selected, the output width can be set to any value up to 48 bits for 7 series and UltraScale Devices and 58 bits for Versal devices. The output formatting has two modes of operation.

  - Full precision binary point is calculated to be zero.

    When the output width is specified to be less than the full precision width, the output is truncated, that is, the LSBs are removed.

  - Full precision binary point is calculated to be greater than zero.

    The binary point is anchored. The output of the core behaves in the same manner as a System Generator Convert block with the following settings: **Quantization → Truncate** and **Overflow → Wrap**. Some restrictions on the binary point values are enforced; it cannot be greater than the full precision binary point value and its permitted minimum value will be modified to ensure that when the binary point value and output width are combined, the resulting MSB value does not exceed 48 bits for 7 series and UltraScale devices and 58 bits for Versal devices.

  - **Width:** Specifies the user-defined output width of the P output port.

  - **Binary Point:** Specifies the user-defined binary point of the P output port.
- **ce:** Selects either a global clock enable pin, or separate clock enable pins for each register. When separate clock enable pins are selected, these are managed within System Generator to correctly handle multirate constraints.
- **rst:** Selects either a global reset pin, or separate reset pins for each datapath. In a similar way to the separate clock enable pins, System Generator manages the situation where datapaths with separate reset controls have different rates.
- **FPGA Area Estimation:** See the System Generator for DSP documentation for detailed information about this section

Related Information

Implementation Page

Constraining the Core

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado® simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

XILINX

Appendix A

Upgrading

This appendix contains information about upgrading to this core from an older superseded IP core. For customers upgrading in the Vivado® Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Upgrading in the Vivado Design Suite

This section provides information about any changes to the currently used user logic or port designations that take place when you upgrade to the current version of this IP core in the Vivado® Design Suite from the superseded DSP48 Macro.

Parameter Changes

No change.

Port Changes

No change.

Functionality Changes

No change.

XILINX

Appendix B

Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT! *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Finding Help on Xilinx.com

To help in the design and debug process when using the core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The Xilinx Community Forums are also available where members can learn, participate, share, and ask questions about Xilinx solutions.

Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx® Documentation Navigator. Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Technical Support

Xilinx provides technical support on the Xilinx Community Forums for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To ask questions, navigate to the Xilinx Community Forums.

Debug Tools

There are many tools available to address DSP Macro design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx® devices.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

XILINX

Appendix C

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. *Vivado Design Suite User Guide: Designing with IP* (UG896)
2. *Versal ACAP DSP Engine Architecture Manual* (AM004)
3. *UltraScale Architecture DSP Slice User Guide* (UG579)
4. *7 Series DSP48E1 Slice User Guide* (UG479)
5. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator* (UG897)
8. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
9. *ISE to Vivado Design Suite Migration Guide* (UG911)
10. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
01/25/2021 Version 1.0	
Initial release	N/A

Please Read: Important Legal Notices

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