

XtremeDSP Design Considerations

This chapter provides technical details for the XtremeDSP™ Digital Signal Processing (DSP) element, the DSP48 slice.

The DSP48 slice is a new element in the Xilinx development model referred to as “Application Specific Modular Blocks” (ASMBL). The purpose of this model is to deliver off-the-shelf programmable devices with the best mix of logic, memory, I/O, processors, clock management, and digital signal processing. ASMBL is an efficient FPGA development model for delivering off-the-shelf, flexible solutions ideally suited to different application domains.

Each XtremeDSP tile contains two DSP48 slices to form the basis of a versatile coarse-grain DSP architecture. Many DSP designs follow a multiply with addition. In Virtex™-4 devices these elements are supported in dedicated circuits.

The DSP48 slices support many independent functions, including multiplier, multiplier-accumulator (MAC), multiplier followed by an adder, three-input adder, barrel shifter, wide bus multiplexers, magnitude comparator, or wide counter. The architecture also supports connecting multiple DSP48 slices to form wide math functions, DSP filters, and complex arithmetic without the use of general FPGA fabric.

The DSP48 slices available in all Virtex-4 family members support new DSP algorithms and higher levels of DSP integration than previously available in FPGAs. Minimal use of general FPGA fabric leads to low power, very high performance, and efficient silicon utilization.

Introduction

The DSP48 slices facilitate higher levels of DSP integration than previously possible in FPGAs. Many DSP algorithms are supported with minimal use of the general-purpose FPGA fabric, resulting in low power, high performance, and efficient device utilization.

At first look, the DSP48 slice is an 18 x 18 bit two’s complement multiplier followed by a 48-bit sign-extended adder/subtractor/accumulator, a function that is widely used in digital signal processing (DSP).

A second look reveals many subtle features that enhance the usefulness, versatility, and speed of this arithmetic building block.

Programmable pipelining of input operands, intermediate products, and accumulator outputs enhances throughput. The 48-bit internal bus allows for practically unlimited aggregation of DSP slices.

One of the most important features is the ability to cascade a result from one XtremeDSP Slice to the next without the use of general fabric routing. This path provides high-performance and low-power post addition for many DSP filter functions of any tap length.

For multi-precision arithmetic this path supports a right-wire-shift. Thus a partial product from one XtremeDSP Slice can be right-justified and added to the next partial product computed in an adjacent such slice. Using this technique, the XtremeDSP Slices can be configured to support any size operands.

Another key feature for filter composition is the ability to cascade an input stream from slice to slice.

The C input port, allows the formation of many 3-input mathematical functions, such as 3-input addition, 2-input multiplication with a single addition. One subset of this function is the very valuable support of rounding a multiplication “away from zero”.

Architecture Highlights

The Virtex-4 DSP slices are organized as vertical DSP columns. Within the DSP column, two vertical DSP slices are combined with extra logic and routing to form a DSP tile. The DSP tile is four CLBs tall.

Each DSP48 slice has a two-input multiplier followed by multiplexers and a three-input adder/subtractor. The multiplier accepts two 18-bit, two's complement operands producing a 36-bit, two's complement result. The result is sign extended to 48 bits and can optionally be fed to the adder/subtractor. The adder/subtractor accepts three 48-bit, two's complement operands, and produces a 48-bit two's complement result.

Higher level DSP functions are supported by cascading individual DSP48 slices in a DSP48 column. One input (cascade B input bus) and the DSP48 slice output (cascade P output bus) provide the cascade capability. For example, a Finite Impulse Response (FIR) filter design can use the cascading input to arrange a series of input data samples and the cascading output to arrange a series of partial output results. For details on this technique, refer to the section titled “Adder Cascade vs. Adder Tree,” page 31.

Architecture highlights of the DSP48 slices are:

- 18-bit by 18-bit, two's-complement multiplier with a full-precision 36-bit result, sign extended to 48 bits
- Three-input, flexible 48-bit adder/subtractor with optional registered accumulation feedback
- Dynamic user-controlled operating modes to adapt DSP48 slice functions from clock cycle to clock cycle
- Cascading 18-bit B bus, supporting input sample propagation
- Cascading 48-bit P bus, supporting output propagation of partial results
- Multi-precision multiplier and arithmetic support with 17-bit operand right shift to align wide multiplier partial products (parallel or sequential multiplication)
- Symmetric intelligent rounding support for greater computational accuracy
- Performance enhancing pipeline options for control and data signals are selectable by configuration bits
- Input port “C” typically used for multiply-add operation, large three-operand addition, or flexible rounding mode
- Separate reset and clock enable for control and data registers

- I/O registers, ensuring maximum clock performance and highest possible sample rates with no area cost
- OPMODE multiplexers

A number of software tools support the DSP48 slice. The Xilinx ISE software supports DSP48 slice instantiations. The Architecture Wizard is a GUI for creating instantiation VHDL and/or Verilog code. It also helps generate code for designs using a single DSP48 slice (i.e., Multiplier, Adder, Multiply-Accumulate or MAC, and Dynamic Control modes). Using the Architecture Wizard, CORE Generator™ tool, or System Generator, a designer can quickly generate math or other functions using Virtex-4 DSP48 slices.

Number of DSP48 Slices Per Virtex-4 Device

Table 2-1 shows the number of DSP48 slices for each device in the Virtex-4 families. The Virtex-4 SX family offers the highest ratio of DSP48 slices to logic, making it ideal for math-intensive applications.

Table 2-1: Number of DSP48 Slices per Family Member

Device	DSP48	Device	DSP48	Device	DSP48
XC4VLX15	32			XC4VFX12	32
XC4VLX25	48	XC4VSX25	128	XC4VFX20	32
		XC4VSX35	192		
XC4VLX40	64			XC4VFX40	48
XC4VLX60	64	XC4VSX55	512	XC4VFX60	128
XC4VLX80	80				
XC4VLX100	96			XC4VFX100	160
XC4VLX160	96				
XC4VLX200	96			XC4VFX140	192

DSP48 Slice Primitive

Figure 2-1 shows the DSP48 slice primitive.

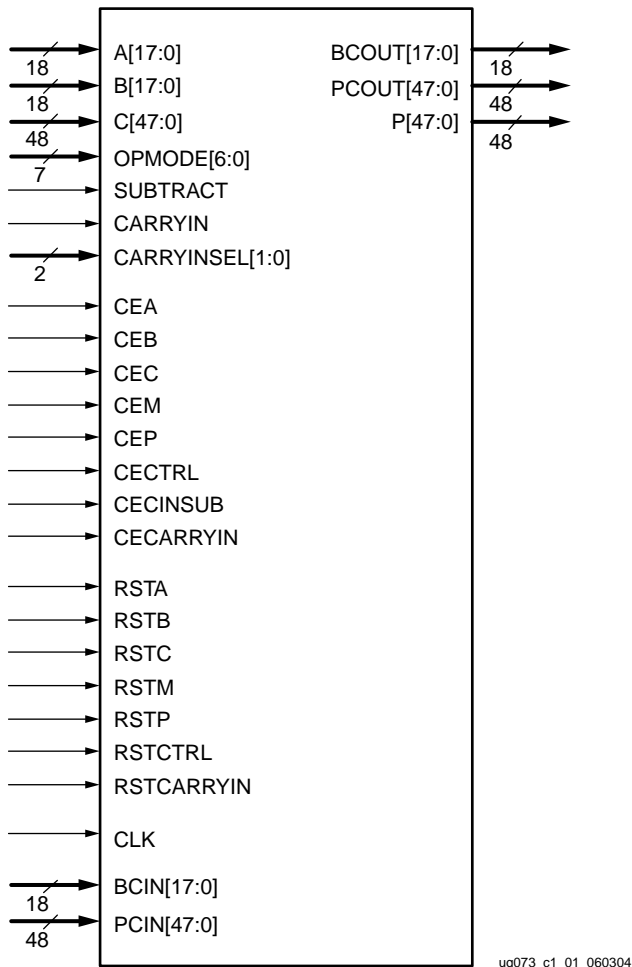


Figure 2-1: DSP48 Slice Primitive

Table 2-2 lists the available ports in the DSP48 slice primitive.

Table 2-2: DSP48 Slice Port List and Definitions

Signal Name	Direction	Size	Function
A	I	18	The multiplier's A input. This signal can also be used as the adder's Most Significant Word (MSW) input
B	I	18	The multiplier's B input. This signal can also be used as the adder's Least Significant Word (LSW) input
C	I	48	The adder's C input

Table 2-2: DSP48 Slice Port List and Definitions (*Continued*)

Signal Name	Direction	Size	Function
OPMODE	I	7	Controls the input to the X, Y, and Z multiplexers in the DSP48 slices (see OPMODE, Table 2-7)
SUBTRACT	I	1	0 = add, 1 = subtract
CARRYIN	I	1	The carry input to the carry select logic
CARRYINSEL	I	2	Selects carry source (see CARRYINSEL, Table 2-8)
CEA	I	1	Clock enable: 0 = hold, 1 = enable AREG
CEB	I	1	Clock enable: 0 = hold, 1 = enable BREG
CEC	I	1	Clock enable: 0 = hold, 1 = enable CREG
CEM	I	1	Clock enable: 0 = hold, 1 = enable MREG
CEP	I	1	Clock enable: 0 = hold, 1 = enable PREG
CECTRL	I	1	Clock enable: 0 = hold, 1 = enable OPMODEREG, CARRYINSELREG
CECINSUB	I	1	Clock enable: 0 = hold, 1 = enable SUBTRACTREG and general interconnect carry input
CECARRYIN	I	1	Clock enable: 0 = hold, 1 = enable (carry input from internal paths)
RSTA	I	1	Reset: 0 = no reset, 1 = reset AREG
RSTB	I	1	Reset: 0 = no reset, 1 = reset BREG
RSTC	I	1	Reset: 0 = no reset, 1 = reset CREG
RSTM	I	1	Reset: 0 = no reset, 1 = reset MREG
RSTP	I	1	Reset: 0 = no reset, 1 = reset PREG
RSTCTRL	I	1	Reset: 0 = no reset, 1 = reset SUBTRACTREG, OPMODEREG, CARRYINSELREG
RSTCARRYIN	I	1	Reset: 0 = no reset, 1 = reset (carry input from general interconnect and internal paths)
CLK	I	1	The DSP48 clock
BCIN	I	18	The multiplier's cascaded B input. This signal can also be used as the adder's LSW input
PCIN	I	48	Cascaded adder's Z input from the previous DSP slice
BCOUT	O	18	The B cascade output
PCOUT	O	48	The P cascade output
P	O	48	The product output

DSP48 Slice Attributes

The synthesis attributes for the DSP48 slice are described in detail throughout this section. With the exception of the `B_INPUT` and `LEGACY_MODE` attributes, all other attributes call out pipeline registers in the control and datapaths. The value of the attribute sets the number of pipeline registers.

The attribute settings are as follows:

- The `AREG` and `BREG` attributes can take a value of 0, 1, or 2. The values define the number of pipeline registers in the A and B input paths. See the “A, B, C, and P Port Logic” section for more information.
- The `CREG`, `MREG`, and `PREG` attributes can take a value of 0 or 1. The value defines the number of pipeline registers at the output of the multiplier (`MREG`) (shown in Figure 2-11) and at the output of the adder (`PREG`) (shown in Figure 2-9). The `CREG` attribute is used to select the pipeline register at the 'C' input (shown in Figure 2-8).
- The `CARRYINREG`, `CARRYINSELREG`, `OPMODEREG`, and `SUBTRACTREG` attributes take a value of 0 if there is no pipelining register on these paths, and take a value of 1 if there is one pipeline register in their path. The `CARRYINSELREG`, `OPMODEREG`, and `SUBTRACTREG` paths are shown in Figure 2-10, and the `CARRYINREG` path is shown in Figure 2-12.
- The `B_INPUT` attribute defines whether the input to the B port is routed from the parallel input (attribute: `DIRECT`) or the cascaded input from the previous slice (attribute: `CASCADE`).
- The `LEGACY_MODE` attribute serves two purposes. The first purpose is similar in nature to the `MREG` attribute. It defines whether or not the multiplier is "flow through" in nature (i.e., `LEGACY_MODE` value equal to `MULT18x18`) or contains a single pipeline register in the middle of the multiplier (i.e., `LEGACY_MODE` value equal to `MULT18x18S` is the same as `MREG` value equal to one). While this is redundant to the `MREG` attribute, it was deemed useful for customers used to the Virtex-II and Virtex-II Pro multipliers since the DSP48 setup and hold timing most closely matches those of the Virtex-II and Virtex-II Pro `MULT18x18S` when the `MREG` is used. Any disagreement between the `MREG` attribute and `LEGACY_MODE` attribute settings are flagged as a software Design Rule Check (DRC) error. The second purpose for the attribute is to convey to the timing tools whether the A and B port through the combinatorial multiplier path (slower timing) or faster X multiplexer bypass path for A:B should be used in the timing calculations. Since the `OPMODE` can change dynamically, the timing tools cannot determine this without an attribute.

To summarize the timing tools behavior:

- ◆ If (attribute: `NONE`), then timing analysis/simulation bypasses the multiplier for the highest performance. The lowest power dissipation is achieved by setting `MREG` to one while CEM input is grounded.
- ◆ If (attribute: `MULT18x18`), then timing analysis/simulation uses the combinatorial path through the multiplier. In this case, `MREG` must be set to zero or a DRC error occurs.
- ◆ If (attribute: `MULT18x18S`), then timing analysis/simulation uses a pipelined multiplier. In this case `MREG` must be set to one or a DRC error occurs.

Attributes in VHDL

```

DSP48 generic map (
  AREG => 1, -- Number of pipeline registers on the A input, 0, 1 or 2
  BREG => 1, -- Number of pipeline registers on the B input, 0, 1 or 2
  B_INPUT => "DIRECT", -- B input DIRECT from fabric or CASCADE from
                        -- another DSP48
  CARRYINREG => 1,      -- Number of pipeline registers for the CARRYIN
                        -- input, 0 or 1
  CARRYINSELREG => 1,  -- Number of pipeline registers for the
                        -- CARRYINSEL, 0 or 1
  CREG => 1, -- Number of pipeline registers on the C input, 0 or 1
  LEGACY_MODE => "MULT18X18S", -- Backward compatibility, NONE,
                                -- MULT18X18 or MULT18X18S
  MREG => 1,      -- Number of multiplier pipeline registers, 0 or 1
  OPMODEREG => 1, -- Number of pipeline registers on OPMODE input,
                        -- 0 or 1
  PREG => 1, -- Number of pipeline registers on the P output, 0 or 1
  SIM_X_INPUT => "GENERATE_X_ONLY",
                -- Simulation parameter for behavior for X on input.
                -- Possible values: GENERATE_X, NONE or WARNING
  SUBTRACTREG => 1) -- Number of pipeline registers on the SUBTRACT
                    -- input, 0 or 1

```

Attributes in Verilog

```

defparam DSP48_inst.AREG = 1;
// Number of pipeline registers on the A input, 0, 1 or 2
defparam DSP48_inst.BREG = 1;
// Number of pipeline registers on the B input, 0, 1 or 2
defparam DSP48_inst.B_INPUT = "DIRECT";
// B input DIRECT from fabric or CASCADE from another DSP48
defparam DSP48_inst.CARRYINREG = 1;
// Number of pipeline registers for the CARRYIN input, 0 or 1
defparam DSP48_inst.CARRYINSELREG = 1;
// Number of pipeline registers for the CARRYINSEL, 0 or 1
defparam DSP48_inst.CREG = 1;
// Number of pipeline registers on the C input, 0 or 1
defparam DSP48_inst.LEGACY_MODE = "MULT18X18S";
// Backward compatibility, NONE, MULT18X18 or MULT18X18S
defparam DSP48_inst.MREG = 1;
// Number of multiplier pipeline registers, 0 or 1
defparam DSP48_inst.OPMODEREG = 1;
// Number of pipeline registers on OPMODE input, 0 or 1
defparam DSP48_inst.PREG = 1;
// Number of pipeline registers on the P output, 0 or 1
defparam DSP48_inst.SIM_X_INPUT = "GENERATE_X_ONLY";
// Simulation parameter for behavior for X on input.
// Possible values: GENERATE_X, NONE or WARNING
defparam DSP48_inst.SUBTRACTREG = 1;
// Number of pipeline registers on the SUBTRACT input, 0 or 1

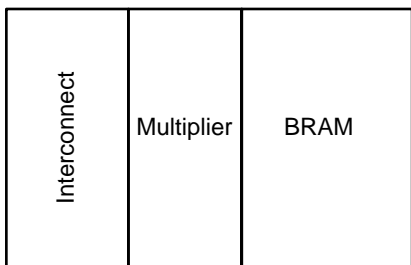
```

DSP48 Tile and Interconnect

Two DSP48 slices, a shared 48-bit C bus, and dedicated interconnect form a DSP48 tile. The DSP48 tiles stack vertically in a DSP48 column. The height of a DSP48 tile is the same as four CLBs and also matches the height of one block RAM. This “regularity” enhances the routing of wide datapaths. Smaller Virtex-4 family members have one DSP48 column while the larger Virtex-4 family members have two, four, or eight DSP48 columns.

As shown in Figure 2-2, the multipliers and block RAM share interconnect resources in the Virtex-II and Virtex-II Pro architectures. Virtex-4 devices, however, have independent routing for the DSP48 tiles and block RAM, effectively doubling the available data bandwidth between the elements.

Virtex-II and Virtex-II Pro Devices



Virtex-4 Devices

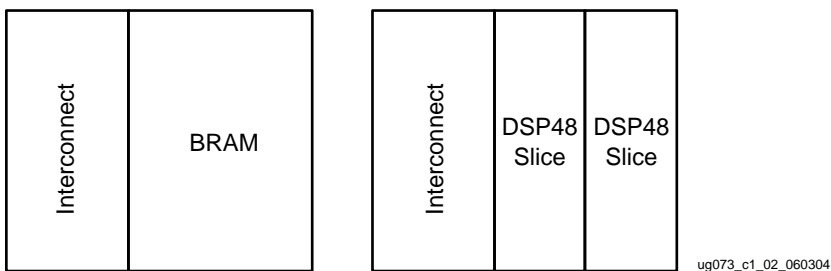


Figure 2-2: DSP48 Interconnect and Relative Dedicated Element Sizes

Figure 2-3 shows two DSP48 slices and their associated datapaths stacked vertically in a DSP48 column. The inputs to the shaded multiplexers are selected by configuration control signals. These are set by attributes in the HDL source code or by the User Constraint File (UCF).

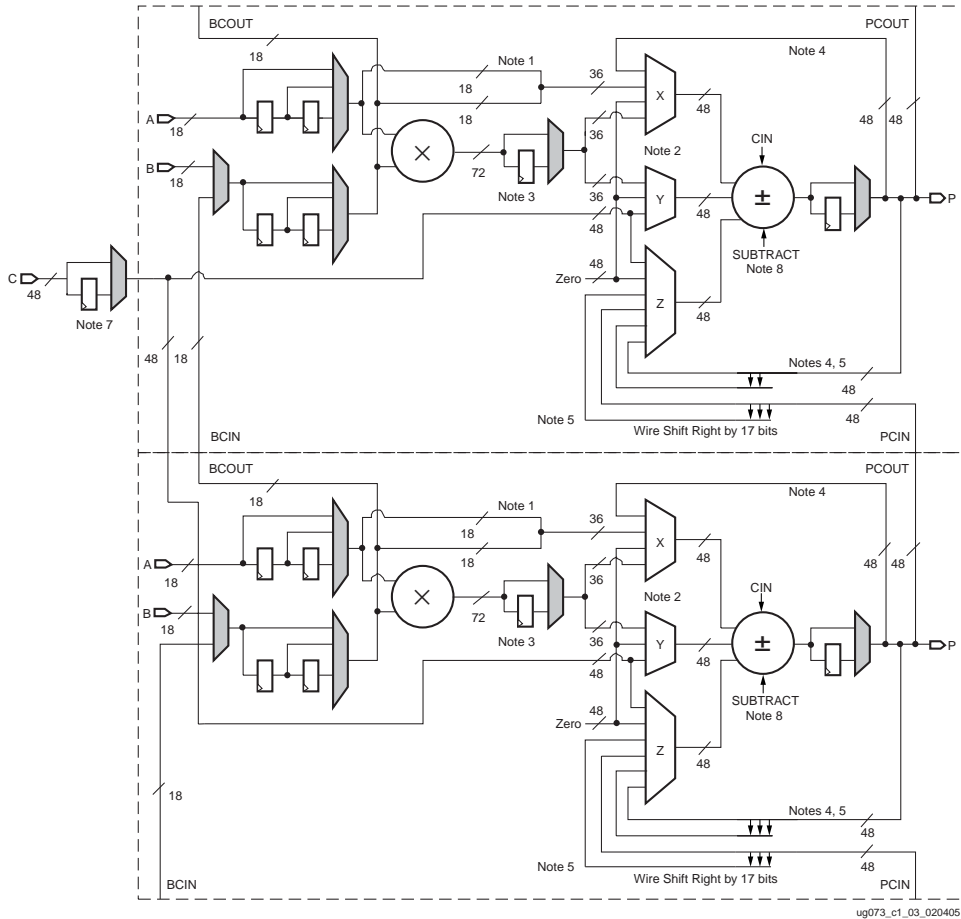


Figure 2-3: A DSP48 Tile Consisting of Two DSP48 Slices

Notes:

1. The 18-bit A bus and B bus are concatenated, with the A bus being the most significant.
2. The X, Y, and Z multiplexers are 48-bit designs. Selecting any of the 36-bit inputs provides a 48-bit sign-extended output.
3. The multiplier outputs two 36-bit partial products, sign extended to 48 bits. The partial products feed the X and Y multiplexers. When OPMODE selects the multiplier, both X and Y multiplexers are utilized and the adder/subtractor combines the partial products into a valid multiplier result.
4. The multiply-accumulate path for P is through the Z multiplexer. The P feedback through the X multiplexer enables accumulation of P cascade when the multiplier is not used.
5. The “Right Wire Shift by 17 bits” path truncates the lower 17 bits and sign extends the upper 17 bits.
6. The grey-colored multiplexers are programmed at configuration time.
7. The shared C register supports multiply-add, wide addition, or rounding.
8. Enabling SUBTRACT implements $Z - (X + Y + CIN)$ at the output of the adder/subtractor.

Simplified DSP48 Slice Operation

The math portion of the DSP48 slice consists of an 18-bit by 18-bit, two’s complement multiplier followed by three 48-bit datapath multiplexers (with outputs X, Y, and Z) followed by a three-input, 48-bit adder/subtractor.

The data and control inputs to the DSP48 slice feed the arithmetic portions directly, or are optionally registered one or two times to assist the construction of different, highly pipelined, DSP application solutions. The data inputs A and B can be registered once or twice. The other data inputs and the control inputs can be registered once. Full speed operation is 500 MHz when using the pipeline registers. More detailed timing information is available in the Timing Section.

In its most basic form the output of the adder/subtractor is a function of its inputs. The inputs are driven by the upstream multiplexers, carry select logic, and multiplier array. Equation 2-1 summarizes the combination of X, Y, Z, and CIN by the adder/subtractor. The CIN, 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.

$$\text{Adder Out} = (Z \pm (X + Y + \text{CIN})) \tag{Equation 2-1}$$

Equation 2-2 describes a typical use where A and B are multiplied and the result is added to or subtracted from the C register. More detailed operations based on control and data inputs are described in later sections. Selecting the multiplier function consumes both X and Y multiplexer outputs to feed the adder. The two 36-bit partial products from the multiplier are sign extended to 48 bits before being sent to the adder/subtractor.

$$\text{Adder Out} = C \pm (A \times B + \text{CIN}) \tag{Equation 2-2}$$

Figure 2-4 shows the DSP48 slice in a very simplified form. The seven OPMODE bits control the selection of the 48-bit datapaths by the three multiplexers feeding each of the three inputs to the adder/subtractor. In all cases, the 36-bit input data to the multiplexers is sign extended, forming 48-bit input datapaths to the adder/subtractor. Based on 36-bit operands and a 48-bit accumulator output, the number of “guard bits” (i.e., bits available to guard against overflow) is 12. Therefore, the number of multiply accumulations possible before overflow occurs is 4096. Combinations of OPMODE, SUBTRACT, CARRYINSEL, and CIN control the function of the adder/subtractor.

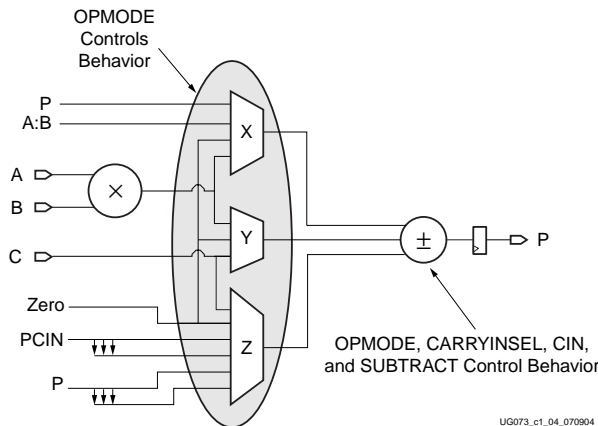


Figure 2-4: Simplified DSP48 Slice Model

Timing Model

Table 2-3 lists the XtremeDSP switching characteristics.

Table 2-3: XtremeDSP Switching Characteristics

Symbol	Description	Function	Control Signal
Setup and Hold of CE Pins			
$T_{DSPCCK_CE}/T_{DSPCKC_CE}$	Setup/Hold of all CE inputs of the DSP48 slice	Clock Enable	CE
$T_{DSPCCK_RST}/T_{DSPCKC_RST}$	Setup/Hold of all RST inputs of the DSP48 slice	Reset	RST
Setup and Hold Times of Data/Control Pins			
$T_{DSPDCK_AA, BB, CC}/T_{DSPCKD_AA, BB, CC}$	Setup/Hold of {A, B, C} input to {A, B, C} register	Data In	A, B, C
$T_{DSPDCK_AM, BM}/T_{DSPCKD_AM, BM}$	Setup/Hold of {A, B} input to M register	Data In	A, B
$T_{DSPDCK_AP, BP}_L/T_{DSPCKD_AP, BP}_L$	Setup/Hold of {A, B} input to P register (LEGACY_MODE = MULT18X18)	Data In	A, B
$T_{DSPDCK_AP_NL, BP_NL, CP}/T_{DSPCKD_AP_NL, BP_NL, CP}$	Setup/Hold of {A, B, C} input to P register (LEGACY_MODE = NONE for A and B)	Data In	A, B, C
$T_{DSPDCK_CRYINC, CRYINSC, OPO, SUBS}/T_{DSPCKD_CRYINC, CRYINSC, OPO, SUBS}$	Setup/Hold of {CARRYIN, CARRYINSEL, OPMODE, SUBTRACT} input to {CARRYIN, CARRYINSEL, OPMODE, SUBTRACT} register	Control In	Various
$T_{DSPDCK_CRYINP, CRYINSP, OPP, SUBPPCINP}/T_{DSPCKD_CRYINP, CRYINSP, OPP, SUBP, PCINP}$	Setup/Hold of {CARRYIN, CARRYINSEL, OPMODE, SUBTRACT, PCIN} input to P register	Control In	Various
Clock to Out			
T_{DSPCKO_PP}	Clock to out from P register to P output	Data Out	P Output
$T_{DSPCKO_PA, PB}_L$	Clock to out from {A, B} register to P output (LEGACY_MODE = MULT18X18)	Data Out	P Output
$T_{DSPCKO_PA_NL, PB_NL, PC}$	Clock to out from {A, B, C} register to P output (LEGACY_MODE = NONE for A and B)	Data Out	P Output
$T_{DSPCKO_PM, PCRYIN, PCRYINS, POP, PSUB}$	Clock to out from {M, CARRYIN, CARRYINSEL, OPMODE, SUBTRACT} register to P output	Data Out	P Output

Table 2-3: XtremeDSP Switching Characteristics (*Continued*)

Symbol	Description	Function	Control Signal
T _{DSPCKO_PCOU TP}	Clock to out from P register to PCOUT output	Data Out	P Output
T _{DSPCKO_{PCOUTA, PCOUTB}_L}	Clock to out from {A, B} register to PCOUT output (LEGACY_MODE = MULT18X18)	Data Out	P Output
T _{DSPCKO_{PCOUTA_NL, PCOUTB_NL, PCOUTC}}	Clock to out from {A, B, C} register to PCOUT output (LEGACY_MODE = NONE for A and B)	Data Out	P Output
T _{DSPCKO_{PCOUTM, PCOUTCRYIN, PCOUTCRYINS, PCOUTOP, PCOUTSUB}}	Clock to out from {M, CARRYIN, CARRYINSEL, OPMODE, SUBTRACT} register to PCOUT output	Data Out	P Output
Combinatorial			
T _{DSPDO_{AP, BP}_L}	{A, B} input to P output (LEGACY_MODE = MULT18X18)	Data In to Out	A, B to P
T _{DSPDO_{AP_NL, BP_NL, CP}}	{A, B, C} input to P output (LEGACY_MODE = NONE for A and B)	Data In to Out	A, B, C to P
T _{DSPDO_{CRYINP, CRYINSP, OPMODEP, SUBTRACTP, PCINP}}	{CARRYIN, CARRYINSEL, OPMODE, SUBTRACT, PCIN} input to P output	Control to Data Out	Various
T _{DSPDO_{APCOUT, BPCOUT}_L}	{A, B} input to PCOUT output (LEGACY_MODE = MULT18X18)	Data In to PC Out	A, B to PC Out
T _{DSPDO_{APCOUT_NL, BPCOUT_NL, CPCOUT}}	{A, B, C} input to PCOUT output (LEGACY_MODE = NONE for A and B)	Data In to PC Out	A, B, C to PC Out
T _{DSPDO_{CRYINPCOUT, CRYINSPCOUT, OPMODEPCOUT, SUBTRACTPCOUT, PCINPCOUT}}	{CARRYIN, CARRYINSEL, OPMODE, SUBTRACT, PCIN} input to PCOUT output	Control to PC Out	Various
Sequential			
T _{DSPCKCK_{AP, BP}_L}	From {A, B} register to P register (LEGACY_MODE = MULT18X18)	Register to register	–
T _{DSPCKCK_{AP_NL, BP_NL, CP, PP}}	From {A, B, C, P} register to P register (LEGACY_MODE = NONE for A and B)	Register to register	–

Table 2-3: XtremeDSP Switching Characteristics (*Continued*)

Symbol	Description	Function	Control Signal
$T_{DSPCKCK_}\{CRYINP, CRYINSP, OPMODEP, SUBTRACTP\}$	From {CARRYIN, CARRYINSEL, OPMODE, SUBTRACT} register to P register	Register to register	–
$T_{DSPCKCK_}\{AM, BM\}$	From {A, B} register to M register	Register to register	–

The timing diagram in Figure 2-5 uses OPMODE equal to 0x05 with all pipeline registers turned on. For other applications, the clock latencies and the parameter names must be adjusted.

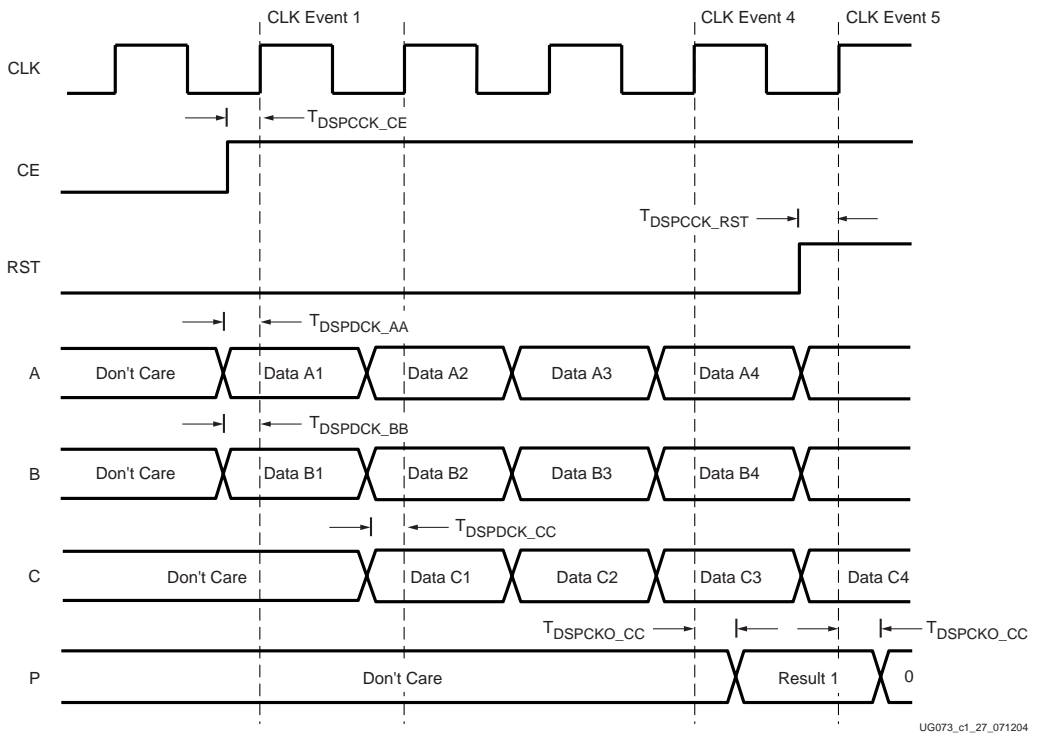


Figure 2-5: XtremeDSP Timing Diagram

The following events occur in Figure 2-5:

1. At time $T_{DSPCKCK_CE}$ before CLK event 1, CE becomes valid High to allow all DSP registers to sample incoming data.
2. At time $T_{DSPDCK_}\{AA, BB, CC\}$ before CLK event 1, data inputs A, B, C have remained stable for sampling into the DSP slice.
3. At time T_{DSPCKO_PP} after CLK event 4, the P output switches into the results of the data captured at CLK event 1. This occurs three clock cycles after CLK event 1.

4. At time $T_{\text{DSPCKK_RST}}$ before CLK event 5, the RST signal becomes valid High to allow a synchronous reset at CLK event 5.
5. At time $T_{\text{DSPCKO_PP}}$ after CLK event 5, the output P becomes a logic 0.

A, B, C, and P Port Logic

The DSP48 slice input and output data ports support many common DSP and math algorithms. The DSP48 slice has two direct 18-bit input data ports labeled A and B. Two DSP48 slices within a DSP48 tile share a direct 48-bit input data port labeled C. Each DSP48 slice has one direct 48-bit output port labeled P, a cascaded input datapath (B cascade), and a cascaded output datapath (P cascade), providing a cascaded input and output stream between adjacent DSP48 slices. Applications benefiting from this feature include FIR filters, complex multiplication, multi-precision multiplication, complex MACs, adder cascade, and adder tree (the final summation of several multiplier outputs) support.

The 18-bit A and B port can supply input data to the 18-bit by 18-bit, two's complement multiplier. A and B concatenated can bypass the multiplier and feed the X multiplexer input. The 48-bit C port is used as a general input to the Y and Z multiplexer to perform multiply, add, subtract, three-input add/subtract functions, or rounding.

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 (i.e., higher performance) vs. data latency. There is also a configuration controlled pipeline register between the multiplier and adder/subtractor known as the M register. The registers have independent clock enables and resets as described in Table 2-2 and shown in Figure 2-1.

The configuration bit enables the C register to select between two potentially different clock domains as shown in Figure 2-8, page 19. The selection of the clock multiplexer is not set by user attributes. If the C register is used, the DSP48 slices packed in the same DSP48 tile must either be in the same clock domain or meet multicycle clock constraints.

The shared “C” input within the DSP tile can be used by the two slices within a tile in any one of the following modes:

1. Neither DSP48 slice uses the C port.
The C inputs in both the slices are connected to GND, “0” in the HDL code. The place and route software maps the two slices in one tile.
2. Both DSP48 slices use the same C port inputs.
The C inputs in both the slices are connected to “C” in the HDL code. The place and route software maps the two slices in one tile.
3. Only one DSP48 slice uses the C port.
In this case, the C input on slice 1 is connected to “C”, and the C input on slice 2 is connected to “0” in the HDL code. A C port connected to “0” is taken as an unused C port in the software. The software can map the two slices in one tile. The simulation shows the C input connected to “0” for slice 2 in the code. However, in the hardware, the C port on slice 2 is connected to the C port on slice 1, causing a potential simulation mismatch for the C port on slice 2. To avoid this potential mismatch, the C port must not be selected on the Y and Z multiplexers of slice 2. To get a “0” at the output of multiplexers Y and Z, choose the “0” input of these multiplexers using OPMODE. Do not use the “C” input to get a zero at the output of Y and Z multiplexers on slice 2.

The A, B, C, and P port logics are shown in Figure 2-6, Figure 2-7, Figure 2-8, and Figure 2-9, respectively.

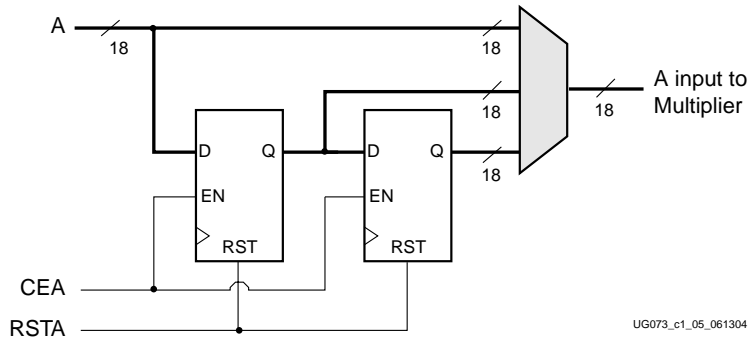


Figure 2-6: A Input Logic

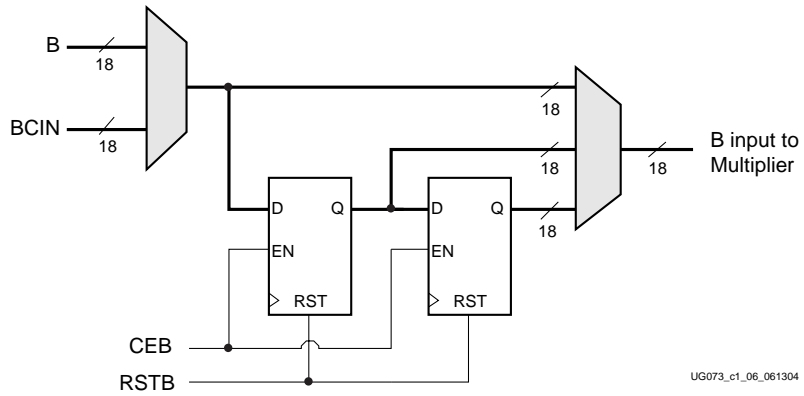


Figure 2-7: B Input Logic

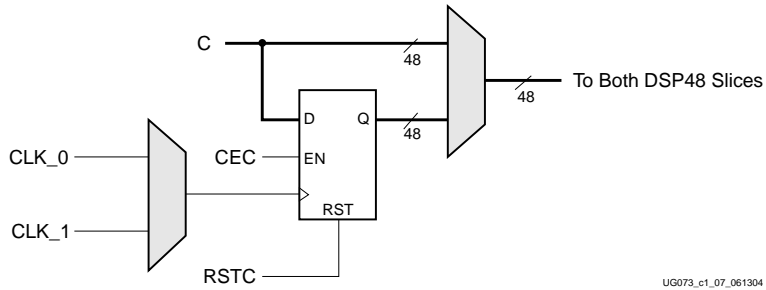


Figure 2-8: C Input Logic

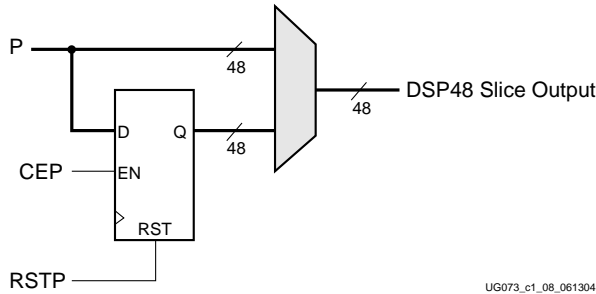


Figure 2-9: P Output Logic

OPMODE, SUBTRACT, and CARRYINSEL Port Logic

The OPMODE, SUBTRACT, and CARRYINSEL port logic supports flowthrough or registered input control signals. Similar to the datapaths, multiplexers controlled by configuration bits select flowthrough or optional registers. The control port registers allow users to trade off increased clock frequency (i.e., higher performance) vs. data latency.

The registers have independent clock enables and resets as described in Table 2-2 and shown in Figure 2-1. The OPMODE, SUBTRACT, and CARRYINSEL registers are reset by RSTCTRL. The SUBTRACT register has a separate enable labeled CECINSUB from OPMODE and CARRYINSEL. This enable signal is also used to enable the carry input from the general interconnect described in the “Carry Input Logic” subsection.

Figure 2-10 shows the OPMODE, SUBTRACT, and CARRYINSEL port logic.

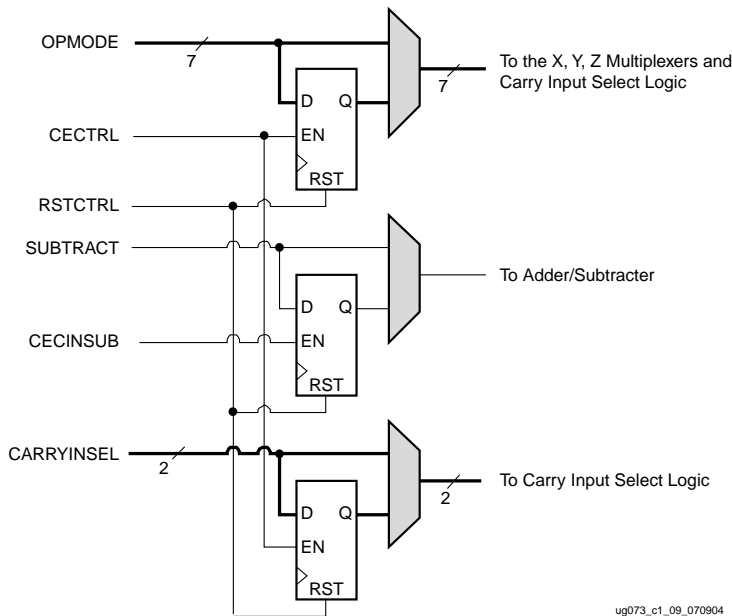


Figure 2-10: OPMODE, SUBTRACT, and CARRYINSEL Port Logic

Two's Complement Multiplier

The two's complement multiplier inside the DSP48 slice accepts two 18-bit x 18-bit two's complement inputs and produces a 36-bit two's complement result. Cascading of multipliers to achieve larger products is supported with a 17-bit right-shifted cascaded bus input to the adder/subtractor to “right justify” partial products by the correct number of bits. MAC functions can also “right justify” intermediate results for multi-precision. The multiplier can emulate unsigned math by setting the MSB of an 18-bit operand to zero.

The output of the multiplier consists of two 36-bit partial products. The 36-bit partial products are sign extended to 48 bits prior to being input to the adder/subtractor. Selecting the output of the multiplier consumes both X and Y multiplexers whereby the adder/subtractor combines the partial products to form the final result.

Figure 2-11 shows an optional pipeline register (MREG) for the output of the multiplier. Using the register provides increased performance with a single clock cycle of increased latency. The grey multiplexer indicates “selected at configuration time by configuration bits”.

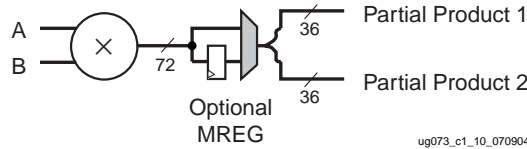


Figure 2-11: Two's Complement Multiplier Followed by Optional MREG

X, Y, and Z Multiplexer

The Operating Mode (OPMODE) inputs provide a way for the design to change its functionality from clock cycle to clock cycle (e.g., when altering the initial or final state of the DSP48 relative to the middle part of a given calculation). The OPMODE bits can be optionally registered under the control of the configuration memory cells (as denoted by the grey colored MUX symbol in Figure 2-10).

Table 2-4, Table 2-5, and Table 2-6 list the possible values of OPMODE and the resulting function at the outputs of the three multiplexers (X, Y, and Z multiplexers). The multiplexer outputs supply three 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 consumed supplying the multiplier output to the adder/subtractor.

Table 2-4: OPMODE Control Bits Select X, Y, and Z Multiplexer Outputs

OPMODE Binary			X Multiplexer Output Fed to Add/Subtract
Z	Y	X	
XXX	XX	00	ZERO (Default)
XXX	01	01	Multiplier Output (Partial Product 1)
XXX	XX	10	P
XXX	XX	11	A concatenate B

Table 2-5: OPMODE Control Bits Select X, Y, and Z Multiplexer Outputs

OPMODE Binary			Y Multiplexer Output Fed to Add/Subtract
Z	Y	X	
XXX	00	XX	ZERO (Default)
XXX	01	01	Multiplier Output (Partial Product 2)
XXX	10	XX	Illegal selection
XXX	11	XX	C

Table 2-6: OPMODE Control Bits Select X, Y, and Z Multiplexer Outputs

OPMODE Binary			Z Multiplexer Output Fed to Add/Subtract
Z	Y	X	
000	XX	XX	ZERO (Default)
001	XX	XX	PCIN
010	XX	XX	P
011	XX	XX	C
100	XX	XX	Illegal selection
101	XX	XX	Shift (PCIN)
110	XX	XX	Shift (P)
111	XX	XX	Illegal selection

There are seven possible non-zero operands for the three-input adder as selected by the three multiplexers, and the 36-bit operands are sign extended to 48 bits at the multiplexer outputs:

1. Multiplier output, supplied as two 36-bit partial products
2. Multiplier bypass bus consisting of A concatenated with B
3. C bus, 48 bits, shared by two slices
4. Cascaded P bus, 48 bits, from a neighbor DSP48 slice
5. Registered P bus output, 48 bits, for accumulator functions
6. Cascaded P bus, 48 bits, right shifted by 17 bits from a neighbor DSP48 slice
7. Registered P bus output, 48 bits, right shifted by 17 bits, for accumulator functions

Three-Input Adder/Subtractor

The adder/subtractor output is a function of control and data inputs. OPMODE, as shown in the previous section, selects the inputs to the X, Y, Z multiplexer directed to the associated three adder/subtractor inputs. It also describes how selecting the multiplier output consumes both X and Y multiplexers.

As with the input multiplexers, the OPMODE bits specify a portion of this function. Table 2-7 shows OPMODE combinations and the resulting functions. The symbol \pm in the table means either add or subtract and is specified by the state of the SUBTRACT control signal (SUBTRACT = 1 is defined as “subtraction”). The symbol $:$ in the table means concatenation. The outputs of the X and Y

multiplexer and CIN are always added together. This result is then added to or subtracted from the output of the Z multiplexer.

Table 2-7: OPMODE Control Bits Adder/Subtractor Function

Hex OPMODE	Binary OPMODE	XYZ Multiplexer Outputs and Adder/Subtractor Output				
		Z	Y	X	Adder/Subtractor Output	
0x00	000 00 00	0	0	0	\pm CIN	
0x02	000 00 10	0	0	P	$\pm(P + \text{CIN})$	
0x03	000 00 11	0	0	A:B	$\pm(A:B + \text{CIN})$	
0x05	000 01 01	0	Note 1		$\pm(A \times B + \text{CIN})$	
0x0c	000 11 00	0	C	0	$\pm(C + \text{CIN})$	
0x0e	000 11 10	0	C	P	$\pm(C + P + \text{CIN})$	
0x0f	000 11 11	0	C	A:B	$\pm(A:B + C + \text{CIN})$	
0x10	001 00 00	PCIN	0	0	PCIN \pm CIN	
0x12	001 00 10	PCIN	0	P	PCIN $\pm (P + \text{CIN})$	
0x13	001 00 11	PCIN	0	A:B	PCIN $\pm (A:B + \text{CIN})$	
0x15	001 01 01	PCIN	Note 1		PCIN $\pm (A \times B + \text{CIN})$	
0x1c	001 11 00	PCIN	C	0	PCIN $\pm (C + \text{CIN})$	
0x1e	001 11 10	PCIN	C	P	PCIN $\pm (C + P + \text{CIN})$	
0x1f	001 11 11	PCIN	C	A:B	PCIN $\pm (A:B + C + \text{CIN})$	
0x20	010 00 00	P	0	0	$P \pm \text{CIN}$	
0x22	010 00 10	P	0	P	$P \pm (P + \text{CIN})$	
0x23	010 00 11	P	0	A:B	$P \pm (A:B + \text{CIN})$	
0x25	010 01 01	P	Note 1		$P \pm (A \times B + \text{CIN})$	
0x2c	010 11 00	P	C	0	$P \pm (C + \text{CIN})$	
0x2e	010 11 10	P	C	P	$P \pm (C + P + \text{CIN})$	
0x2f	010 11 11	P	C	A:B	$P \pm (A:B + C + \text{CIN})$	
0x30	011 00 00	C	0	0	$C \pm \text{CIN}$	
0x32	011 00 10	C	0	P	$C \pm (P + \text{CIN})$	
0x33	011 00 11	C	0	A:B	$C \pm (A:B + \text{CIN})$	
0x35	011 01 01	C	Note 1		$C \pm (A \times B + \text{CIN})$	
0x3c	011 11 00	C	C	0	$C \pm (C + \text{CIN})$	
0x3e	011 11 10	C	C	P	$C \pm (C + P + \text{CIN})$	
0x3f	011 11 11	C	C	A:B	$C \pm (A:B + C + \text{CIN})$	
0x50	101 00 00	Shift (PCIN)	0	0	Shift(PCIN) \pm CIN	
0x52	101 00 10	Shift (PCIN)	0	P	Shift(PCIN) $\pm (P + \text{CIN})$	

Table 2-7: OPMODE Control Bits Adder/Subtractor Function (*Continued*)

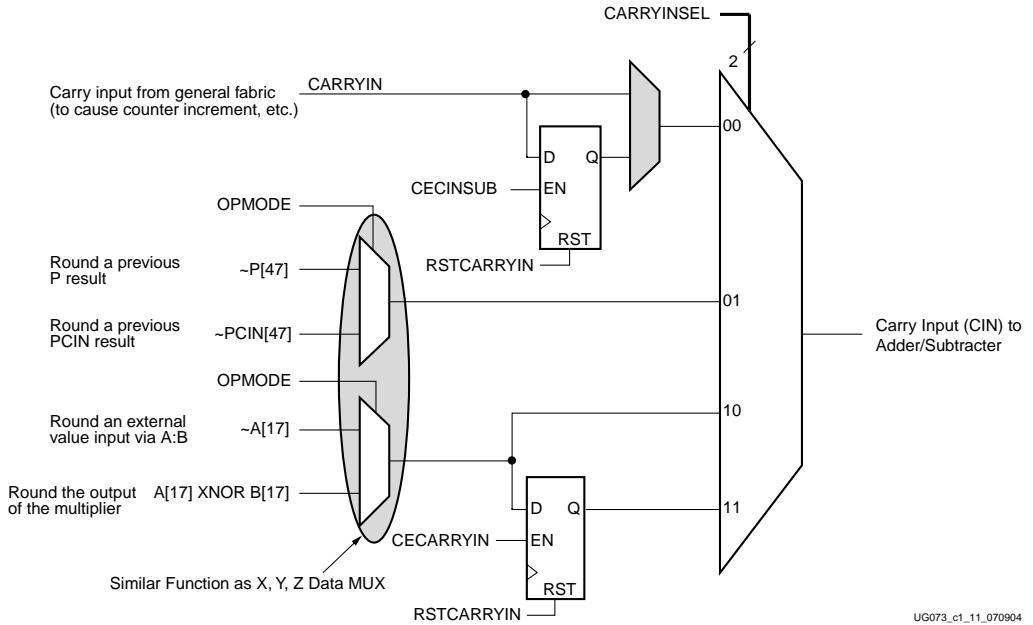
Hex OPMODE	Binary OPMODE			XYZ Multiplexer Outputs and Adder/Subtractor Output			
	Z	Y	X	Z	Y	X	Adder/Subtractor Output
0x53	1	0	1	Shift (PCIN)	0	A:B	Shift(PCIN) \pm (A:B + CIN)
0x55	1	0	1	Shift (PCIN)	Note 1		Shift(PCIN) \pm (A \times B + CIN)
0x5c	1	0	1	Shift (PCIN)	C	0	Shift(PCIN) \pm (C + CIN)
0x5e	1	0	1	Shift (PCIN)	C	P	Shift(PCIN) \pm (C + P + CIN)
0x5f	1	0	1	Shift (PCIN)	C	A:B	Shift(PCIN) \pm (A:B + C + CIN)
0x60	1	1	0	Shift (P)	0	0	Shift(P) \pm CIN
0x62	1	1	0	Shift (P)	0	P	Shift(P) \pm (P + CIN)
0x63	1	1	0	Shift (P)	0	A:B	Shift(P) \pm (A:B + CIN)
0x65	1	1	0	Shift (P)	Note 1		Shift(P) \pm (A \times B + CIN)
0x6c	1	1	0	Shift (P)	C	0	Shift(P) \pm (C + CIN)
0x6e	1	1	0	Shift (P)	C	P	Shift(P) \pm (C + P + CIN)
0x6f	1	1	0	Shift (P)	C	A:B	Shift(P) \pm (A:B + C + CIN)

Notes:

1. When the multiplier output is selected, both X and Y multiplexers are used to feed the multiplier partial products to the adder input.

Carry Input Logic

The carry input logic result is a function of the OPMODE control bits and CARRYINSEL. The inputs to the carry input logic appear in Figure 2-12. 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 the diffused silicon. The possible carry inputs to the carry logic are “gathered” prior to the outputs of the X, Y, and Z multiplexers. In a sense, the X, Y, and Z multiplexer function is duplicated for the carry inputs to the carry logic. Both OPMODE and CARRYINSEL must be in the correct state to ensure the correct carry input (CIN) is selected.



UG073_c1_11_070904

Figure 2-12: Carry Input Logic Feeding the Adder/Subtractor

Figure 2-12 shows four inputs, selected by the 2-bit CARRYINSEL control with the OPMODE bits providing additional control. The first input CARRYIN (CARRYINSEL is equal to binary 00) is driven from general logic. This option allows implementation of a carry function based on user logic. It can be optionally registered to match the pipeline delay of the MREG when used. This register delay is controlled by configuration. The next input (CARRYINSEL is equal to binary 01) is the inverted MSB of either the output P or the cascaded output, PCIN (from an adjacent DSP48 slice). The final selection between P or PCIN is dictated by OPMODE[4] and OPMODE[6]. The third input (CARRYINSEL is equal to binary 10) is the inverted MSB of A, for rounding A concatenated with B values, or A[17] XNOR B[17] for rounding multiplier outputs. Again, the state of OPMODE determines the final selection. The fourth and final input is merely a registered version of the third input to adjust the carry input delay when using the multiplier output register or MREG.

Table 2-8 lists the possible values of the two carry input select bits (CARRYINSEL), the operation mode bus (OPMODE), and the resulting carry inputs or sources.

Table 2-8: OPMODE and CARRYINSEL Control Carry Source

CARRYINSEL[1:0]	OPMODE	Carry Source	Comments
00	XXX XX XX	CARRYIN	General fabric carry source (registered or not)
01	Z MUX output = P or Shift(P)	$\sim P[47]$	Rounding P or Shift(P)
01	Z MUX output = PCIN or Shift(PCIN)	$\sim PCIN[47]$	Rounding the cascaded PCIN or Shift(PCIN) from adjacent slice
10	X and Y MUX output = multiplier partial products	A[17] xnor B[17]	Rounding multiplier (MREG pipeline register disabled)
11	X and Y MUX output = multiplier partial products	A[17] xnor B[17]	Rounding multiplier (MREG pipeline register enabled)
10	X MUX output = A:B	$\sim A[17]$	Rounding A:B (not pipelined)
11	X MUX output = A:B	$\sim A[17]$	Rounding A:B (pipelined)

Symmetric Rounding Supported by Carry Logic

Arithmetic rounding is a process where a result is quantized in an “intelligent” manner. The bit position placement where rounding occurs is up to the designer and is determined solely by a constant loaded in the C register. While the binary point placement and bit position where rounding occurs are independent of each other, the following discussion assumes one wants to round off the fractional bits.

One form of rounding is simple truncation or just 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 is biased in the wrong direction. 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.

Another method of quantization is known as “symmetric rounding”. Symmetric rounding accomplishes the more desirable effect of quantizing numbers to keep them from becoming biased in the wrong direction. For example, the number 2.8 rounds to 3.0 and the number 2.2 rounds to 2.0. Negative numbers, such as -2.8 and -2.2 , round to -3.0 and -2.0 respectively. The midpoint number 2.5 rounds to 3.0 and -2.5 rounds to -3 .

Another way to describe this type of quantization (for fractional rounding) is to round to the nearest integer and at the midpoint round away from zero. For positive numbers this effect is achieved by adding $0.1000\dots$ binary and truncating the fraction of the result. For negative numbers this effect is achieved by adding $0.0111\dots$ and truncating the fraction of the result.

The implementation of the symmetric rounding in the DSP48 slice allows the user to load a single constant. If the design calls for eight bits (out of 48 total bits) to be rounded, then load $0x00000000007F$ into the C register. The number of bits to be rounded off is one more than the

number of ones present in the C register. Table 2-9 has examples for rounding off the fractional bits from a value (binary point placement and rounded bits placement coincide).

Table 2-9: Symmetric Rounding Examples

Multiplier Output (Decimal)	Multiplier Output (Binary)	C Value	Internally Generated CIN	Multiplier Plus C Plus CIN	After Truncation (Binary)	After Truncation (Decimal)
2.4375	0010.0111	0000.0111	1	0010.1111	0010	2
2.5	0010.1000	0000.0111	1	0011.0000	0011	3
2.5625	0010.1001	0000.0111	1	0011.0001	0011	3
-2.4375	1101.1001	0000.0111	0	1110.0000	1110	-2
-2.5	1101.1000	0000.0111	0	1101.1111	1101	-3
-2.5625	1101.0111	0000.0111	0	1101.1110	1101	-3

Forming Larger Multipliers

Figure 2-13 illustrates the formation of a 35 x 35-bit multiplication from smaller 18 x 18-bit multipliers. The notation “0,B[16:0]” denotes B has a leading zero followed by 17 bits, forming a positive two's complement number.

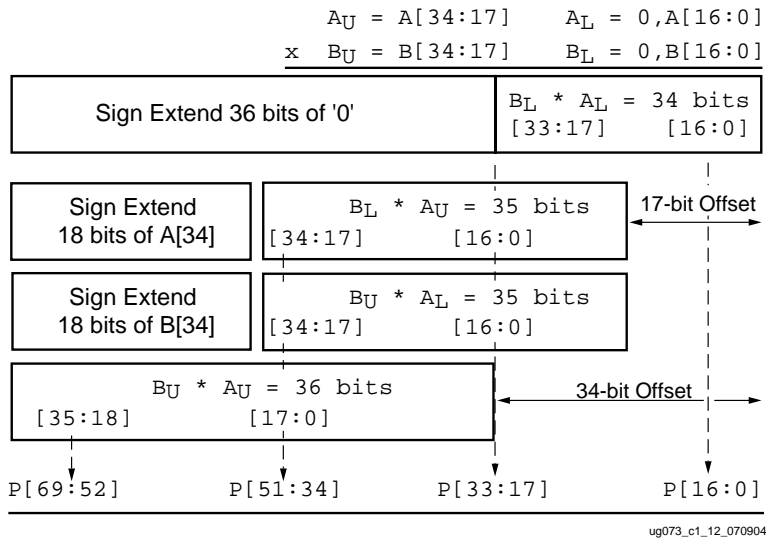


Figure 2-13: 35x35-bit Multiplication from 18x18-bit Multipliers

When separating two's complement numbers into two parts, only the most-significant part carries the original sign. The least-significant part must have a “forced zero” in the sign position meaning they are positive operands. While it seems logical to separate a positive number into the sum of two positive numbers, it can be counter intuitive to separate a negative number into a negative most-significant part and a positive least-significant part. However, after separation, the most-significant part becomes “more negative” by the amount the least-significant part becomes “more

positive”. The “forced zero sign” bit in the least-significant part is why only 35-bit operands are found instead of 36-bit operands.

The DSP48 slices, with 18 x 18 multipliers and post adder, can now be used to implement the sum of the four partial products shown in Figure 2-13. The lessor significant partial products must be right-shifted by 17 bit positions before being summed with the next most-significant partial products. This is accomplished with a built in “wire shift” applied to PCIN supplied as one selectable Z multiplexer input. The entire process of multiplication, shifting, and addition using adder cascade to form the 70-bit result can remain in the dedicated silicon of the DSP48 slice, resulting in maximum performance with minimal power consumption. Figure 2-21, page 41 illustrates the implementation of a 35 x 35 multiplier using the DSP48 slices.

FIR Filters

Basic FIR Filters

FIR filters are used extensively in video broadcasting and wireless communications. DSP 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)

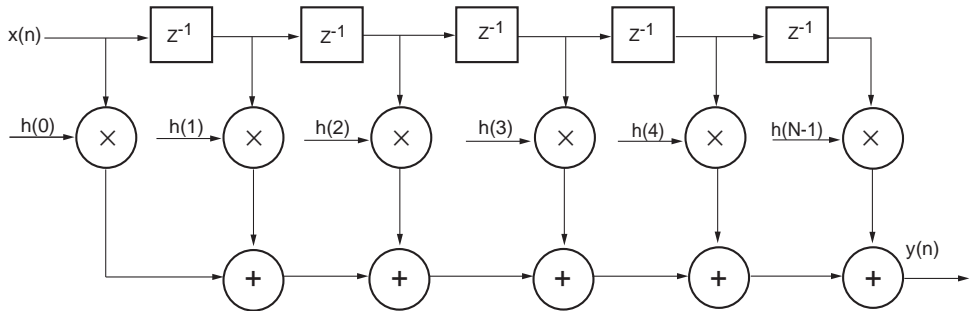
Equation 2-3 shows the basic equation for a single-channel FIR filter.

$$y(n) = \sum_{k=0}^{n-1} h(k)x(n-k)$$

Equation 2-3

The terms in the equation can be described as input samples, output samples, and coefficients. Imagine x as a continuous stream of input samples and y as a resulting stream (i.e., a filtered stream) of output samples. The n and k in the equation correspond to a particular instant in time, so to compute the output sample y(n) at time n, a group of input samples at N different points in time, or x(n), x(n-1), x(n-2), ... x(n-N+1) is required. The group of N input samples are multiplied by N coefficients and summed together to form the final result y.

The main components used to implement a digital filter algorithm include adders, multipliers, storage, and delay elements. The DSP48 slice includes all of the above elements, which makes it ideal to implement digital filter functions. All of the input samples from the set of n samples are present at the input of each DSP48 slice. Each slice multiplies the samples with the corresponding coefficients within the DSP48 slice. The outputs of the multipliers are combined in the cascaded adders.



UG073_g6_01_070904

Figure 2-14: Conventional Tapped Delay Line FIR Filter

In Figure 2-14, the sample delay logic is denoted by Z^{-1} , whereas the -1 represents a single clock delay. The delayed input samples are supplied to one input of the multiplier. The coefficients (denoted by b_0 to $b(N-1)$) are supplied to the other input of the multiplier through individual ROMs, RAMs, registers, or constants. $Y(n)$ is merely the summation of a set of input samples, and in time, multiplied by their respective coefficients.

Multi-Channel FIR Filters

Multi-channel filters are used to filter multiple data streams of input signals using the same set of coefficients for all the channels, or using different coefficient sets for different channels.

A common example of a multi-channel filter is a radio receiver's digital down converter. Equation 2-4 shows the equation, and Figure 2-15 shows the block diagram. A digitized baseband signal is applied to a matched low-pass filter $M(z)$ to reduce the data rate from the input sample rate to the bit rate. The resulting in-phase and quadrature components are each processed by the same filter and, therefore, could be processed by a single, multi-channel filter running at twice the sample rate.

$$x(n) = x_I(n) + jx_Q(n) \tag{Equation 2-4}$$

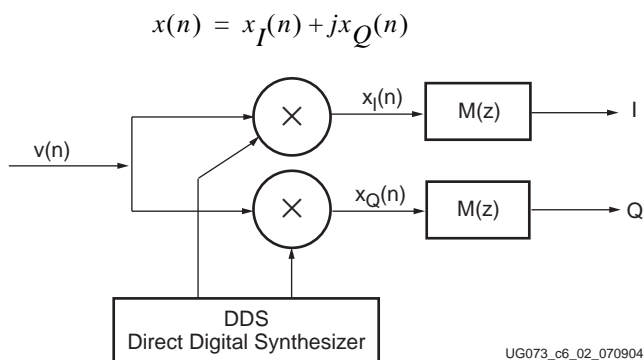


Figure 2-15: **Software-Defined Radio Digital Down Converter**

Some video applications use multi-channel implementations for multiple components of a video stream. Typical video components are red, green, and blue (RGB) or luma, chroma red, and chroma blue (YCrCb). The different video components can have the same coefficient sets or different coefficient sets for each channel by simply changing the coefficient ROM structure.

Creating FIR Filters

Referring to Figure 2-4, Table 2-4, Table 2-5, and Table 2-6, an inner product MAC operation starts by loading the first operand into the P register. The output of the multiplier is passed through the X and Y multiplexer, added to zero, and loaded into the P register. Note the load operation OPMODE with value 0000101 selects zero to be output on the Z multiplexer supplying one of the adder inputs. A previous MAC inner product can exit via the P bus during this clock cycle.

In subsequent clock cycles, the MAC operation requires the X and Y multiplexers to supply the multiplier output and the Z multiplexer to supply the output of the P register to the adder. The OPMODE for this operation, which differs from the load cycle by a single bit, has a value of 0100101. The description above allows for continuous operation with the previous resulting output and initial load occurring in the same clock cycle.

Refer to Chapter 4, “MAC FIR Filters,” for detailed information on using DSP48 slices to create MAC FIR filters.

To create a simple multiply-add processing element using the DSP48 slice shown in Figure 2-4, set the X and Y multiplexers to multiply and select the cascaded input from another DSP48 output (PCIN) as the Z MUX input to the arithmetic unit. For a normal multiply-add operation, the OPMODE value is set to 0010101.

Refer to Chapter 5, “Parallel FIR Filters,” for detailed information on using DSP48 slices to create Parallel FIR filters.

Adder Cascade vs. Adder Tree

In typical direct form FIR filters, an input stream of samples is presented to one data input of separate multipliers where coefficients supply the other input to the multipliers. An adder tree follows the multipliers where the outputs from many multipliers are combined as shown in Figure 2-16.

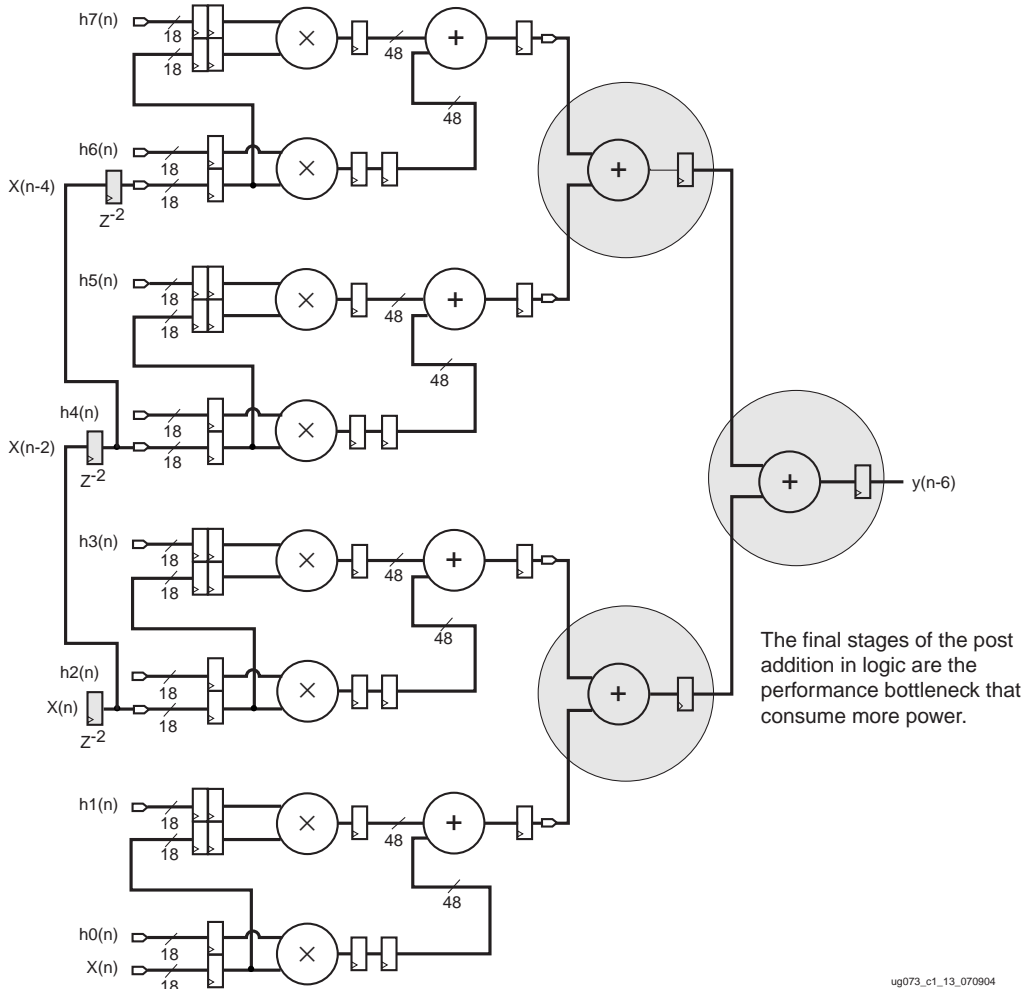


Figure 2-16: FIR Filter Adder Tree Using DSP48 Slices

One difficulty of the adder tree concept is defining the size. Filters come in various lengths and consume a variable number of adders forming an adder tree. Placing a fixed number of adder tree components in silicon displaces other elements or requires a larger FPGA, thereby increasing the cost of the design. In addition, the adder tree structure with a fixed number of additions forces the designer to use logic resources when the fixed number of additions is exceeded. Using logic resources dramatically reduces performance and increases power consumption. The key to maximizing

performance and lowering power for DSP math is to remain inside the DSP48 column consisting entirely of dedicated silicon.

The Virtex-4 solution accomplishes the post-addition process while guaranteeing no wasted silicon resources. It involves computing the additive result incrementally utilizing a cascaded approach as illustrated in Figure 2-17. Figure 2-17 is a systolic version of a direct form FIR with a latency of 10 clocks versus an adder tree latency of six clocks.

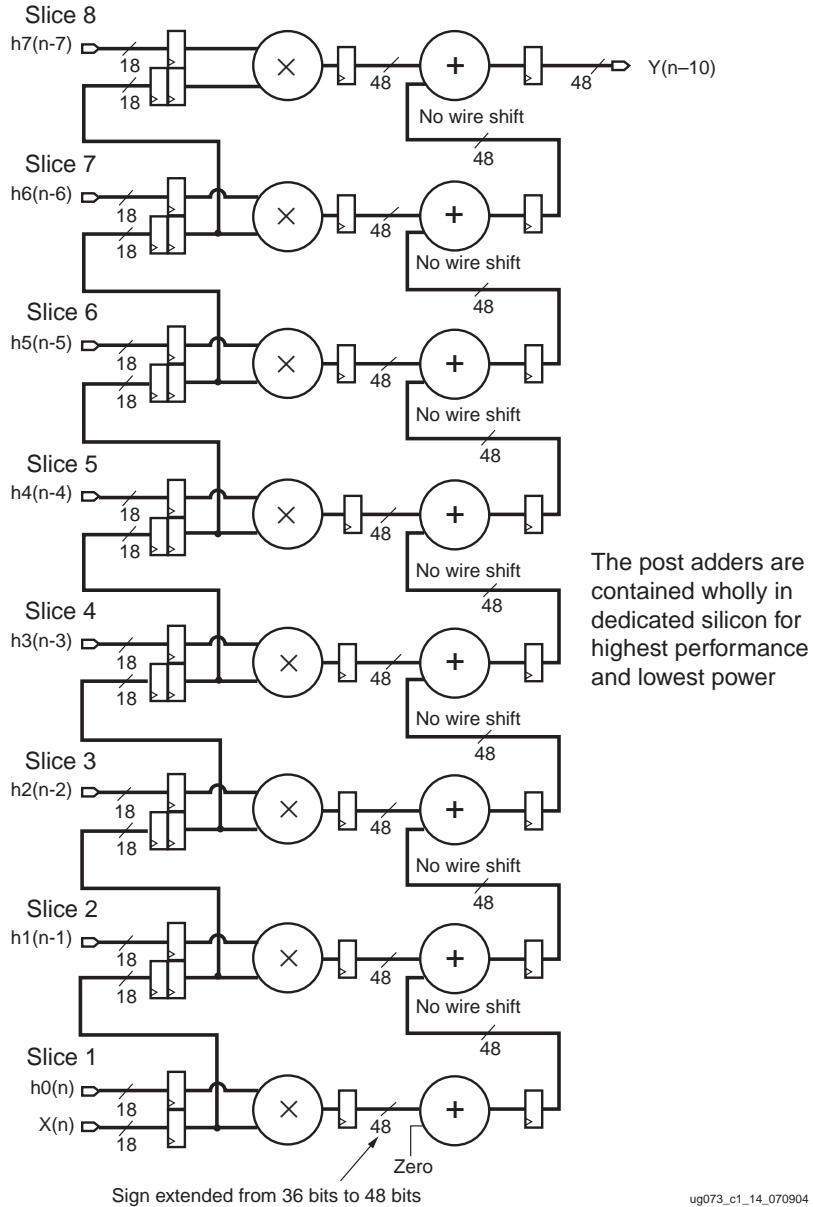


Figure 2-17: Systolic FIR with Adder Cascade

Care should be taken to balance the input sample delay and the coefficients with the cascaded adder. The adaptive coefficients are staggered in time (wave coefficients).

DSP48 Slice Functional Use Models

The use models in this section explain how the DSP48 slices are used in various DSP applications. Starting with simple multiplication and then growing in complexity, DSP48 slices can be connected in a variety of ways, trading performance and slice utilization. The tables and use models illustrate a sampling of different connections.

In some designs full performance is desired and several slices with pipelined registers are used. In designs with lower sample rates, a single slice is used with multiple clock cycles creating partial results to be combined at the very end of the computation. Performance choices (i.e., using multiple clock cycles) can produce efficient slice counts. In either case, the use of pipeline registers allows the DSP48 slice to run at a very fast, full performance clock rate.

Block diagrams showing the basic connections are also included. The “VHDL and Verilog Instantiation Templates” section shows how to instantiate and connect the DSP48 slice. In many cases, starting or ending states are different than the middle states of operation.

Single Slice, Multi-Cycle, Functional Use Models

Table 2-10 lists and summarizes four single slice use models. These examples use the high speed of the DSP48 slice to accomplish a complicated multi-cycle function by changing the OPMODE bits from cycle to cycle. Entries in the table name the function with suggestions for DSP48 slice function during different clock cycles. Further details are in the following subsections. DSP48 designs support extra pipeline stages to increase overall performance, however, the function remains the same with increased clock-cycle latency.

Table 2-10: Single Slice DSP48 Implementation

Single Slice Mode	Slice Number	Cycle	Inputs			Function and OPMODE[6:0]		Output
			A	B	C			
35 x 18 Multiply	1	1	0,A[16:0]	B[17:0]	X	Multiply	0x05	P[16:0]
		2	A[34:17]	B[17:0]	X	17-Bit Shift Feedback Multiply Add	0x65	P[52:17]
35 x 35 Multiply	1	1	0,A[16:0]	0,B[16:0]	X	Multiply	0x05	P[16:0]
		2	A[34:17]	0,B[16:0]	X	17-Bit Shift Feedback Multiply Add	0x65	
		3	0,A[16:0]	B[34:17]	X	Multiply-Accumulate	0x25	P[33:17]
		4	A[34:17]	B[34:17]	X	17-Bit Shift Feedback Multiply Add	0x65	P[69:34]

Table 2-10: Single Slice DSP48 Implementation (*Continued*)

Single Slice Mode	Slice Number	Cycle	Inputs			Function and OPCODE[6:0]		Output
			A	B	C			
Complex Multiply	1	1	A _{Re} [17:0]	B _{Re} [17:0]	X	Multiply	0x05	
		2	A _{Im} [17:0]	B _{Im} [17:0]	X	Multiply-Accumulate	0x25	P (Real)
		3	A _{Re} [17:0]	B _{Im} [17:0]	X	Multiply	0x05	
		4	A _{Im} [17:0]	B _{Re} [17:0]	X	Multiply-Accumulate	0x25	P (Imaginary)

Single Slice, 35 x 18 Multiplier Use Model

The first entry in Table 2-10 indicates how the signed 35 x 18 multiply is designed using a single DSP48 slice. The 35-bit A and 18-bit B operands are assumed to be signed, two's complement numbers with results also expressed as a signed, two's complement, 53-bit output. Operand A can only be 35 bits because when separating an operand into two 18-bit parts, the least-significant part must have the MSB forced to zero, thereby reducing the available operand bits from 36 to 35.

The multiply function uses one slice (labeled slice 0 in Table 2-10) and computes the final result in two clocks. The 36-bit, least-significant partial product output formed during the first clock cycle is computed by multiplying the least-significant 17 bits of Operand A, which are forced positive (sign bit = 0), with the 18 bits of Operand B (including the original sign).

$$0, A[16:0] \times B[17:0]$$

The first product is loaded into the output register on this cycle. The lower 17 bits of the first partial product are the lower 17 bits of the final result. During the second clock cycle, the first partial product is shifted right by 17 bits, leaving the remaining bits to be fed back and added to the next partial product. This partial product is formed by multiplying the signed 18-bit Operand B with the signed upper 18 bits of Operand A. The lower 36 bits of the second partial product are the upper 36 bits of the final result.

$$A[34:17] \times B[17:0]$$

Figure 2-18 shows the function during both clock cycles for a single DSP48 slice used as a 35-bit x 18-bit, signed, two's complement multiplier. Increased performance is obtained by using the pipeline registers before and after the multiplier, however, the clock latency is increased.

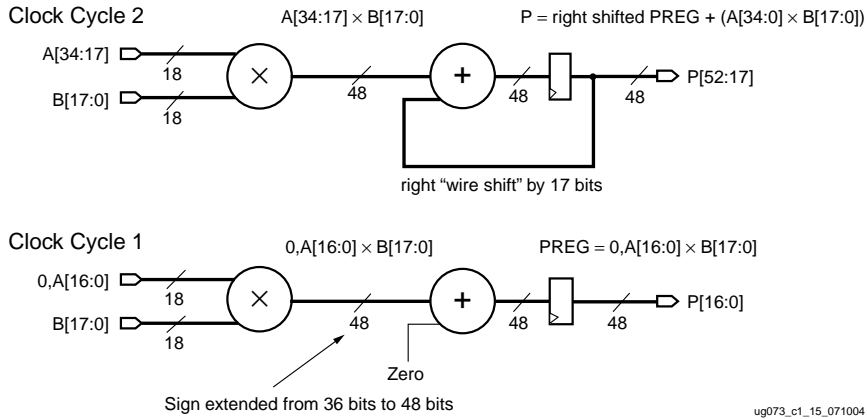


Figure 2-18: Single Slice, 35 x 18-bit Multiplier

Single Slice, 35 x 35 Multiplier Use Model

The next entry in Table 2-10 indicates how the signed 35 x 35 multiply is designed using a single DSP48 slice. The 35-bit A and B operands are assumed to be signed, two's complement numbers with results expressed as a signed, two's complement, 70-bit output. Operands can only be 35 bits because when separating an operand into two 18-bit parts. The least-significant 18-bit part must have the MSB forced to zero, thereby reducing the available operand bits from 36 to 35. The flow is similar to the 35 x 18 multiply, but instead of two partial products, there are four: a lower partial product, two middle partial products, and an upper partial product.

The multiply function uses one slice (labeled slice 1 in Table 2-10) and computes the final result in four clocks. The 36-bit lower partial product formed during the first clock cycle is computed by multiplying the least-significant 17 bits of Operand A, which are forced positive (sign bit = 0), with the least-significant 17 bits of Operand B, also forced positive.

$$0,A[16:0] \times 0,B[16:0]$$

The first product is loaded into the output register on this cycle. All 36-bit products from the multiplier are sign extended to 48 bits. During the second and third clock cycles, the two middle products are computed. In clock cycle two, the first or lower partial product in the P register is shifted right by 17 bits and fed back to the adder/subtractor. The output of the multiplier is the first middle product, expressed as:

$$A[34:17] \times 0,B[16:0]$$

The adder/subtractor is set to “add” and the two partial products are added.

In the third clock cycle, the previous result is fed back to the adder/subtractor, but not right shifted since its bits align with the next computed middle product expressed as:

$$B[34:17] \times 0,A[16:0]$$

The adder/subtractor is again set to add, and the P register receives the sum of the three partial products.

Finally, in the fourth clock cycle, the accumulated sum of partial products is again shifted right by 17 bits, and sign extended leaving the remaining bits to be fed back and added to the next partial

product. The upper partial product is formed by multiplying the signed upper 18 bits of B with the signed upper 18 bits of A.

$$A[34:17] \times B[34:17]$$

The 70-bit result is output sequentially in 17-bit, 17-bit, and 36-bit segments as shown in Figure 2-19.

Figure 2-19 shows the function during all four clock cycles for a single DSP48 slice used as a 35-bit x 35-bit, signed, two's complement multiplier. Increased performance can be obtained by using the pipeline registers before and after the multiplier, however, the clock latency is increased.

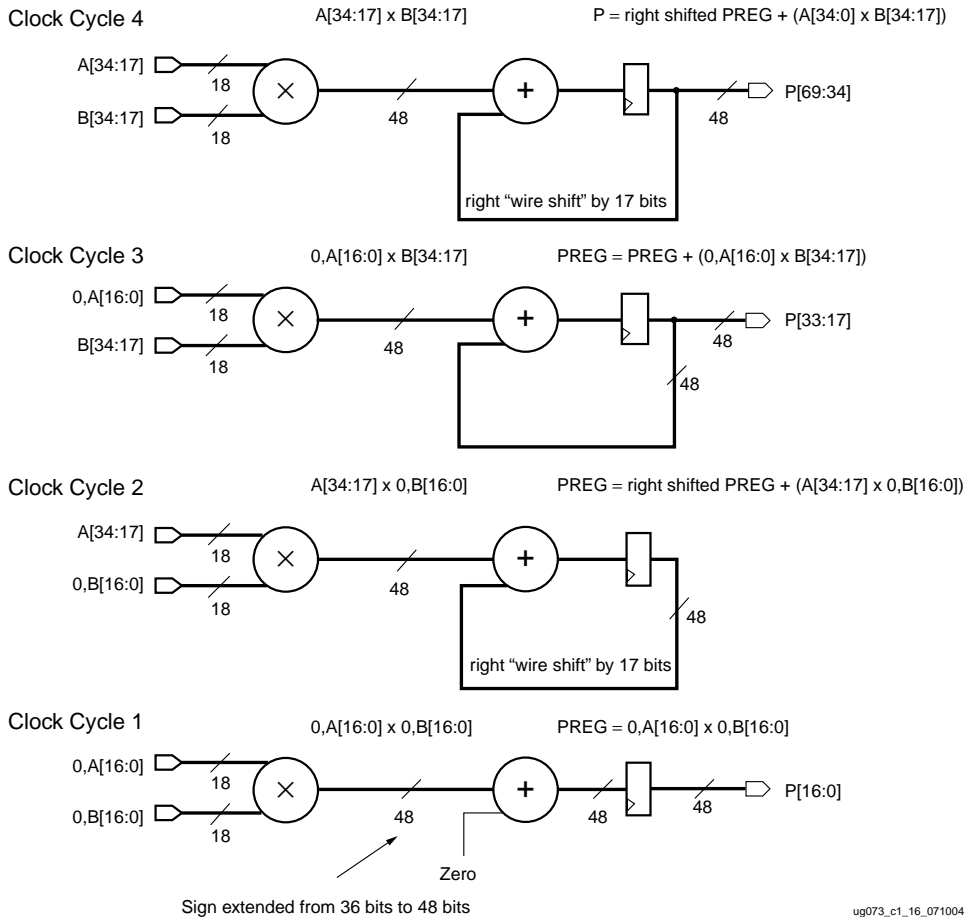


Figure 2-19: Single Slice, 35 x 35-bit Multiplier

Fully Pipelined Functional Use Models

Table 2-11 summarizes six fully pipelined functional use models. The table provides the function name and suggests what each DSP48 slice is doing. More details are provided in the following subsections. The designs are fully pipelined and run at the maximum DSP48 slice clock rate.

Table 2-11: Fully Pipelined DSP48 Implementations

Multiple Slice Mode	Slice	Inputs			Function and OPCODE[6:0]	Output	
		A	B	C			
35 x 18 Multiply Figure 2-20	1	0,A[16:0]	B[17:0]	X	Multiply	0x05	P[16:0]
	2	A[34:17]	B[17:0]	X	17-Bit Shifted Feedback Multiply Add	0x65	P[52:17]
35 x 35 Multiply Figure 2-21	1	0,A[16:0]	0,B[16:0]	X	Multiply	0x05	P[16:0]
	2	A[34:17]	0,B[16:0]	X	17-Bit Shifted Feedback Multiply Add	0x65	
	3	0,A[16:0]	B[34:17]	X	Multiply Accumulate	0x25	P[33:17]
	4	A[34:17]	B[34:17]	X	17-Bit Shifted Feedback Multiply Add	0x65	P[69:34]
18 x 18 Complex Multiply Figure 2-22	1	A _{Re} [17:0]	B _{Re} [17:0]	X	Multiply	0x05	
	2	A _{Im} [17:0]	B _{Im} [17:0]	X	Multiply Accumulate	0x25	P (Real)
	3	A _{Re} [17:0]	B _{Im} [17:0]	X	Multiply	0x05	
	4	A _{Im} [17:0]	B _{Re} [17:0]	X	Multiply Accumulate	0x25	P (Imaginary)
18 x 18 Complex MAC Figure 2-23	1	A _{Re} [17:0]	B _{Re} [17:0]	X	Multiply	0x05	
	2	A _{Im} [17:0]	B _{Im} [17:0]	X	Multiply Accumulate	0x25	P (Real)
	3	A _{Re} [17:0]	B _{Im} [17:0]	X	Multiply	0x05	
	4	A _{Im} [17:0]	B _{Re} [17:0]	X	Multiply Accumulate	0x25	P (Imaginary)
35 x 18 Complex Multiply Real Part Figure 2-26	1	A _{Re} [17:0]	B _{Re} [17:0]	X	Multiply	0x05	
	2	A _{Im} [17:0]	B _{Im} [17:0]	X	Multiply Accumulate	0x25	P (Real)
	3	A _{Re} [17:0]	B _{Im} [17:0]	X	Multiply	0x05	
	4	A _{Im} [17:0]	B _{Re} [17:0]	X	Multiply Accumulate	0x25	P (Imaginary)
35 x 18 Complex Multiply Imaginary Part Figure 2-27	1	A _{Re} [17:0]	B _{Re} [17:0]	X	Multiply	0x05	
	2	A _{Im} [17:0]	B _{Im} [17:0]	X	Multiply Accumulate	0x25	P (Real)
	3	A _{Re} [17:0]	B _{Im} [17:0]	X	Multiply	0x05	
	4	A _{Im} [17:0]	B _{Re} [17:0]	X	Multiply Accumulate	0x25	P (Imaginary)

Table 2-12 summarizes utilization of more complex digital filters possible using the DSP48. The small “n” in the Silicon Utilization column indicates the number of DSP48 filter taps. The construction and operation of complex filters is discussed in Chapter 4, “MAC FIR Filters,” Chapter 5, “Parallel FIR Filters,” and Chapter 6, “Semi-Parallel FIR Filters.”

Table 2-12: Composite Digital Filters

Digital Filter	Silicon Utilization	OPMODE
Multi-Channel FIR	n DSP slices, n RAM	Static
Direct Form FIR	n DSP slices	Static
Transposed Form FIR	n DSP slices	Static
Systolic Form FIR	n DSP slices	Static
Polyphase Interpolator	n DSP slices, n RAM	Static
Polyphase Decimator	n DSP slices, n RAM	Dynamic
CIC Decimation/Interpolation Filters	1 DSP slice per stage	Static

Fully Pipelined, 35 x 18 Multiplier Use Model

The previous single slice use models show how performance and power consumption can be traded for a very small implementation (i.e., single slice). However, many DSP solutions require very high sample rates. When sample rates approach the maximum inherent clock rate for the math elements in the FPGA, it becomes necessary to design using parallel, fully pipelined math elements.

With fully pipelined designs, inputs can be presented and an output computed every single clock cycle. In addition, the DSP48 slice circuits and interconnect are very carefully matched, ensuring no path becomes the timing bottleneck. Keeping math implementations mostly inside the DSP48 maximizes performance and minimizes power consumption. Of course, pipelining does have increased clock latency, but this is usually not a problem in DPS algorithms.

In the single slice versions of this algorithm, partial products are computed sequentially and summed in the adder. For the fully pipelined version of the algorithm, the same partial products are computed in parallel and summed in the last slice producing a result and consuming new input operands every clock cycle.

The single slice version of the 35 x 18 multiply uses two clock cycles. In each clock cycle the slice is presented with different operands, and switching the OPMODE bits modifies the behavior. The fully pipelined versions connect separate slices with fixed behavior.

In the 35 x 18-bit multiply block diagram (Figure 2-20), the most-significant input data part for the 35-bit A is delayed with an extra input register in the second slice. This allows the cascading B input to be available to the second slice multiply at the same time as the most-significant data part for A. The extra register delay for the cascading B input and most-significant part of A also guarantee the output of the multiply in the second slice arrive at the same time as the partial product result from the first slice multiply.

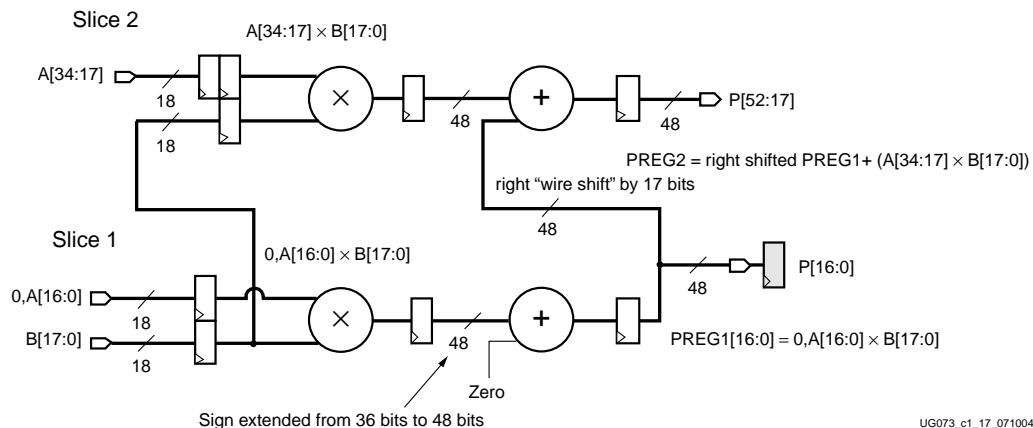


Figure 2-20: Fully Pipelined, 35 x 18 Multiplier

Fully Pipelined, 35 x 35 Multiplier Use Model

Similar to the 35 x 18-bit example, this fully pipelined design can present inputs every clock cycle. An output is also computed every single clock cycle. Once again, no particular path becomes the timing bottleneck. The single slice version of the 35 x 35-bit multiply uses four clock cycles. In each clock cycle the slice is presented with different operands and switching the OPMODE bits modifies the behavior. The fully pipelined version connects separate slices with fixed behavior.

In the single slice versions of this algorithm, partial products are computed sequentially and summed in the adder. For the fully pipelined version of the algorithm, the same partial products are computed in parallel and summed in the last slice, producing a result and consuming new input operands every clock cycle.

As in the 35 x 18-bit example, there are additional register stages placed in the input paths to delay input data until the needed cascading results arrive. In Figure 2-21, the block diagram for the fully pipelined, 35 x 35 multiply shows where additional input register stages are placed. The 35 x 35-bit multiplier has additional output registers outside of the slice to align the output data. The notation Z^{-3} is in the external register to signify the data must be delayed by three clock cycles. If the delay is only one cycle, then registers are typically used. When the delay is larger than one, an SRL16 followed by the associated CLB flip-flop achieves maximum design performance.

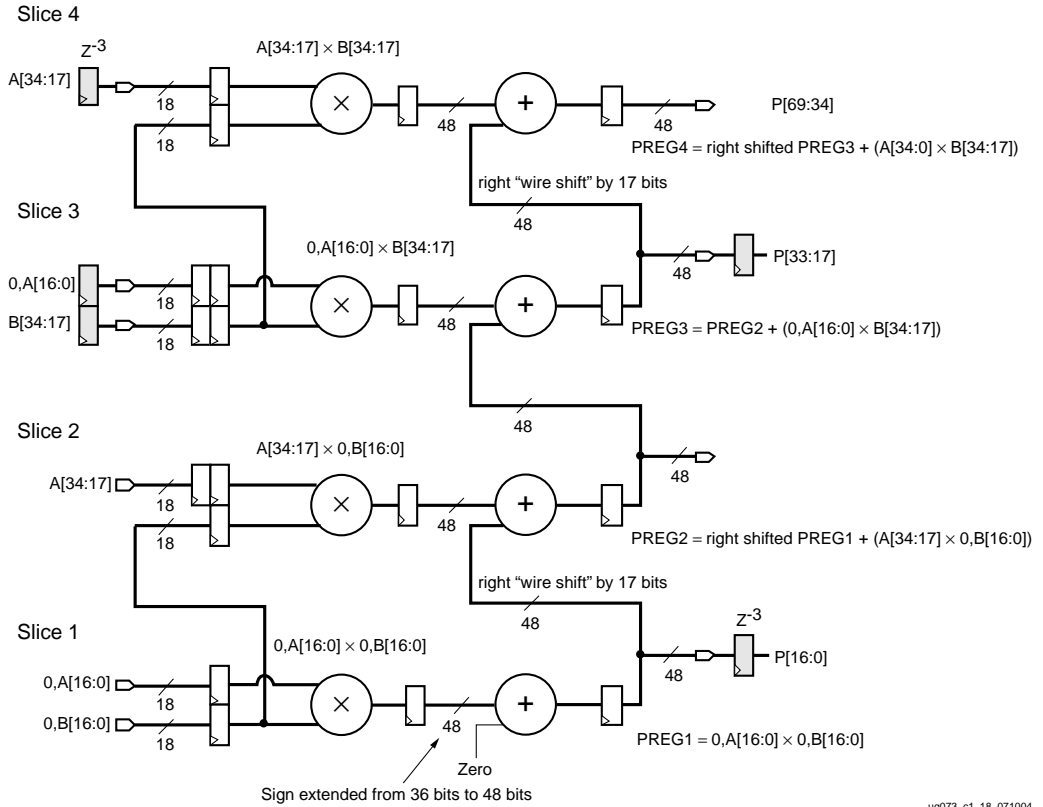


Figure 2-21: Fully Pipelined, 35 x 35 Multiplier

Fully Pipelined, Complex, 18 x 18 Multiplier Use Model

Complex multiplication used in many DSP applications combines operands having both real and imaginary parts into results with real and imaginary parts. Two operands A and B, each having real and imaginary parts, are combined as shown in the following equations:

$$(A_real \times B_real) - (A_imaginary \times B_imaginary) = P_real$$

$$(A_real \times B_imaginary) + (A_imaginary \times B_real) = P_imaginary$$

The real and imaginary results use the same slice configuration with the exception of the adder/subtractor. The adder/subtractor performs subtraction for the real result and addition for the imaginary result.

Figure 2-22 shows several DSP48 slices used as a complex, 18-bit x 18-bit multiplier.

ug073_c1_18_071004

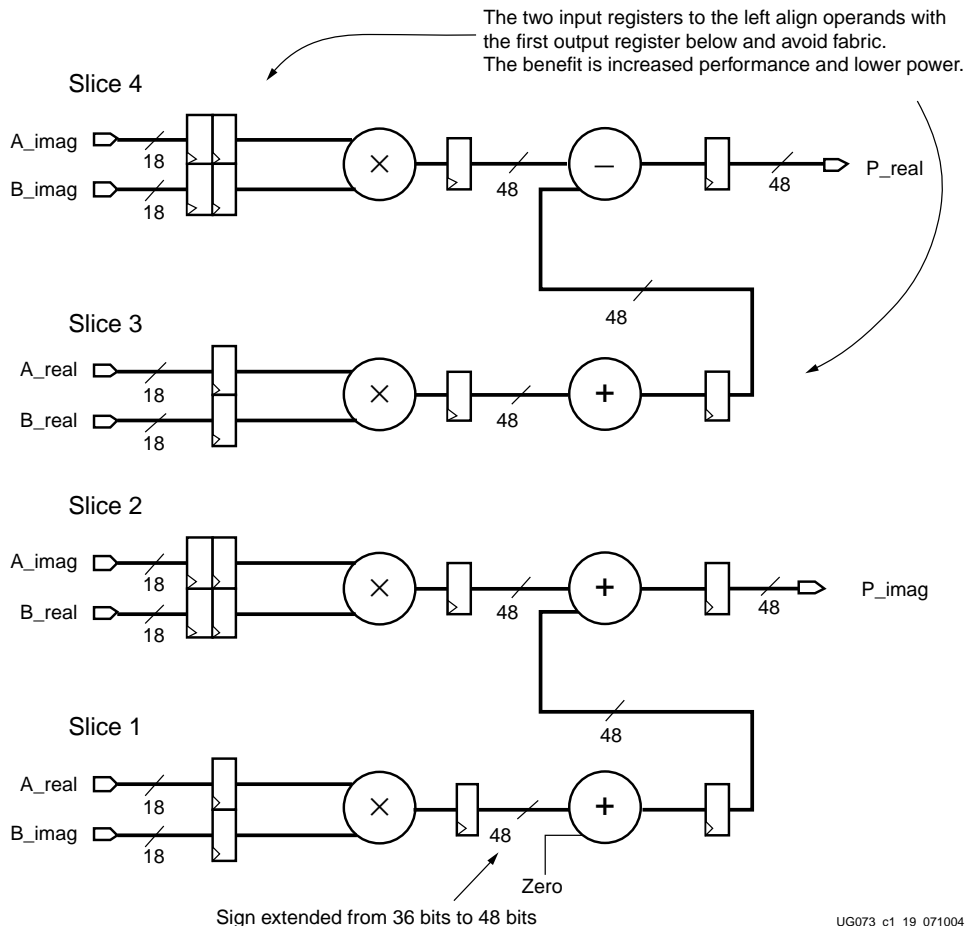


Figure 2-22: Pipelined, Complex, 18 x 18 Multiply

Note: The real and the imaginary computations are functionally similar using different input data. The real output subtracts the multiplied terms, and the imaginary output adds the multiplied terms.

Fully Pipelined, Complex, 18 x 18 MAC Use Model

The differences between complex multiply and complex MAC implementations using several DSP48 slices is illustrated with the next set of equations. As shown, the addition and subtraction of the terms only occur after the desired number of MAC operations.

For N Cycles:

- Slice 1 = (A_real × B_imaginary) accumulation
- Slice 2 = (A_imaginary × B_real) accumulation
- Slice 3 = (A_real × B_real) accumulation
- Slice 4 = (A_imaginary × B_imaginary) accumulation

Last Cycle:

Slice 1 + Slice 2 = P_imaginary

Slice 3 – Slice 4 = P_real

During the last cycle, the input data must stall while the final terms are added. To avoid having to stall the data, instead of using the complex multiply implementation shown in Figure 2-23 and Figure 2-24, use the complex multiply implementation shown in Figure 2-25.

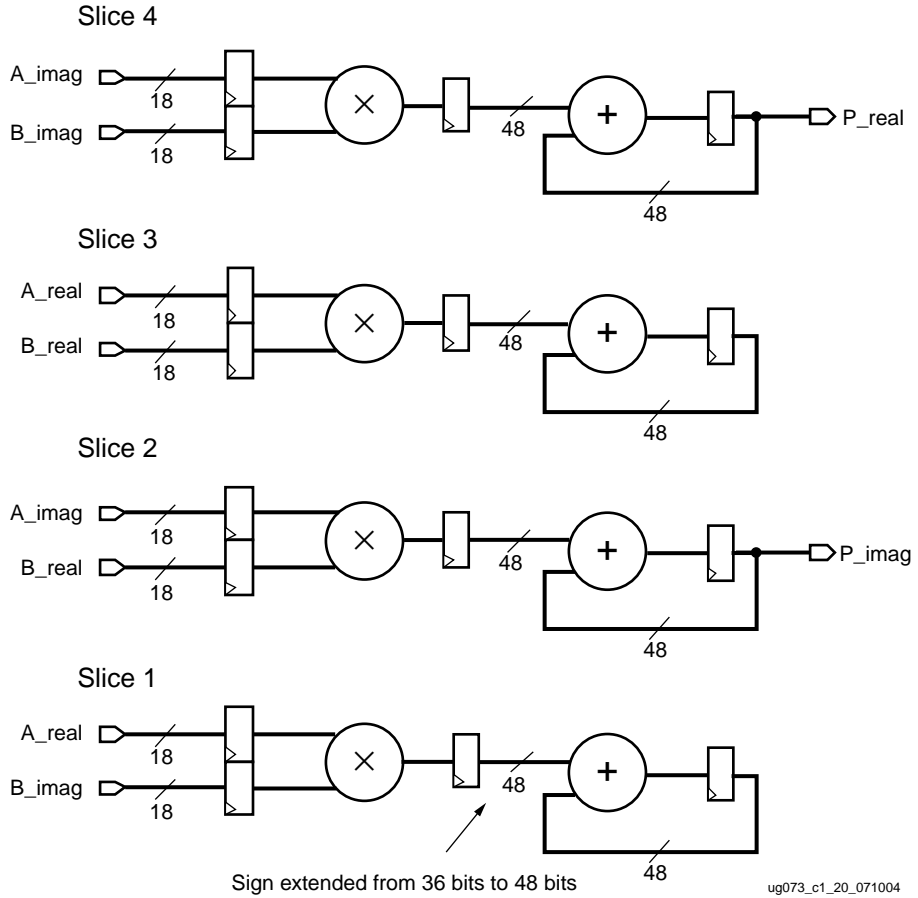
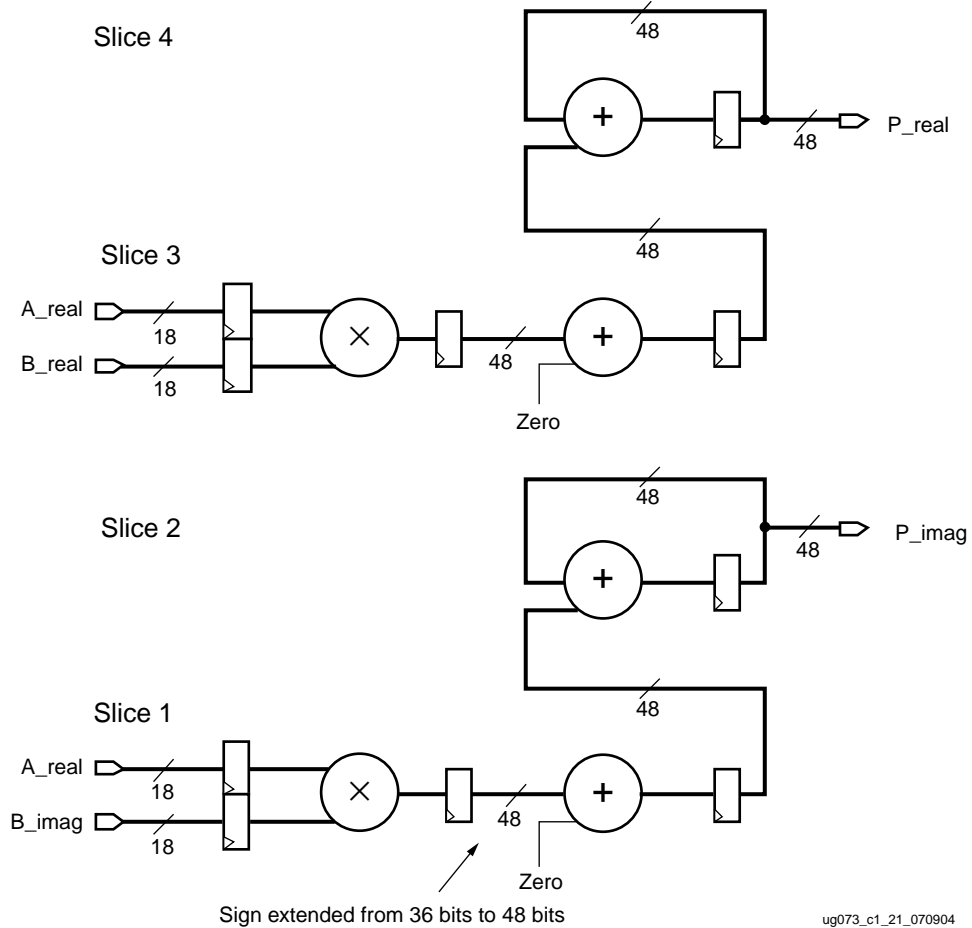


Figure 2-23: Fully Pipelined, Complex, 18 x 18 MAC (N Cycles)

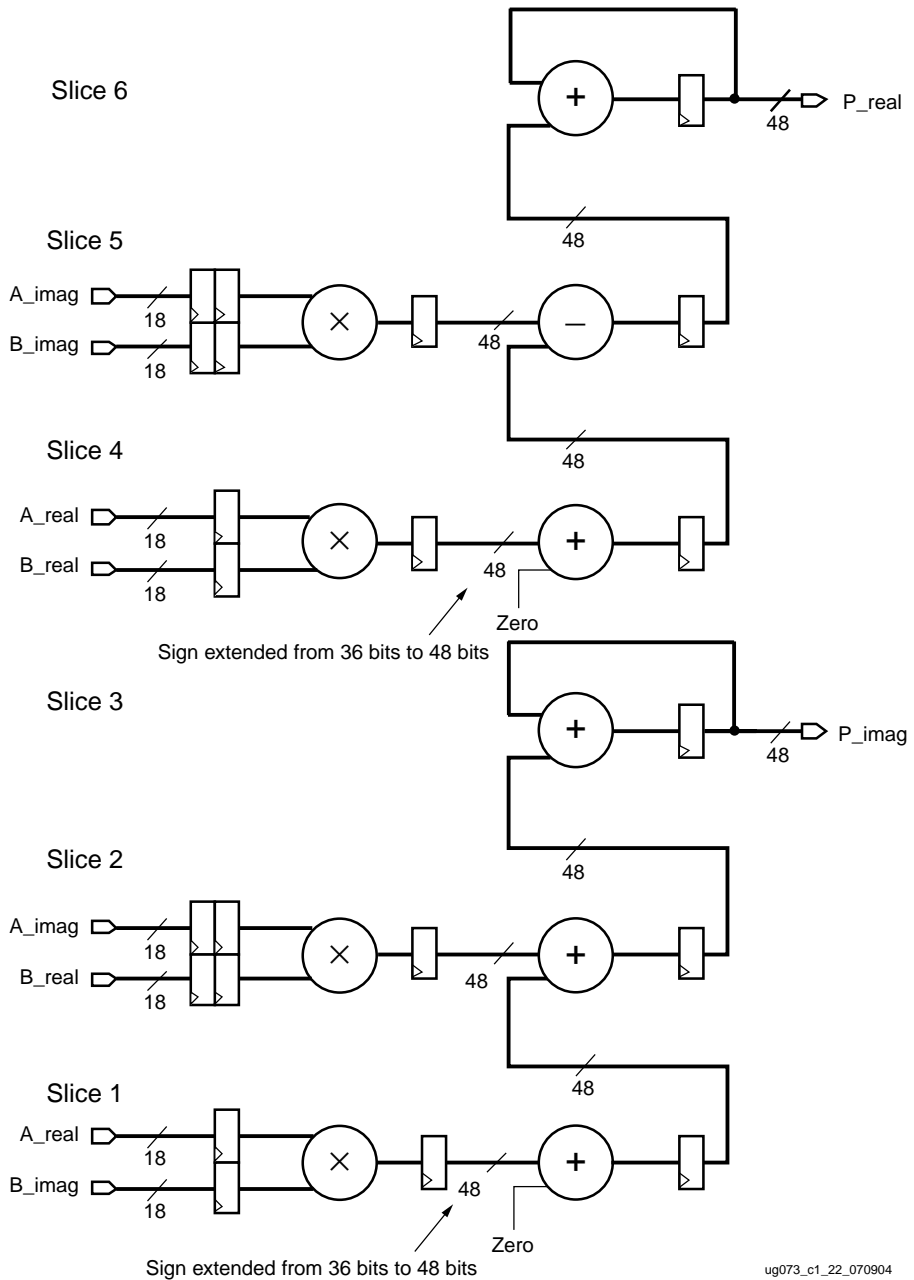
In Figure 2-24, the N+1 cycle adds the accumulated products, and the input data stalls one cycle.



ug073_c1_21_070904

Figure 2-24: Fully Pipelined, Complex, 18 x 18 MAC (Last or N+1 Cycle)

An additional slice used for the accumulation is shown in Figure 2-25. The extra slice prevents the input data from stalling on the last cycle. The capability of accumulating the P cascade through the X mux feedback eliminates the pipeline stall.



ug073_c1_22_070904

Figure 2-25: Fully Pipelined, Complex, 18 x 18 MAC with Extra Slice

Fully Pipelined, Complex, 35 x 18 Multiplier Usage Model

Many complex multiply algorithms require higher precision in one of the operands. The equations for combining the real and imaginary parts in complex multiplication are the same, but the larger operands must be separated into two parts and combined using partial product techniques. As shown in the other examples, the real and imaginary results use the same slice configuration with the exception of the adder/subtractor. The adder/subtractor performs subtraction for the real result and addition for the imaginary result. The following equations describe the math used to form the real and imaginary parts for the fully pipelined, complex, 35-bit x 18-bit multiplication.

$$(A_real \times B_real) - (A_imaginary \times B_imaginary) = P_real$$

$$(A_real \times B_imaginary) + (A_imaginary \times B_real) = P_imaginary$$

Figure 2-26 shows the real part of a fully pipelined, complex, 35-bit x 18-bit multiplier.

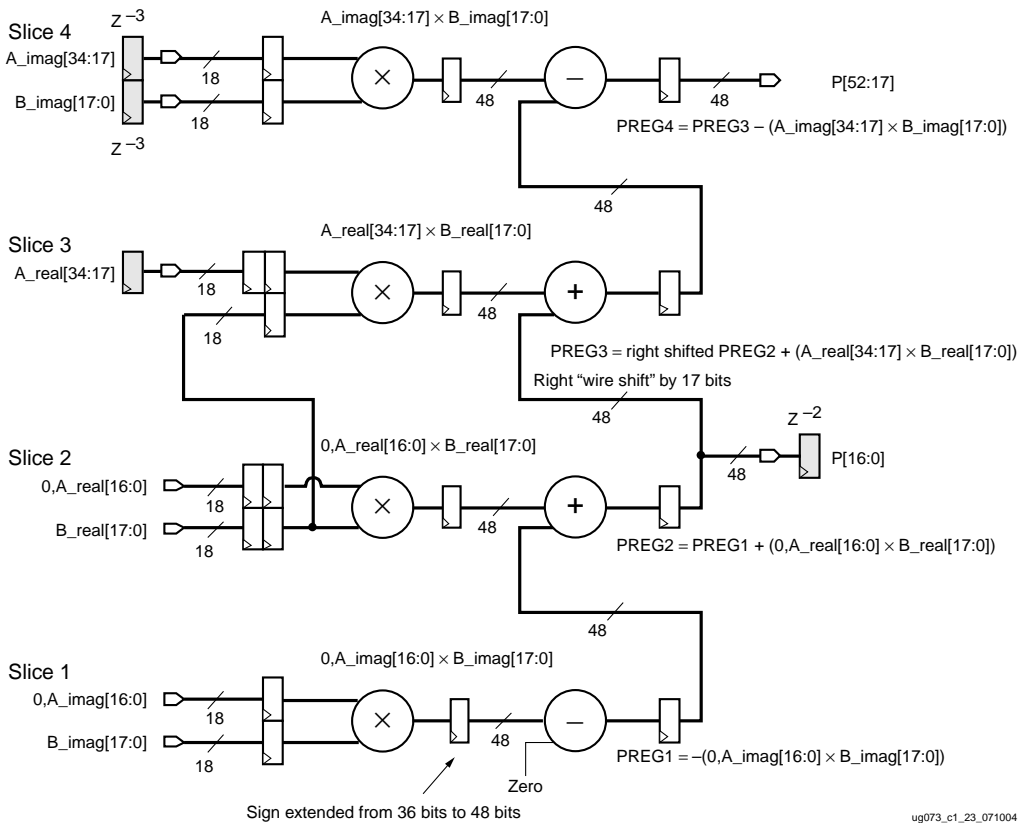


Figure 2-26: Real Part of a Pipelined, Complex, 35 x 18 Multiply

Figure 2-27 shows the imaginary part of a fully pipelined, complex, 35-bit x 18-bit multiplier.

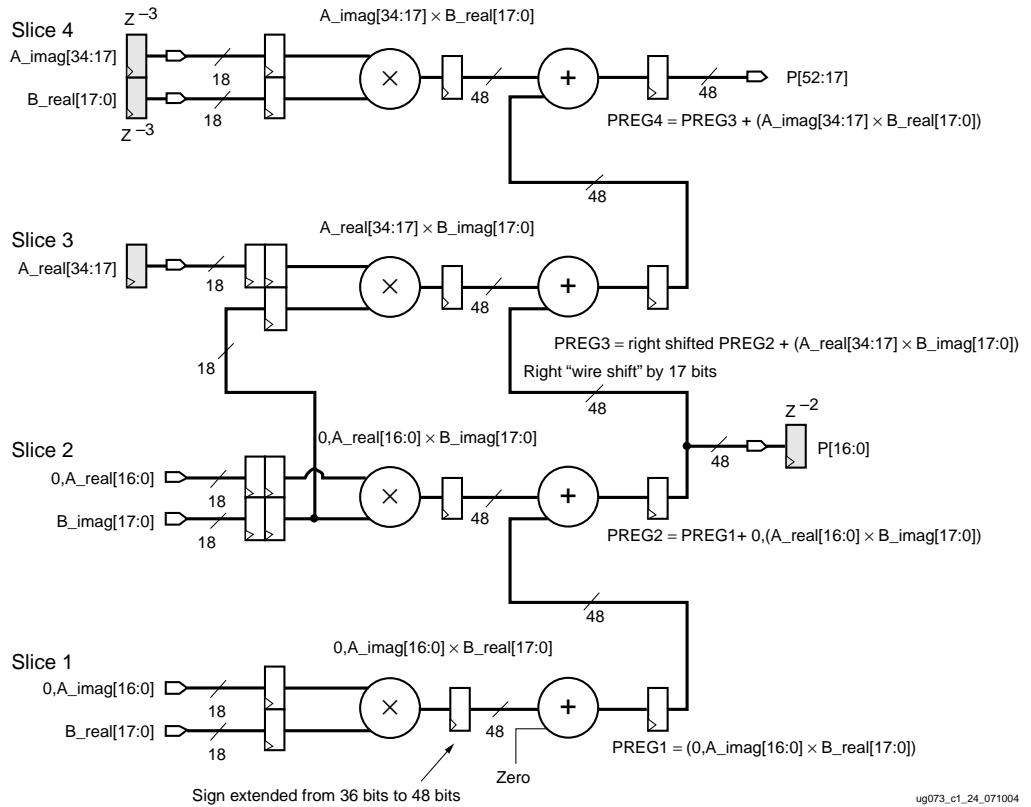


Figure 2-27: Imaginary Part of a Pipelined, Complex, 35 x 18 Multiply

Miscellaneous Functional Use Models

Table 2-13 summarizes a few miscellaneous functional use models.

Table 2-13: Miscellaneous Functional Use Models

Miscellaneous	Silicon Utilization	OPMODE
18-bit Barrel Shifter	2 DSP slices	Static
48-bit Add Subtract	2 DSP slices	Static
36-bit Add Subtract Cascade	n DSP slices	Static
n word MUX, 48 bit words	2n DSP slices	Dynamic
n word MUX, 36 bit words	n DSP slices	Dynamic
48-bit Counter	1 DSP slice	Static
Magnitude Compare	1 DSP slice, logic	Static
Equal to Zero Compare	1 DSP slice, logic	Static

Table 2-13: Miscellaneous Functional Use Models (*Continued*)

Miscellaneous	Silicon Utilization	OPMODE
24 2-input ANDs	1 DSP slice	Static
24 2-input XORs	1 DSP slice	Static
Up to 48-bit AND	1 DSP slice	Static

Dynamic, 18-bit Circular Barrel Shifter Use Model

The barrel shift function is very useful when trying to realign data very quickly. Using two DSP48 slices, an 18-bit circular barrel shifter can be implemented. This implementation shifts 18 bits of data left by the number of bit positions represented by *n*. The bits shifted out of the most-significant part reappear in the lower significant part of the answer completing the circular shift. The equations in Figure 2-28 describe what value is carried out of the first slice, what this value looks like after shifting right 17 bits, and finally what is visible as a result.

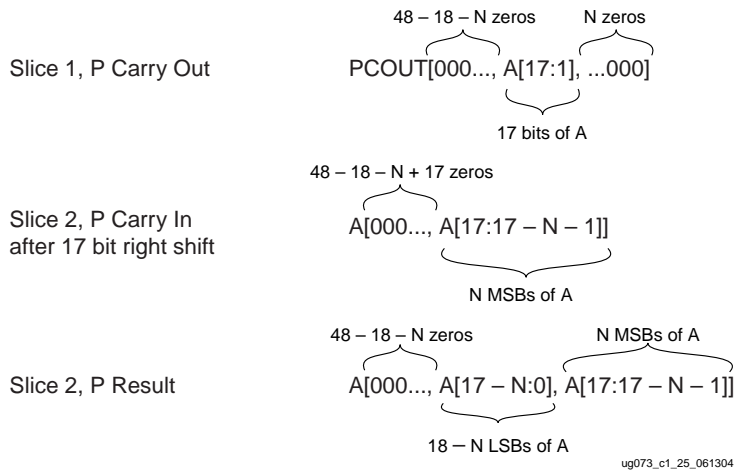


Figure 2-28: Circular Barrel Shifter Equations

Figure 2-29 shows the DSP48 used an 18-bit circular barrel shifter. The P register for slice 1 contains leading zeros in the MSBs, followed by the most-significant 17 bits of *A*, followed by *n* trailing zeros. If *n* equals zero, then there are no trailing zeros and the P register contains leading zeros followed by 17 bits of *A*.

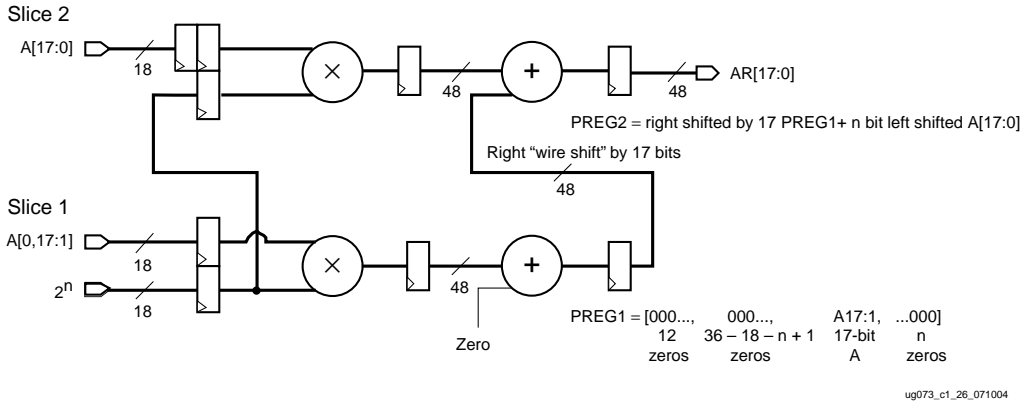


Figure 2-29: Dynamic 18-Bit Barrel Shifter

In the case of n equal to zero (i.e., no shift), the P register of slice 1 is passed to slice 2 with 17 bits of right shift. This leaves all 48 bits of the P carry input effectively equal to zero since A[17:1] were shifted toward the least-significant direction. If there is a positive shift amount, then P carry out of slice 1 contains A[17:1] padded in front by $48 - 17 - n$ zeros and in back by n zeros. After the right shift by 17, only the n most-significant bits of A remain in the lower 48 bits of the P carry input.

This n -bit guaranteed positive number is added to the A[17:0], left shifted by n bits. In the n least-significant bits there are zeros. The end result contained in A[17:0] of the second slice P register is A[17 - n : n , 17:17 - n + 1] or a barrel shifted A[17:0]. The design is fully pipelined and can generate a new result every clock cycle at the maximum DSP48 clock rate.

A single slice version of the dynamic 18-bit barrel shifter can be implemented. For this implementation, Table 2-14 describes the DSP48 slice function and OPMODE settings for each clock cycle.

Table 2-14: Miscellaneous DSP48 Implementations

Single Slice Mode	Cycle	Inputs			Function and OPMODE[6:0]		Output
		A	B	C			
18-bit Barrel Shifter	0	A[17:0]	B[17:0]	X	Multiply	0x05	
	1	A[17:0]	B[17:0]	X	Multiply Accumulate	0x25	P
	2	A[17:0]	B[17:0]	X	Multiply	0x05	
	3	A[17:0]	B[17:0]	X	Multiply Accumulate	0x25	P

VHDL and Verilog Instantiation Templates

This section describes the VHDL and Verilog instantiation templates. In VHDL, each template has a component declaration section and an architecture section. Insert each part of the template within the VHDL design file. The port map of the architecture section should include the design signal names.

VHDL Instantiation Template

```
-- DSP48                               : In order to incorporate this function into the
design,                                  :
-- VHDL                                 : the following instance declaration needs to be
placed                                  :
-- instance                             : in the body of the design code. The instance name
-- declaration                          : (DSP48_inst) and/or the port declarations after the
-- code                                  : "=" declaration maybe changed to properly
reference and                            :
--                                     : connect this function to the design. All inputs
--                                     : and outputs must be connected.

-- Library                               : In addition to adding the instance declaration, a
use                                      :
-- declaration                          : statement for the UNISIM.vcomponents library needs
-- for                                  : to be added before the entity declaration. This
library                                  :
-- Xilinx                               : contains the component declarations for all Xilinx
-- primitives                           : primitives and points to the models that will be
used                                     :
--                                     : for simulation.

-- Copy the following two statements and paste them before the
-- Entity declaration, unless they already exists.

Library UNISIM;
use UNISIM.vcomponents.all;

-- <-----Cut code below this line and paste into the architecture body----->

-- DSP48: DSP Function Block
--      Virtex-4
-- Xilinx HDL Language Template version 6.1i

DSP48 inst: DSP48 generic map (
  AREG => 1, -- Number of pipeline registers on the A input, 0, 1 or 2
  BREG => 1, -- Number of pipeline registers on the B input, 0, 1 or 2
  B_INPUT => "DIRECT", -- B input DIRECT from fabric or CASCADE from another
DSP48
  CARRYINREG => 1, -- Number of pipeline registers for the CARRYIN input,
0 or 1
  CARRYINSELREG => 1, -- Number of pipeline registers for the CARRYINSEL,
0 or 1
  CREG => 1, -- Number of pipeline registers on the C input, 0 or 1
  LEGACY_MODE => "MULT18X18S", -- Backward compatibility, NONE,

-- MULT18X18 or MULT18X18S
  MREG => 1, -- Number of multiplier pipeline registers, 0 or 1
  OPMODEREG => 1, -- Number of pipeline registers on OPMODE input, 0 or 1
  PREG => 1, -- Number of pipeline registers on the P output, 0 or 1
  SIM_X_INPUT => "GENERATE_X_ONLY", -- Simulation parameter for
behavior_for X on input.
-- Possible values: GENERATE_X,
NONE or WARNING
  SUBTRACTREG => 1) -- Number of pipeline registers on the SUBTRACT input,
0 or 1

  port map (
    BCOUT => BCOUT, -- 18-bit B cascade output
```

```

P => P,          -- 48-bit product output
PCOUT => PCOUT, -- 38-bit cascade output
A => A,          -- 18-bit A data input
B => B,          -- 18-bit B data input
BCIN => BCIN,   -- 18-bit B cascade input
C => C,          -- 48-bit cascade input
CARRYIN => CARRYIN, -- Carry input signal
CARRYINSEL => CARRYINSEL, -- 2-bit carry input select
CEA => CEA,     -- A data clock enable input
CEB => CEB,     -- B data clock enable input
CEC => CEC,     -- C data clock enable input
CECARRYIN => CECARRYIN, -- CARRYIN clock enable input
CECINSUB => CECINSUB, -- CINSUB clock enable input
CECTRL => CECTRL, -- Clock Enable input for CTRL registers
CEM => CEM,     -- Clock Enable input for multiplier
                registers
CEP => CEP,     -- Clock Enable input for P registers
CLK => CLK,     -- Clock input
OPMODE => OPMODE, -- 7-bit operation mode input
PCIN => PCIN,   -- 48-bit PCIN input
RSTA => RSTA,   -- Reset input for A pipeline registers
RSTB => RSTB,   -- Reset input for B pipeline registers
RSTC => RSTC,   -- Reset input for C pipeline registers
RSTCARRYIN => RSTCARRYIN, -- Reset input for CARRYIN registers
RSTCTRL => RSTCTRL, -- Reset input for CTRL registers
RSTM => RSTM,   -- Reset input for multiplier registers
RSTP => RSTP,   -- Reset input for P pipeline registers
SUBTRACT => SUBTRACT -- SUBTRACT input
);
-- End of DSP48_inst instantiation

```

Verilog Instantiation Template

The following is a synthesis instantiation template for the DSP48 slice in Verilog. After the port list are synthesis attributes with syntax written for the Xilinx Synthesis Tool (XST). If using a different synthesis tool, consult the tools user guide and change the attributes appropriately. The section after the synthesis attributes consists of “defparam” statements that are ignored by the synthesis process, but are used to initialize the simulation model to match the synthesis attributes during simulation.

```

// DSP48      : In order to incorporate this function into the design,
// Verilog    : the following instance declaration needs to be placed
// instance   : in the body of the design code. The instance name
// declaration : (DSP48_inst) and/or the port declarations within the
// code       : parenthesis maybe changed to properly reference and
//           : connect this function to the design. All inputs
//           : and outputs must be connected.

// <-----Cut code below this line----->

// DSP48: DSP Function Block
//       Virtex-4
// Xilinx HDL Language Template version 7.1i

DSP48 DSP48_inst (
    .BCOUT(BCOUT), // 18-bit B cascade output
    .P(P),         // 48-bit product output
    .PCOUT(PCOUT), // 38-bit cascade output
    .A(A),         // 18-bit A data input
    .B(B),         // 18-bit B data input
    .BCIN(BCIN),  // 18-bit B cascade input
    .C(C),         // 48-bit cascade input
    .CARRYIN(CARRYIN), // Carry input signal
    .CARRYINSEL(CARRYINSEL), // 2-bit carry input select
    .CEA(CEA),    // A data clock enable input

```

```

        .CEB(CEB),           // B data clock enable input
        .CEC(CEC),           // C data clock enable input
        .CECARRYIN(CECARRYIN), // CARRYIN clock enable input
        .CECINSUB(CECINSUB), // CINSUB clock enable input
        .CECTRL(CECTRL),     // Clock Enable input for CTRL registers
        .CEM(CEM),           // Clock Enable input for multiplier
registers
        .CEP(CEP),           // Clock Enable input for P registers
        .CLK(CLK),           // Clock input
        .OPMODE(OPMODE),     // 7-bit operation mode input
        .PCIN(PCIN),         // 48-bit PCIN input
        .RSTA(RSTA),         // Reset input for A pipeline registers
        .RSTB(RSTB),         // Reset input for B pipeline registers
        .RSTC(RSTC),         // Reset input for C pipeline registers
        .RSTCARRYIN(RSTCARRYIN), // Reset input for CARRYIN registers
        .RSTCTRL(RSTCTRL),   // Reset input for CTRL registers
        .RSTM(RSTM),         // Reset input for multiplier registers
        .RSTP(RSTP),         // Reset input for P pipeline registers
        .SUBTRACT(SUBTRACT) // SUBTRACT input
    );

    // The following defparams specify the behavior of the DSP48 slice.
    // If the instance name to the DSP48 is changed, that change needs to
    // be reflected in the defparam statements.

    defparam DSP48_inst.AREG = 1; // Number of pipeline registers on the A
input, 0, 1 or 2
    defparam DSP48_inst.BREG = 1; // Number of pipeline registers on the B
input, 0, 1 or 2
    defparam DSP48_inst.B_INPUT = "DIRECT"; // B input DIRECT from fabric
// or CASCADE from another
DSP48
    defparam DSP48_inst.CARRYINREG = 1; // Number of pipeline
registers
// for the CARRYIN input, 0
or 1
    defparam DSP48_inst.CARRYINSELREG = 1; // Number of pipeline
registers for the
// CARRYINSEL, 0 or 1
    defparam DSP48_inst.CREG = 1; // Number of pipeline registers on the C
input, 0 or 1
    defparam DSP48_inst.LEGACY_MODE = "MULT18X18S"; // Backward compatibility,
// NONE, MULT18X18 or
MULT18X18S
    defparam DSP48_inst.MREG = 1; // Number of multiplier
pipeline registers, 0 or 1
    defparam DSP48_inst.OPMODEREG = 1; // Number of pipeline
registers on
// OPMODE input, 0 or 1
    defparam DSP48_inst.PREG = 1; // Number of pipeline registers on the P
output, 0 or 1
    defparam DSP48_inst.SIM_X_INPUT = "GENERATE_X_ONLY"; // Simulation
parameter for behavior
// for X on input.
Possible values:
// GENERATE_X, NONE
or WARNING
    defparam DSP48_inst.SUBTRACTREG = 1; // Number of pipeline
registers
// on the SUBTRACT input, 0
or 1

    // End of DSP48_inst instantiation

```