Implementing Multipliers in FPGA Devices

Introduction

Stratix® II, Stratix, Stratix GX, Cyclone™ II, and Cyclone devices have dedicated architectural features that make it easy to implement high-performance multipliers. Stratix II, Stratix, and Stratix GX devices feature embedded high-performance multiplier-accumulators (MACs) in dedicated digital signal processing (DSP) blocks. DSP blocks can operate at data rates above 300 million samples per second (MSPS), making Stratix II, Stratix, and Stratix GX devices ideal for high-speed DSP applications. Cyclone II devices have embedded multiplier blocks for DSP.

In addition to the dedicated DSP blocks, designers can also use the Stratix II, Stratix, and Stratix GX devices' TriMatrix™ memory blocks to implement high-performance soft multipliers of variable depths and widths. For example, designers can useTriMatrix memory blocks as look-up tables (LUTs) that contain partial results from multiplication of input data with coefficients. Cyclone II and Cyclone devices have M4K memory blocks which can be used as LUTs to implement variable depth/width high-performance soft multipliers for low cost, high volume DSP applications.

Stratix II, Stratix, Stratix GX, Cyclone II, and Cyclone devices can implement the multiplier types shown in Table 1.

Multiplier Type	Description	Devices				
		Stratix II	**Stratix**	**Stratix GX**	**Cyclone II**	**Cyclone**
Soft multiplier	These multipliers are implemented as LUTs in memory, which contains all possible partial results from multiplication. There are five soft multiplier modes: ■ Parallel multiplication ■ Semi-parallel multiplication ■ Sum of multiplication ■ Hybrid multiplication ■ Fully variable multipliers	✓	✓	✓	✓	✓
Multipliers using DSP blocks, embedded multipliers, or logic resources	These multipliers are implemented in dedicated DSP blocks, embedded multipliers, or logic resources using the `lpm_mult`, `altmult_add`, or `altmult_accum` megafunctions.	✓	✓	✓	✓	*(1)*
Firm multiplier	These multipliers are implemented in a combination of DSP blocks or embedded multipliers and logic resources.	✓	✓	✓	✓	-

Table 1. Supported Multiplier Implementations

Note to Table 1:
(1) Cyclone devices can implement these multiplication functions using logic resources only.

Tables 2 through 4 show the total number of multipliers available in Stratix II, Stratix, and Stratix GX devices using DSP blocks and soft multipliers. Table 5 shows the total number of multipliers available in

Cyclone II devices using embedded multipliers and soft multipliers. Table 6 shows the total number of soft multipliers available in Cyclone devices.

Table 2. Number of Multipliers in Stratix II Devices			
Device	**DSP Blocks (18 × 18)**	**Soft Multipliers (16 × 16)** *(1)*	**Total Multipliers** *(2)*, *(3)*
EP2S15	48	100	148 (3.08)
EP2S30	64	189	253 (3.95)
EP2S60	144	325	469 (3.26)
EP2S90	192	509	701 (3.65)
EP2S130	252	750	1,002 (3.98)
EP2S180	384	962	1,346 (3.51)

Notes to Table 2:
(1)   Soft multipliers implemented in sum of multiplication mode. RAM blocks are configured with 18-bit data widths and sum of coefficients up to 18-bits.
(2)   The number in parentheses represents the increase factor, which is the total number of multipliers with soft multipliers divided by the number of 18 × 18 multipliers supported by DSP blocks only.
(3)   The total number of multipliers may vary according to the multiplier mode used.

Table 3. Number of Multipliers in Stratix Devices			
Device	**DSP Blocks (18 × 18)**	**Soft Multipliers (16 × 16)** *(1)*	**Total Multipliers** *(2)*, *(3)*
EP1S10	24	89	113 (4.71)
EP1S20	40	142	182 (4.55)
EP1S25	40	208	248 (6.20)
EP1S30	48	263	311 (6.48)
EP1S40	56	303	359 (6.41)
EP1S60	72	471	543 (7.54)
EP1S80	88	603	691 (7.85)

Notes to Table 3:
(1)   Soft multipliers implemented in sum of multiplication mode. RAM blocks configured with 18-bit data widths and sum of coefficients up to 18 bits.
(2)   The number in parentheses represents the increase factor, which is the total number of multipliers with soft multipliers divided by the number of 18 × 18 multipliers supported by DSP blocks only.
(3)   The total number of multipliers may vary according to the multiplier mode used.

Table 4. Number of Multipliers in Stratix GX Devices

Device	DSP Blocks (18 × 18)	Soft Multipliers (16 × 16) (1)	Total Multipliers (2), (3)
EP1SGX10C	24	89	113 (4.71)
EP1SGX10D	24	89	113 (4.71)
EP1SGX25C	40	208	248 (6.20)
EP1SGX25D	40	208	248 (6.20)
EP1SGX25F	40	208	248 (6.20)
EP1SGX40D	56	303	359 (6.41)
EP1SGX40G	56	303	359 (6.41)

Notes to Table 4:
(1) Soft multipliers implemented in sum of multiplication mode. RAM blocks configured with 18-bit data widths and sum of coefficients up to 18 bits.
(2) The number in parentheses represents the increase factor, which is the total number of multipliers with soft multipliers divided by the number of 18 × 18 multipliers supported by DSP blocks only.
(3) The total number of multipliers may vary according to the multiplier mode used.

Table 5. Number of Multipliers in Cyclone II Devices

Device	Embedded Multipliers (18 × 18)	Soft Multipliers (16 × 16) (1)	Total Multipliers (2), (3)
EP2C5	13	26	39 (3.00)
EP2C8	18	36	54 (3.00)
EP2C20	26	52	78 (3.00)
EP2C35	35	105	140 (4.00)
EP2C50	86	129	215 (2.50)
EP2C70	150	250	400 (2.67)

Notes to Table 4:
(1) Soft multipliers implemented in sum of multiplication mode. RAM blocks configured with 18-bit data widths and sum of coefficients up to 18 bits.
(2) The number in parentheses represents the increase factor, which is the total number of multipliers with soft multipliers divided by the number of 18 × 18 multipliers supported by DSP blocks only.
(3) The total number of multipliers may vary according to the multiplier mode used.

Table 6. Number of Multipliers in Cyclone Devices	
Device	**Soft Multipliers (16 × 16)** *(1)*, *(2)*
EP1C3	13
EP1C4	17
EP1C6	20
EP1C12	52
EP1C20	64

Notes to Table 6:
(1) Soft multipliers implemented in sum of multiplication mode. RAM blocks configured with 18-bit data widths and sum of coefficients up to 18 bits.
(2) The total number of multipliers may vary according to the multiplier mode used.

This application note describes the dedicated memory and DSP blocks, the supported multiplier types, and includes an example of each type.

Memory Blocks

The Stratix II, Stratix, and Stratix GX TriMatrix memory blocks consist of three types of RAM blocks: M512, M4K, and M-RAM. The M512 and M4K RAM blocks are memory blocks with a maximum width of 18 and 36 bits, respectively, and a maximum performance of approximately 300 MHz, which is ideal for implementing soft multipliers.

Tables 7 through 9 show the available TriMatrix memory blocks in Stratix II, Stratix, and Stratix GX devices, respectively.

Table 7. Stratix II TriMatrix Memory Blocks				
Device	**M512 RAM (32 × 18 Bits)**	**M4K RAM (128 × 36 Bits)**	**M-RAM (4K × 144 Bits)**	**Total RAM Bits**
EP2S15	104	78	0	419,328
EP2S30	202	144	1	1,369,728
EP2S60	329	255	2	2,544,192
EP2S90	488	408	4	4,520,448
EP2S130	699	609	6	6,747,840
EP2S180	930	768	9	9,383,040

Table 8. Stratix TriMatrix Memory Blocks

Device	M512 RAM (32 × 18 Bits)	M4K RAM (128 × 36 Bits)	M-RAM (4K × 144 Bits)	Total RAM Bits
EP1S10	94	60	1	920,448
EP1S20	194	82	2	1,669,248
EP1S25	224	138	2	1,944,576
EP1S30	295	171	4	3,317,184
EP1S40	384	183	4	3,423,744
EP1S60	574	292	6	5,215,104
EP1S80	767	364	9	7,427,520

Table 9. Stratix GX TriMatrix Memory Blocks

Device	M512 RAM (32 × 18 Bits)	M4K RAM (128 × 36 Bits)	M-RAM (4K × 144 Bits)	Total RAM Bits
EP1SGX10C	94	60	1	920,448
EP1SGX10D	94	60	1	920,448
EP1SGX25C	224	138	2	1,944,576
EP1SGX25D	224	138	2	1,944,576
EP1SGX25F	224	138	2	1,944,576
EP1SGX40D	384	183	4	3,423,744
EP1SGX40G	384	183	4	3,423,744

The Cyclone II and Cyclone M4K memory blocks have a maximum width of 36 bits and a maximum performance of 250 MHz (200 MHz for Cyclone M4K blocks). Tables 10 and 11 show the number of Cyclone II and Cyclone M4K memory blocks in each device, respectively.

Table 10. Cyclone II M4K Memory Blocks

Device	M4K RAM (128 × 36 Bits)
EP2C5	26
EP2C8	36
EP2C20	52
EP2C35	105
EP2C50	129
EP2C70	250

Table 11. Cyclone M4K Memory Blocks

Device	M4K RAM (128 × 36 Bits)
EP1C3	13
EP1C4	17
EP1C6	20
EP1C12	52
EP1C20	64

Table 12 shows the possible configurations of the M512, M4K, and M-RAM blocks found in Stratix II, Stratix, Stratix GX, Cyclone II, and Cyclone devices.

Table 12. M512, M4K & M-RAM Memory Configurations

M512 RAM Block (32 × 18 Bits)	M4K RAM Block (128 × 36 Bits)	M-RAM Block (4K × 144 Bits)
512 × 1	4K × 1	64K × 8
256 × 2	2K × 2	64K × 9
128 × 4	1K × 4	32K × 16
64 × 8	512 × 8	32K × 18
64 × 9	512 × 9	16K × 32
32 × 16	256 × 16	16K × 36
32 × 18	256 × 18	8K × 64
-	128 × 32	8K × 72
-	128 × 36	4K × 128
-	-	4K × 144

DSP Blocks

Stratix II, Stratix, and Stratix GX devices contain dedicated DSP blocks for implementing high-speed multiplication functions within the device. Tables 13 through 15 show the number of DSP blocks in Stratix II, Stratix, and Stratix GX devices, respectively.

Table 13. Number of DSP Blocks in Stratix II Devices (Part 1 of 2)   Note (1)

Device	DSP Blocks	9 × 9 Multipliers	18 × 18 Multipliers	36 × 36 Multipliers
EP2S15	12	96	48	48
EP2S30	16	128	64	64

Table 13. Number of DSP Blocks in Stratix II Devices (Part 2 of 2) Note (1)				
Device	DSP Blocks	9 × 9 Multipliers	18 × 18 Multipliers	36 × 36 Multipliers
EP2S60	36	288	144	144
EP2S90	48	384	192	192
EP2S130	63	504	252	252
EP2S180	96	768	384	384

Note to Table 13:
(1)   Each device has either the number of 9 × 9, 18 × 18, or 36 × 36 multipliers shown. The total number of multipliers for each device is not the sum of all the multipliers.

Table 14. Number of DSP Blocks in Stratix Devices Note (1)				
Device	DSP Blocks	9 × 9 Multipliers	18 × 18 Multipliers	36 × 36 Multipliers
EP1S10	6	48	24	6
EP1S20	10	80	40	10
EP1S25	10	80	40	10
EP1S30	12	96	48	12
EP1S40	14	112	56	14
EP1S60	18	144	72	18
EP1S80	22	176	88	22

Note to Table 14:
(1)   Each device has either the number of 9 × 9, 18 × 18, or 36 × 36 multipliers shown. The total number of multipliers for each device is not the sum of all the multipliers.

Table 15. Number of DSP Blocks in Stratix GX Devices Note (1)				
Device	DSP Blocks	9 × 9 Multipliers	18 × 18 Multipliers	36 × 36 Multipliers
EP1SGX10C	6	48	24	6
EP1SGX10D	6	48	24	6
EP1SGX25C	10	80	40	10
EP1SGX25D	10	80	40	10
EP1SGX25F	10	80	40	10
EP1SGX40D	14	112	56	14
EP1SGX40G	14	112	56	14

Note to Table 15:
(1)   Each device has either the number of 9 × 9, 18 × 18, or 36 × 36 multipliers shown. The total number of multipliers for each device is not the sum of all the multipliers.

DSP Arithmetic

DSP is a multiplication-intensive technology and to achieve high speeds, these multiplication operations must be accelerated. This section provides information on the mathematical theory and algorithms behind common DSP arithmetic implementations.

Multiplication

The base of many DSP algorithms is multiplication in which a multiplier is multiplied to a multiplicand. In this operation, each element of the multiplier is multiplied by each bit of the multiplicand. Then, the partial product of each multiplication is accumulated according to the weight of the partial product, where the weight indicates the location of a bit corresponding to other bits. For example, if a partial product of bits 4 through 7 is added to a partial product of bits 0 through 3, the partial product of 4 through 7 is shifted according to their weight and then accumulated to the partial product of previous stages. Figure 1 shows a simple 2×2 multiplication of multiplier a1a0 to multiplicand b1b0.

Figure 1. Multiplication of Two 2-Bit Numbers
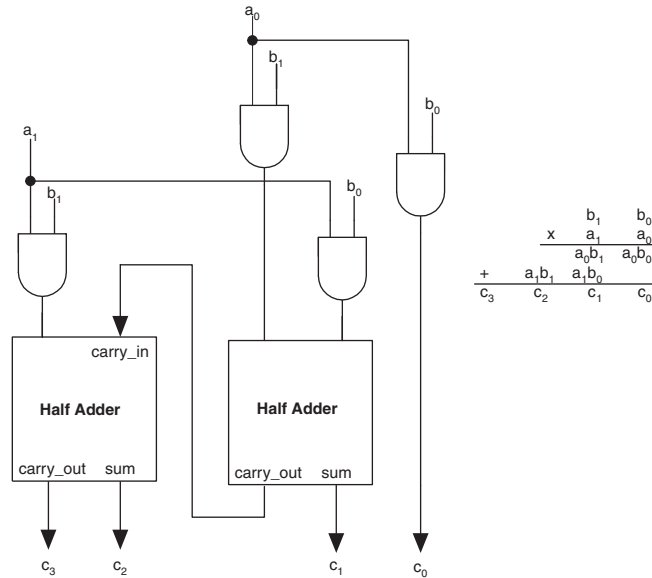
Distributed Arithmetic

Distributed arithmetic is a method of performing multiplication by distributing the operation over many LUTs. Figure 2 shows a four-product MAC function that uses sequential shift and add to multiply four pairs, and then sums their partial product to obtain a final result. Each multiplier forms partial products by multiplying the multiplicand by one bit of the input data (multiplier) at a time, using an AND gate.

Figure 2. Distributed Arithmetic with Four Constant Multiplicands
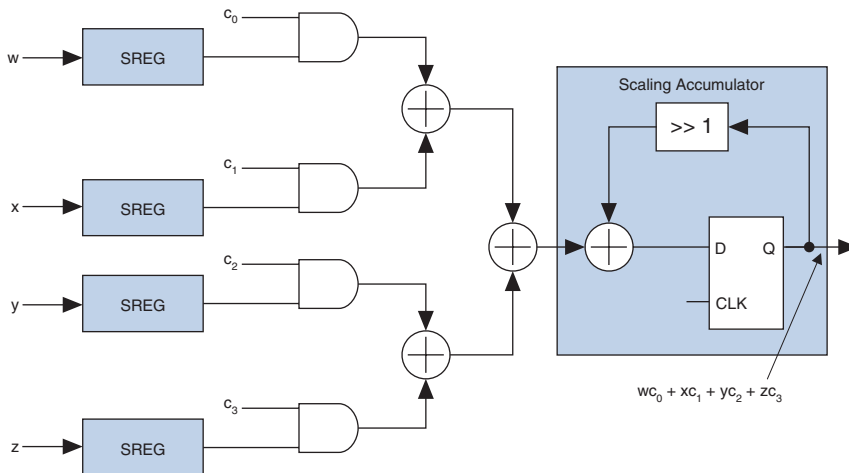
At the end of the process, each partial product result of each input bit is summed prior to the final scaling accumulator stage, which performs a shift-accumulate.

The distributed-arithmetic circuit simultaneously performs four multiplications and sums the results when all of the products are completed. The scaling accumulator shifts the sums of partial products according to the appropriate number of bits and accumulates the result to provide the final multiplier output.

Distributed Arithmetic in LUTs

Figure 3 shows how to implement distributed arithmetic using LUTs. The combined product and adder tree are reduced for the LUT implementation. In this example, the LUT contains the sums of constant coefficients for all possible input combinations to the LUT. The sums of the bits from the LUTs are added together in the scaling accumulator and shifted by the appropriate weights.

Figure 3. Four-Bit Multiplication with Constant Coefficients   *Note (1)*
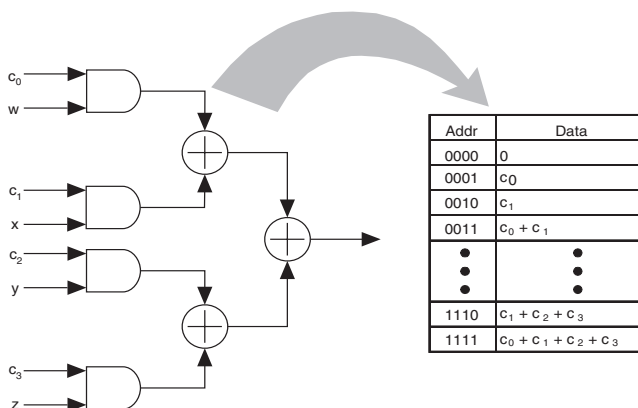
Note to Figure 3:
(1) c_0 to c_3 are constant coefficients.

The addressing method and data values stored in the LUT in Figure 3 apply to the sum of multiplication operation mode. The addressing method and LUT data values vary depending on the multiplier implementation mode.

Implementing Soft Multipliers Using Memory Blocks

You can use the Stratix II, Stratix, and Stratix GX M512 or M4K RAM memory blocks and Cyclone II and Cyclone M4K RAM memory blocks as LUTs to implement multiplication for DSP applications. Combinations of the coefficient results are pre-calculated and stored in the M512 or M4K RAM blocks as a LUT. The address port of the RAM block represents one of the multiplication operands. The content of the RAM block at each address represents a unique multiplication result calculated between the input operand and a known coefficient value based on the multiplier mode implemented.

The five soft multiplier modes supported by Stratix II, Stratix, Stratix GX, Cyclone II, and Cyclone devices are:

■   Parallel multiplication—Multiple memory blocks produce one multiplication result every clock cycle. This mode is useful for high-speed data scaling.

■   Semi-parallel multiplication—Each memory block produces one multiplication with multi-cycle operation. This mode is useful for coefficient update of least mean squares (LMSs) and coefficient update of equalizers.

■ Sum of multiplication—One memory block or group of memory blocks produces the sum of multiplication results. This mode is useful in applications such as finite impulse response (FIR) filtering and discrete cosine transforms (DCTs).

■ Hybrid multiplication—Combination and optimization of semi-parallel and sum of multiplication modes of operation. This mode is ideal for a complex number of multiplications in complex fast Fourier transforms (FFTs) and infinite impulse response (IIR) filters.

■ Fully variable multiplication—This mode is useful for a soft multiplier implementations in which both the input data and coefficients are varying. This mode is ideal for low-resolution multiplication functions.

The following sections describe each of these modes and provide examples.

Parallel Multiplication

Parallel multiplication involves multiplying all sections of a single input bus or multiplier value with a single multiplicand or coefficient and summing the partial product of each multiplication to obtain the final result. All of the input bits are parallel-loaded into the RAM block address port registers and a new multiplication is completed each clock cycle. For example, a 16-bit input bus can be separated into two groups of eight bits (one group of eight least significant bits [LSBs] and another group of eight most significant bits [MSBs]) and simultaneously shifted into the address ports of two RAM blocks. The output of the RAM blocks indicate the multiplication result for the particular set of bits with the coefficient. Figure 4 represents the decomposition of a 16-bit data input, 10-bit constant coefficient parallel multiplier.

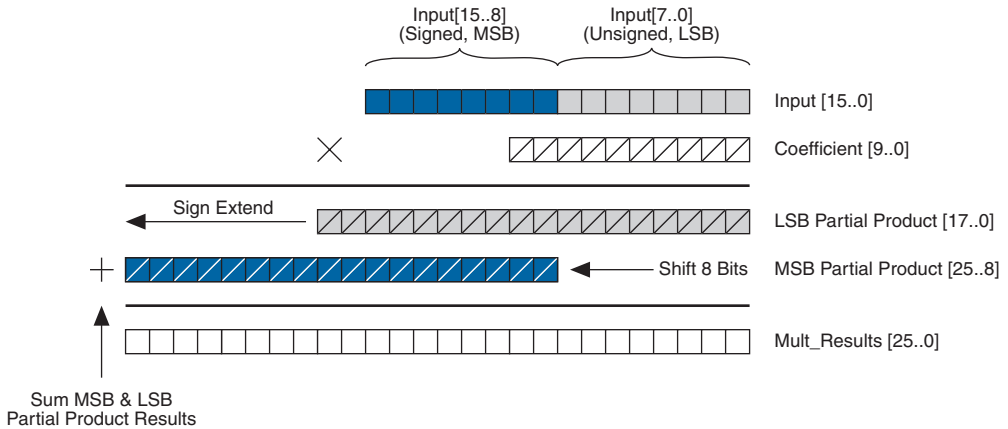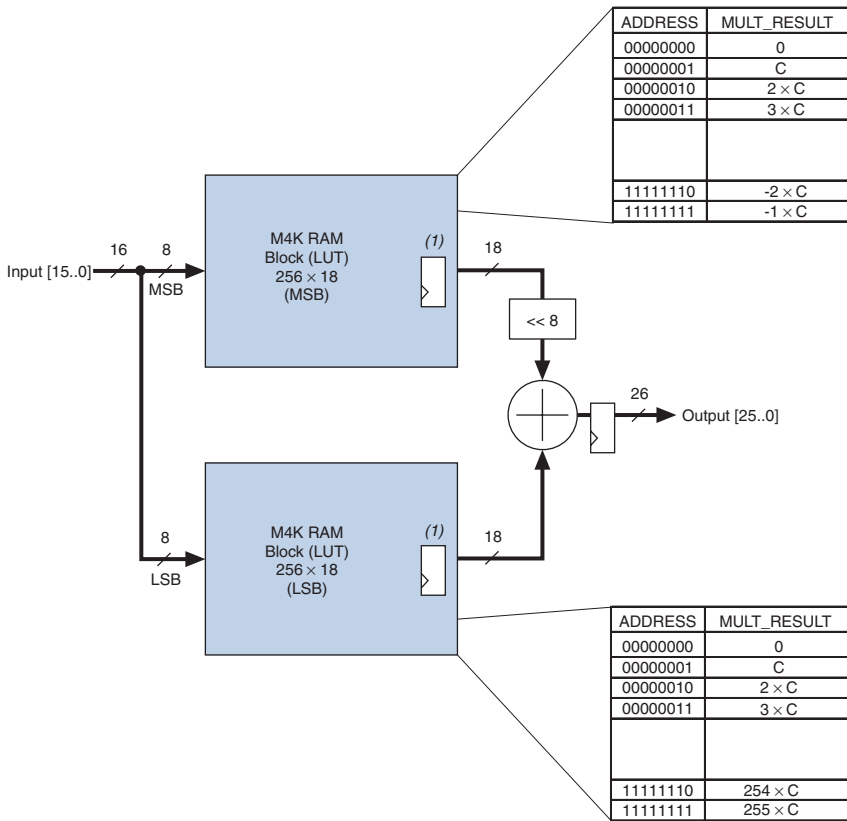*Figure 4. Decomposition of a 16-Bit Input, 10-Bit Coefficient Parallel Multiplier*

Figure 5 shows the RAM LUT implementation of the parallel multiplier decomposition shown in Figure 4. Because a parallel multiplier accepts a new input every clock cycle, this implementation takes three clock cycles (one to load the input values into the RAM block address ports and two pipeline delays) to compute the final multiplication result. New partial products are obtained from the RAM blocks every clock cycle and the partial products are summed according to their weights. Each partial product multiplication generates an output of 18 bits. At the end of the partial product accumulation, the multiplier generates a 26-bit output.

Figure 5. 16-Bit Input, 10-Bit Coefficient Parallel Multiplication Implementation Using M4K RAM Blocks as LUTs   *Note (1)*

ADDRESS	MULT_RESULT
00000000	0
00000001	C
00000010	$2 \times C$
00000011	$3 \times C$
11111110	$-2 \times C$
11111111	$-1 \times C$

ADDRESS	MULT_RESULT
00000000	0
00000001	C
00000010	$2 \times C$
00000011	$3 \times C$
11111110	$254 \times C$
11111111	$255 \times C$

Note to Figure 5:

(1)   This is an optional pipeline register to increase system performance.

Figure 5 shows an implementation for a 16-bit data input, split into two 8-bit sections implemented using two M4K RAM blocks, one for the MSB section and the other for the LSB section. For signed input buses, the M4K RAM block that accepts the MSB bits must contain precalculated coefficient values for signed inputs because the eight MSB bits that feed this RAM block are treated as signed values. The M4K RAM block that accepts the LSB bits must contain precalculated coefficient values for unsigned inputs because the eight LSB bits that feed this RAM blocks are unsigned values.

M4K RAM blocks are 256×18 bits, so the maximum number of bits per section for each M4K RAM block for this coefficient size is eight ($2^8 = 256$ addresses). The input bus and coefficient size directly affects the number and configuration of RAM blocks used to implement the multiplier. The parallel multiplication mode ensures maximum data throughput (i.e., a new data value every clock cycle).

You can also implement the parallel fixed-coefficient multiplier using the `altmemmult` Quartus II megafunction. You can use the MegaWizard® Plug-In Manager to customize the `altmemmult` megafunction to specify a parallel, fixed coefficient soft multiplier in your design. The input and coefficient bit width settings as well as RAM block selection type determine whether the `altmemmult` function implements a semi-parallel or parallel mode soft multiplier, whichever is more efficient. Figures 6 and 7 show the appropriate settings required to implement both the MSB and LSB M4K RAM blocks respectively, for the 16-bit input, 10-bit parallel multiplier example shown in Figure 14. The coefficient implemented in this example is a constant value of five.

Figure 6. altmemmult MegaWizard Settings for the MSB RAM Block 16-Bit Input, 10-Bit Constant Coefficient Parallel Multiplier
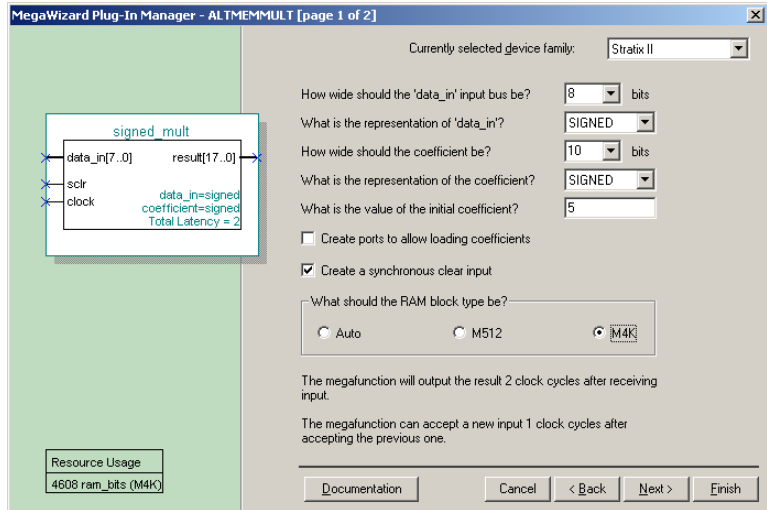
Figure 7. altmemmult MegaWizard Settings for the LSB RAM Block for a 16-Bit Input, 10-Bit Constant Coefficient Parallel Multiplier
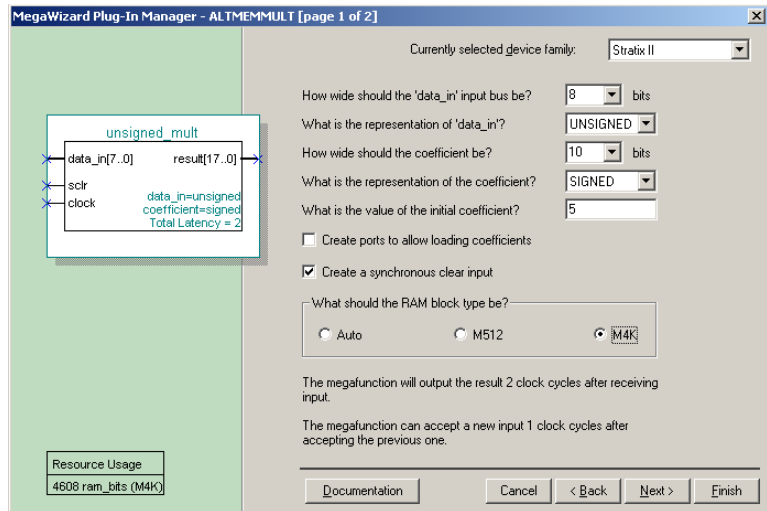
The sload_data signal and the message located at the bottom right hand corner of the MegaWizard window indicates whether the altmemmult function chose to implement a semi-parallel or parallel mode soft multiplier. A parallel soft multiplier does not have the sload_data signal and the megafunction can accept a new input every clock cycle. The altmemmult megafunction can only implement small parallel mode soft multipliers (i.e., 8-bit input, 10-bit coefficient multipliers). Larger parallel multipliers require multiple altmemmult megafunctions to generate partial product results. To obtain the final multiplication result, these partial products must be summed in an end-stage adder implemented externally to the altmemmult function.

Fixed-Coefficient Multiplication

Figure 8 shows the simulation results for the example shown in Figure 5. This example multiplies the input, which has a decimal value of 297, with a coefficient, which has a value of 5.

Figure 8. Parallel Multiplication Simulation Results
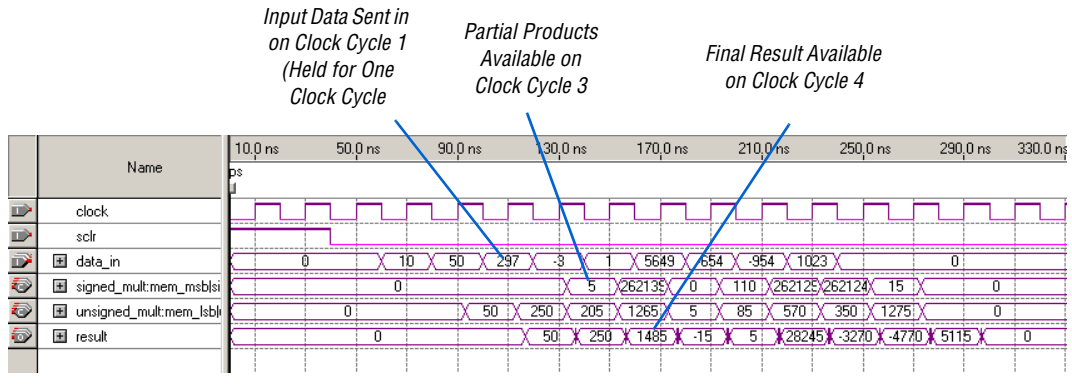
Tables 16 and 17 shows the implementation results for the parallel fixed coefficient multiplication example shown in Figure 5 for Stratix II and Stratix devices, respectively. The example is implemented using the `altmemmult` megafunction.

Table 16. 16-Bit Input, 10-Bit Constant Coefficient Parallel Multiplication Implementation Results Using Stratix II Devices	
Device	EP2S15F484C3
Utilization	ALUTs: 26/12,480 (<1%) M4K RAM blocks: 2/78 (2%)
Latency *(1)*	3 clock cycles
Throughput	351 megasamples per second
Performance	351.0 MHz

Note to Table 16:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

Table 17. 16-Bit Input, 10-Bit Constant Coefficient Parallel Multiplication Implementation Results Using Stratix Devices (Part 1 of 2)	
Device	EP1S10F484C5
Utilization	Logic cells: 26/10,570 (1%) M4K RAM blocks: 2/60 (3%)
Latency *(1)*	3 clock cycles
Throughput	291 megasamples per second

Table 17. 16-Bit Input, 10-Bit Constant Coefficient Parallel Multiplication Implementation Results Using Stratix Devices (Part 2 of 2)	
Performance	291.0 Mhz

Note to Table 17:

(1) Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**parallel_fixed.zip**) for the design described in Tables 16 and 17 from the Design Examples section of the Altera web site **www.altera.com.**

Variable Coefficient Multiplication

To perform constant coefficient multiplication, you can implement the Stratix II, Stratix, Stratix GX, Cyclone II, and Cyclone memory blocks as ROM. For variable coefficient multiplication, these memory blocks must be implemented as RAM blocks, which allow you to rewrite blocks with new precalculated coefficients. Figure 9 shows an implementation for variable coefficient parallel multiplication implementation using M4K single-port RAM blocks. Using the method shown in Figure 9, the multiplier function is stalled while the coefficients are updated. However, by implementing multiple sets of RAM blocks for storing different precalculated coefficient sets, you can switch multiplication between two different sets of coefficients in a single clock cycle. One way of doing this is to partition the RAM block to store two unique sets of coefficients and to use the MSB address bit to select which coefficient set to use. Also, with the use of dual-port RAM blocks, you can write or update the values of a set of coefficients in a partition while simultaneously using a different set of coefficients in another partition to perform multiplication.

Figure 9. 16-Bit Input, 10-Bit Variable Coefficient Parallel Multiplication Implementation Using M4K Single-Port RAM Blocks as LUTs

ADDRESS	MULT_RESULT
00000000	0
00000001	C
00000010	$2 \times C$
00000011	$3 \times C$
11111110	$-2 \times C$
11111111	$-1 \times C$

ADDRESS	MULT_RESULT
00000000	0
00000001	C
00000010	$2 \times C$
00000011	$3 \times C$
11111110	$254 \times C$
11111111	$255 \times C$
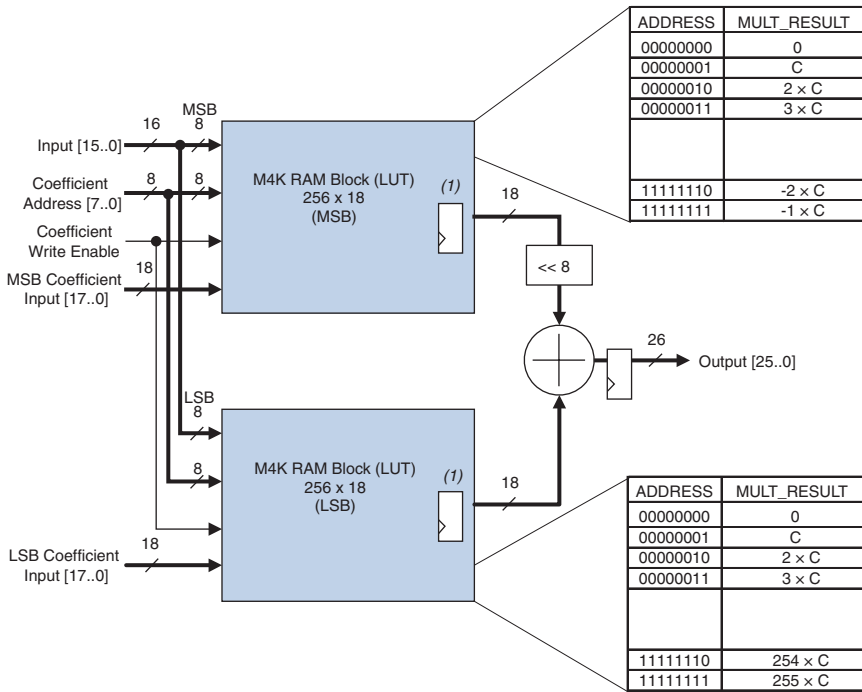
Note to Figure 9:
(1)   This is an optional pipeline register to increase system performance.

The `altmemmult` megafunction also supports variable coefficient parallel and semi-parallel soft multipliers by enabling the **Create ports to allow loading coefficients** option in the MegaWizard window.

Tables 18 and 19 show the implementation results for a parallel variable coefficient multiplication example for Stratix II and Stratix devices, respectively.

Table 18. 16-Bit Input, 10-Bit Variable Coefficient Parallel Multiplication Implementation Results Using Stratix II Devices	
Device	EP2S15F484C3
Utilization	ALUTs: 43/12,480 (<1%) M4K RAM blocks: 2/78 (3%)
Latency *(1)*	3 clock cycles
Throughput	350 megasamples per second
Performance	350.0 MHz

Note to Table 18:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

Table 19. 16-Bit Input, 10-Bit Variable Coefficient Parallel Multiplication Implementation Results Using Stratix Devices	
Device	EP1S10F484C5
Utilization	Logic cells: 43/10,570 (<1%) M4K RAM blocks: 2/60 (3%)
Latency *(1)*	3 clock cycles
Throughput	291 megasamples per second
Performance	291.0 MHz

Note to Table 19:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**parallel_var.zip**) for the design described in Tables 18 and 19 from the Design Examples section of the Altera web site **(www.altera.com).**

Semi-Parallel Multiplication

Semi-parallel multiplication involves multiplying sections of a single input bus or multiplier value with a single multiplicand or coefficient and shift accumulating the partial product of each multiplication to obtain the final result. For example, a 16-bit input bus can be separated into four groups of four bits that are consecutively shifted into the address port of the RAM block once every clock cycle, beginning with the first four LSB bits. The output of the RAM block indicates the multiplication result for

a particular set of bits with the coefficient, every clock cycle. Figure 10 shows the decomposition of a 16-bit data input, 14-bit coefficient semi-parallel multiplier.

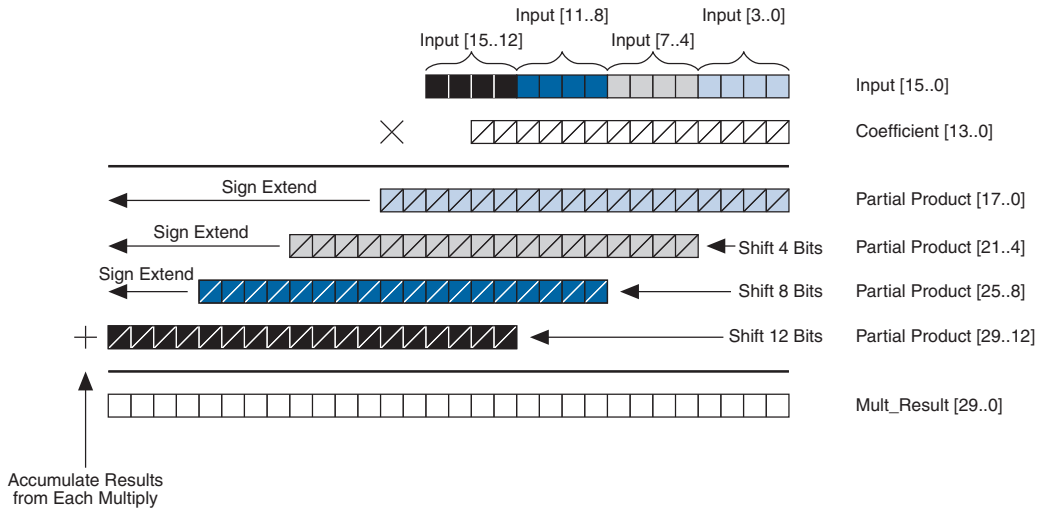*Figure 10. Decomposition of a 16-Bit Input, 14-Bit Coefficient Semi-Parallel Multiplier*

Figure 11 shows the RAM LUT implementation of the semi-parallel multiplier decomposition shown in Figure 10. This implementation loads four bits of the input data every clock cycle, taking six clock cycles (four to load the input values into the RAM block plus two pipeline delays) to complete the multiplication operation by shift-accumulating the partial products obtained from the RAM block once per clock cycle, according to their weights. Each shift-accumulation of a partial product generates four extra bits. At the end of the fourth partial product accumulation, the multiplier generates a 30-bit output.

Figure 11. 16-Bit Input, 14-Bit Coefficient Semi-Parallel Multiplication Implementation Using M512 RAM Blocks as LUTs *Note (1)*

ADDRESS	MULT_RESULT
0000	0
0001	C
0010	$2 \times C$
0011	$3 \times C$
•	•
•	•
•	•
1110	$14 \times C$
1111	$15 \times C$
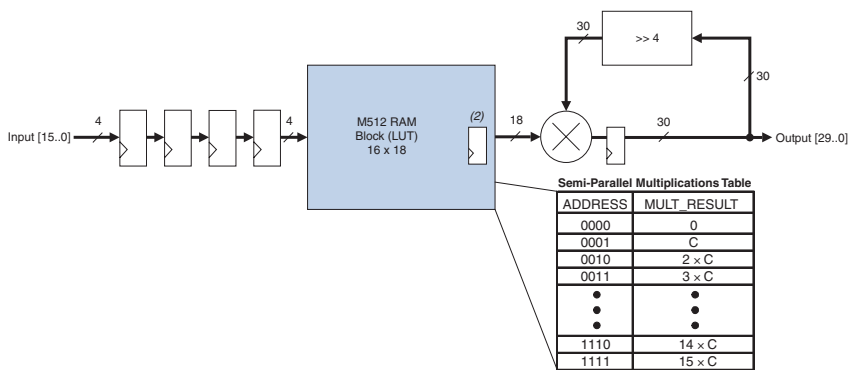
Notes to Figure 11:
(1)   The input bus is 16 bits wide, but it is sent to the RAM block 4 bits at a time. This is why the bus line is only 4 bits wide.
(2)   This is an optional pipeline register to increase system performance.

Figure 11 shows an implementation for a 16-bit data input, split into four 4-bit sections implemented using a single M512 RAM block. In this example, for the same memory block utilization, factors like the input bus size help determine the output bit width and the latency of the multiplier. Increasing the bit width of the sections (i.e., implementing more than 4-bit sections in this case) can reduce the latency of the multiplier. This implementation may require more M512 RAM blocks or that you use M4K RAM blocks.

You can also implement the semi-parallel fixed coefficient multiplier using the `altmemmult` Quartus II megafunction. You can use the MegaWizard Plug-In Manager to customize the `altmemmult` megafunction to specify a semi-parallel, fixed coefficient soft multiplier in your design. The input and coefficient bit width settings as well as RAM block selection type determine whether the `altmemmult` function implements a semi-parallel or parallel mode soft multiplier; it implements whichever is more efficient. Figure 12 shows the settings required to implement the 16-bit input, 14-bit semi-parallel multiplier example shown in Figure 11. The coefficient implemented in this example is a constant value of two.

Figure 12. altmemmult MegaWizard Settings for a 16-Bit Input, 14-Bit Constant Coefficient Semi-Parallel Multiplier
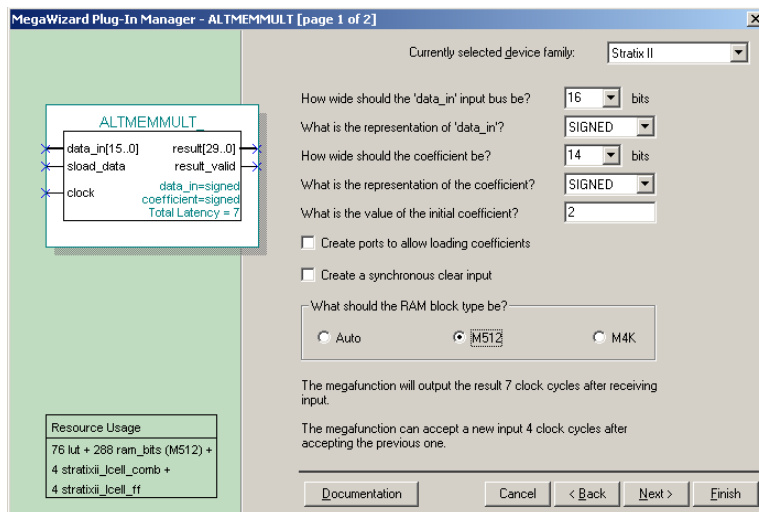

The `sload_data` signal and the message located at the bottom right-hand corner of the MegaWizard window indicate whether the `altmemmult` function chose to implement a semi-parallel or parallel mode soft multiplier. A semi-parallel soft multiplier has an `sload_data` signal and can only accept a new input after more than one clock cycle. The semi-parallel multiplier in Figure 11 indicates that the 16-bit input is split into four groups of four bits each. Because it takes four clock cycles to load the entire 16-bits into the RAM block, the current input must remain stable for four clock cycles prior to loading the new input. A high signal on `sload_data` for one clock cycle indicates the start of a new block of input data.

For information on implementing variable coefficient soft multipliers, refer to the "Variable Coefficient Multiplication" on page 18.

Figure 13 shows the simulation results for the example shown in Figure 11. This example multiplies the input, which has a decimal value of 10, with a coefficient, which has a value of 2.

Figure 13. Semi-Parallel Simulation Results
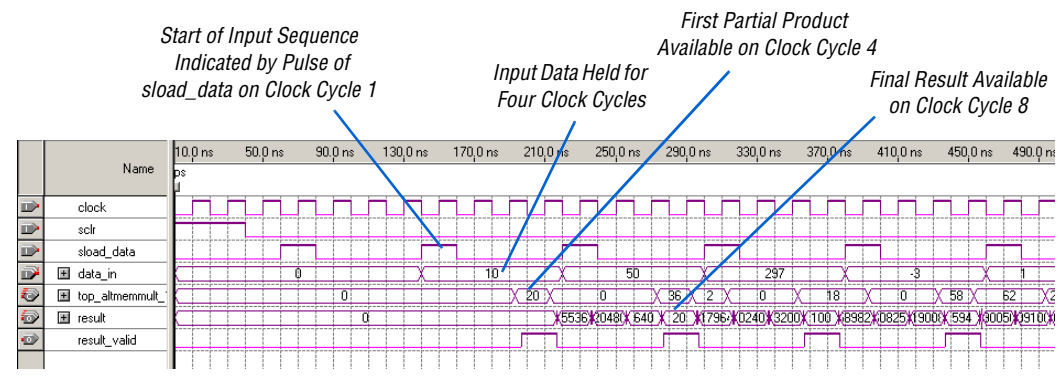
Tables 20 and 21 show the implementation result for the semi-parallel fixed coefficient multiplication example shown in Figure 11 for Stratix II and Stratix devices, respectively.

Table 20. 16-Bit Input, 14-Bit Constant Coefficient Semi-Parallel Multiplication Implementation Results Using Stratix II Devices	
Device	EP2S15F484C3
Utilization	ALUTs: 67/12,480 (<1%) M512 RAM blocks: 1/104 (<1%)
Latency *(1)*	7 clock cycles
Throughput	93 megasamples per second
Performance	370.0 MHz

Note to Table 20:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

Table 21. 16-Bit Input, 14-Bit Constant Coefficient Semi-Parallel Multiplication Implementation Results Using Stratix Devices

Device	EP1S10F484C5
Utilization	Logic cells: 61/10,570 (1%) M512 RAM blocks: 1/94 (2%)
Latency *(1)*	7 clock cycles
Throughput	80 megasamples per second
Performance	321.0 MHz

Note to Table 21:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**semi_prl_fixed.zip**) for the design described in Tables 20 and 21 from the Design Examples section of the Altera web site (**www.altera.com**).

Tables 22 and 23 show the implementation results for a semi-parallel variable coefficient multiplication example for Stratix II and Stratix devices, respectively.

Table 22. 16-Bit Input, 14-Bit Variable Coefficient Semi-Parallel Multiplication Implementation Results Using Stratix II Devices

Device	EP2S15F484C3
Utilization	ALUTs: 94/12,480 (<1%) M512 RAM blocks: 1/104 (<1%)
Latency *(1)*	7 clock cycles
Throughput	78 megasamples per second
Performance	310.0 MHz

Note to Table 22:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

Table 23. 16-Bit Input, 14-Bit Variable Coefficient Semi-Parallel Multiplication Implementation Results Using Stratix Devices	
Device	EP1S10F484C5
Utilization	Logic cells: 118/10,570 (1%) M512 RAM blocks: 1/94 (2%)
Latency *(1)*	7 clock cycles
Throughput	65 megasamples per second
Performance	261.0 MHz

Note to Table 23:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**semi_prl_var.zip**) for the design described in Tables 22 and 23 from the Design Examples section of the Altera web site (**www.altera.com**).

Sum of Multiplication

The sum of multiplication mode result is the weighted summation of results produced by multiplying a set of input data (multiplier) to a set of multiplicands. This sum forms the basis of a MAC function that is useful in functions such as FIR filters, where each input data (multiplier) value is multiplied with a particular coefficient (or multiplicand) and summed to provide the final result.

In the sum of multiplication mode, each input bus shifts into the address port of the memory block one bit per clock cycle, starting with the LSB. If there are four inputs (called A, B, C, and D) to the multiplier block, at the first clock cycle, the LSB of inputs A, B, C, and D forms the 4-bit address value to the RAM block. The next clock cycle, the second LSB bit for each input forms the next address value to the RAM block, and so on. For an *n*-bit input data width, it takes *n* clock cycles to load in all of the data bits required to compute the multiplication result. The RAM block output indicates the multiplication result for a specific bit position at each clock cycle.

Figure 14 shows the RAM LUT implementation of four 4-bit data inputs and up to 16-bit constant coefficients. This fixed coefficient implementation takes six clock cycles (four to load the input values into the RAM block plus two pipeline delays) to complete the multiplication operation by shift-accumulating the partial products obtained from the RAM block once per clock cycle, according to their weights. Each shift-accumulation of a partial product generates an extra carry bit. At the end

of the fourth partial product accumulation, the multiplier generates a 22-bit output. The size of the input data helps determine the output bit width and the latency of the multiplier.

Figure 14. 4-Input Sum of Multiplication Implementation Using M512 RAM Blocks as LUTs
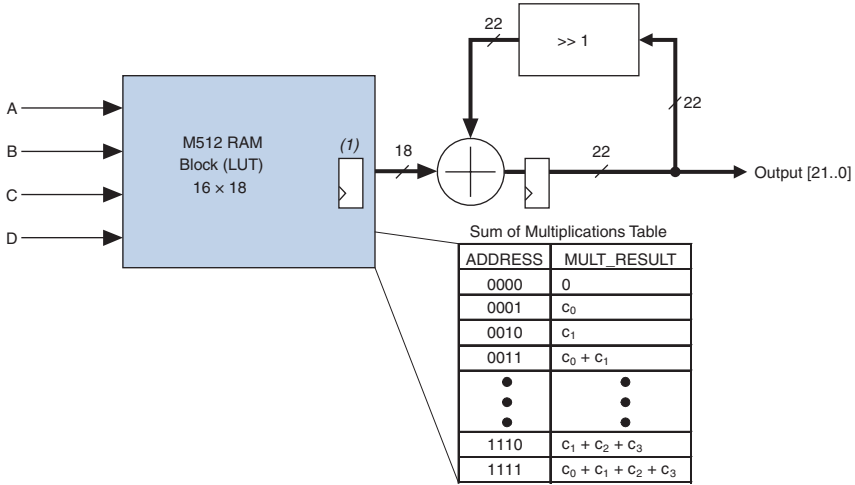
Note to Figure 14:
(1)   This is an optional pipeline register to increase system performance.

Figure 15 shows the equivalent circuit of the sum of multiplication implementation shown in Figure 14.

Figure 15. Equivalent Circuit of a Four Multiplier Sum of Multiplication Function
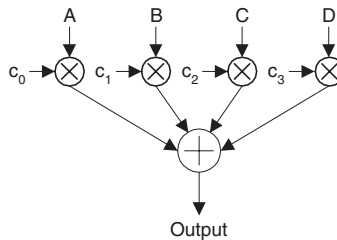
Figure 14 shows an implementation for four 4-bit data inputs. Because M512 RAM blocks are 32×18 bits, the maximum number of inputs for each M512 RAM block for this coefficient size is five ($2^5 = 32$ addresses).

Depending on the number of inputs, size and number of coefficients, and the required operating speed, the number of RAM blocks used varies. The example shown in Figure 14 requires only one M512 RAM block.

For information on implementing variable coefficient soft multipliers, refer to "Variable Coefficient Multiplication" on page 18.

Figure 16 shows the simulation result for an example based on Figure 14. This example has additional pipeline stages and multiplies input A, which has a binary value of 0001, with the c0 coefficient, which has a value of -3.

☞   You can choose to reduce the number of pipeline stages to reduce the latency, but your design may have reduced f_{MAX} as a result.

Figure 16. Sum of Multiplication Simulation Results
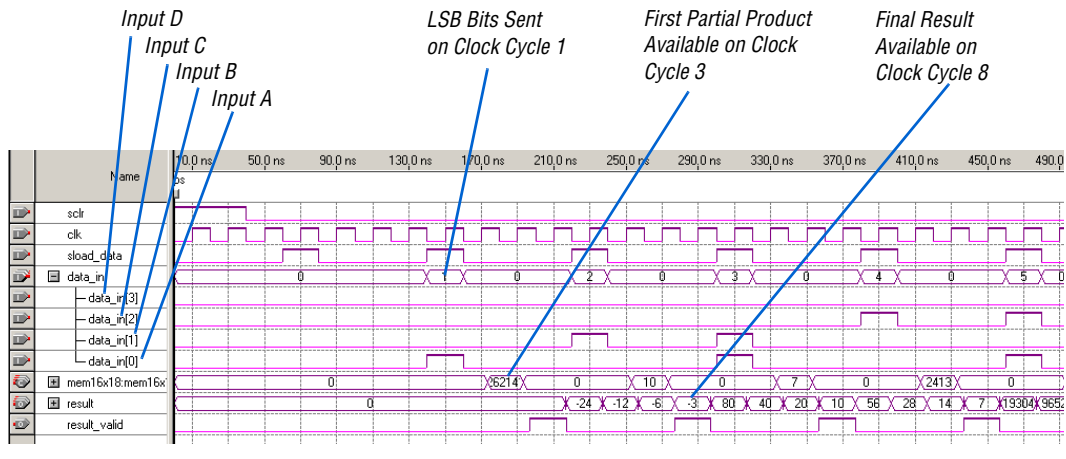
Tables 24 and 25 show the implementation results of the four input, 16-bit fixed coefficient sum of multiplication example shown in Figure 14 for Stratix II and Stratix devices, respectively.

Table 24. 4-Input, 16-Bit Fixed Coefficient Sum of Multiplication Implementation Results Using Stratix II Devices (Part 1 of 2)	
Device	EP2S15F484C3
Utilization	ALUTs: 80/12,480 (<1%) M512 RAM blocks: 1/104 (<1%)

Table 24. 4-Input, 16-Bit Fixed Coefficient Sum of Multiplication Implementation Results Using Stratix II Devices (Part 2 of 2)	
Latency *(1)*	7 clock cycles
Throughput	66 megasamples per second
Performance	265.0 MHz

Note to Table 24:
(1)   Latency is the number of clock cycles required to complete an entire sum of multiplication computation.

| Table 25. 4-Input, 16-Bit Fixed Coefficient Sum of Multiplication Implementation Results Using Stratix Devices	
Device	EP1S10F484C5
Utilization	Logic cells: 67/10,570 (1%) M512 RAM blocks: 1/94 (2%)
Latency *(1)*	7 clock cycles
Throughput	53 megasamples per second
Performance	212.0 MHz

Note to Table 25:
(1)   Latency is the number of clock cycles required to complete an entire sum of multiplication computation.

You can download the files (**sum_mult_fixed.zip**) for the design described in Tables 24 and 25 from the Design Examples section of the Altera web site (**www.altera.com**).

Tables 26 and 27 show the implementation results of a four input, 16-bit variable coefficient sum of multiplication example for Stratix II and Stratix devices, respectively.

Table 26. Four Input, 16-Bit Variable Coefficient Sum of Multiplication Implementation Results Using Stratix II Devices

Device	EP2S15F484C3
Utilization	ALUTs: 88/12,480 (<1%) M512 RAM blocks: 1/104 (<1%)
Latency *(1)*	8 clock cycles
Throughput	76 megasamples per second
Performance	305.0 MHz

Note to Table 26:
(1)   Latency is the number of clock cycles required to complete an entire sum of
        multiplication computation.

Table 27. Four Input, 16-Bit Variable Coefficient Sum of Multiplication Implementation Results Using Stratix Devices

Device	EP1S10F484C5
Utilization	Logic cells: 96/10,570 (1%) M512 RAM blocks: 1/94 (1%)
Latency *(1)*	8 clock cycles
Throughput	48 megasamples per second
Performance	191.0 MHz

Note to Table 27:
(1)   Latency is the number of clock cycles required to complete an entire sum of
        multiplication computation.

You can download the files (**sum_mult_var.zip**) for the design described in Tables 26 and 27 from the Design Examples section of the Altera web site (**www.altera.com**).

You can combine multiple M512 blocks and/or M4K blocks to create larger multiplier structures that are capable of multiplying more data inputs and coefficients simultaneously. Figure 17 shows the multiplication of eight 4-bit data inputs to eight 16-bit constant coefficients in two M512 RAM blocks.
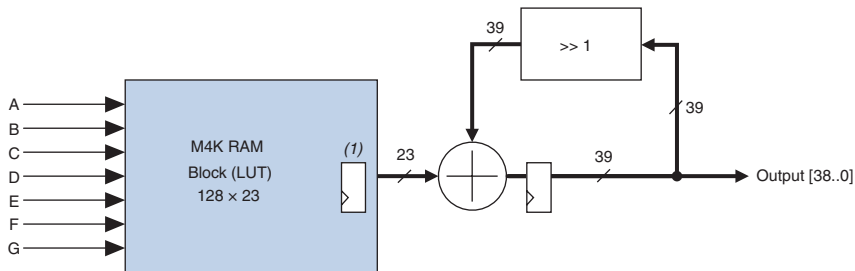
Figure 17. Using Multiple M512 RAM Blocks for an 8-Coefficient Multiplier
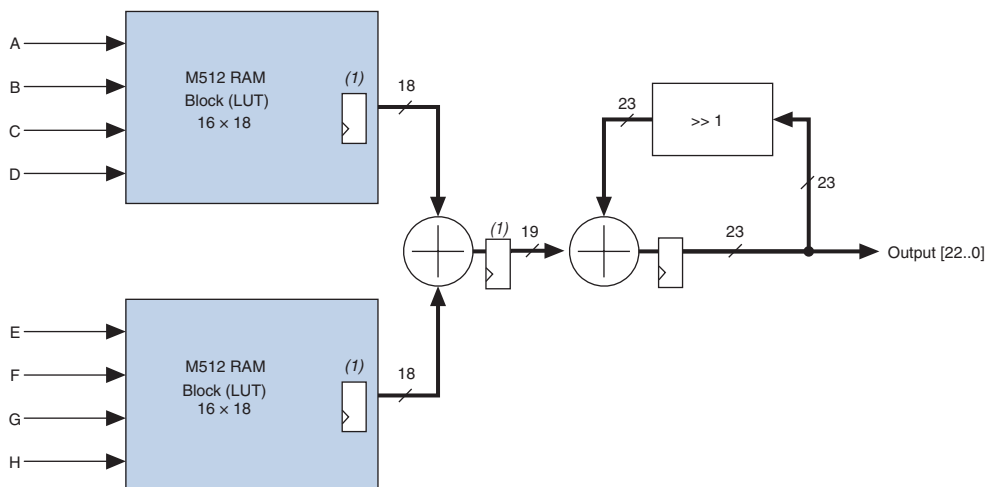
Note to Figure 17:

(1)   This is an optional pipeline register to increase system performance.

For information on implementing variable coefficient soft multipliers, refer to "Variable Coefficient Multiplication" on page 18.

You can also create similar implementations using M4K RAM blocks, particularly if the coefficients are larger than 16 bits. Figure 18 shows multiplication of seven 16-bit data inputs to a 20-bit constant coefficient in one M4K RAM block. The 128 addressed lines correspond to seven data inputs or unique coefficients in a M4K RAM block. Performing seven 16×20 multiplications generates a 23-bit output from a M4K RAM block. It takes 18 clock cycles to complete accumulation of the partial products (16 clock cycles to shift the input values into the address port of the RAM block plus two pipeline delays). After each partial product accumulation, one bit is added to the total number of output bits, making the final output 39 bits wide.

Figure 18. Using a M4K RAM Block for a 7-Coefficient Multiplier

Note to Figure 18:
(1)   This is an optional pipeline register to increase system performance.

For information on implementing variable coefficient soft multipliers, refer to "Variable Coefficient Multiplication" on page 18.
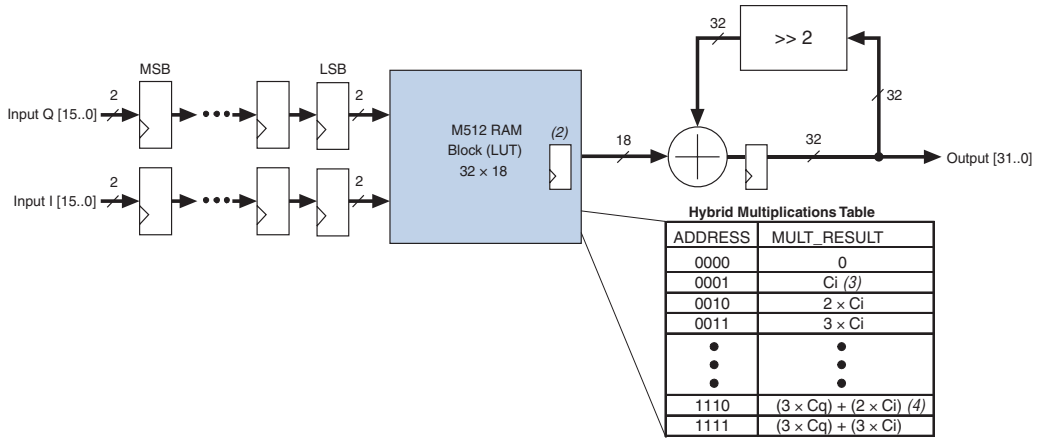
Hybrid Multiplication

The hybrid multiplication mode is a combination of the semi-parallel and sum of multiplication modes where bit sections from two unique input streams are multiplied with two different coefficients values. This mode is useful in applications that require complex multiplication like FFTs where each signal generally has a real and imaginary component that could be multiplied by two unique coefficient values. The partial products obtained from each bit section within the components are shift accumulated to obtain the final result.

In the hybrid multiplication mode, an equal number of bits from each input is concatenated and shifted into the address port of the RAM block every clock cycle, starting with the LSB. If the address port to the RAM block is four bits wide, each input contributes two bits to the partial product calculation every clock cycle until the entire bit width of the inputs have completely shifted into the RAM block. In this case, for an input bus of 16-bits, it takes eight clock cycles to shift in all of the data bits of that particular input. The output of the RAM block indicates the sum of multiplication result for a particular set of bits with the coefficients, every clock cycle.

Figure 19 shows the RAM LUT implementation of two 16-bit inputs, labeled Input I and Input Q, respectively, and up to 15-bit constant coefficients. This implementation takes 11 clock cycles (eight to load the input values into the RAM block plus three pipeline delays) to complete the multiplication operation by shift-accumulating the partial products obtained from the RAM once per clock cycle, according to their weights. Each shift-accumulation of a partial product generates two extra bits. At

the end of the last (eighth) partial product accumulation, the multiplier generates a 32-bit output. The size of the input data helps determine the output bit width and the latency of the multiplier.

Figure 19. Two-Input Hybrid Multiplication Implementation Using M512 RAM Blocks as LUTs   *Note (1)*

Hybrid Multiplications Table

ADDRESS	MULT_RESULT
0000	0
0001	Ci *(3)*
0010	2 × Ci
0011	3 × Ci
•	•
•	•
•	•
1110	(3 × Cq) + (2 × Ci) *(4)*
1111	(3 × Cq) + (3 × Ci)

Notes to Figure 19:
(1) The input bus is 16 bits wide, but it is sent to the RAM block 2 bits at a time. This is why the bus line is only 2 bits wide.
(2) Optional pipeline register to increase system performance.
(3) Ci means I Coefficient.
(4) Cq means Q Coefficient.

Figure 19 shows an implementation for two 16-bit data inputs. Even though the 32 × 18-bit configured M512 RAM block can accept five address bits (2^5 = 32 addresses), the maximum number of bits equally contributed by each input is two bits (totaling four bits). In this example, for the same memory block utilization, factors such as the input bus size help determine the output bit width and the latency of the multiplier. Increasing the number of M512 RAM blocks used or moving to larger memory blocks like M4K RAM blocks can reduce the latency of the multiplier an support larger coefficient bit widths.

For information on implementing variable coefficient soft multipliers, refer to "Variable Coefficient Multiplication" on page 18.
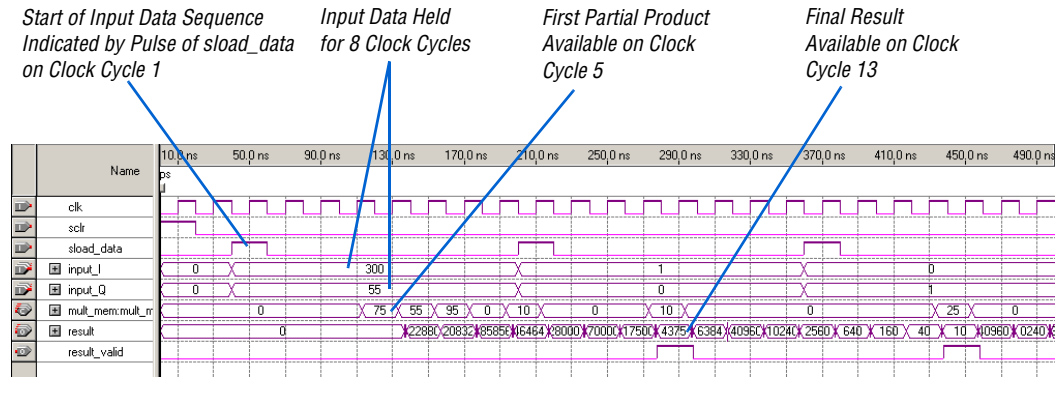
Figure 20 shows the simulation results for an example based on Figure 19. This example has additional pipeline stages and multiplies Input_I and Input_Q , which have values of 300 and 55, respectively, with coefficients Ci and Cq, which have values of 10 and 25, respectively. The result is:

(Input_I × Ci) + (Input_Q × Cq) = (300 × 10) + (55 × 25) = 4375

☞ You can choose to reduce the number of pipeline stages to reduce the latency, but your design may have reduced f_{MAX} as a result.

Figure 20. Hybrid Multiplication Simulation Results

Start of Input Data Sequence Indicated by Pulse of sload_data on Clock Cycle 1

Input Data Held for 8 Clock Cycles

First Partial Product Available on Clock Cycle 5

Final Result Available on Clock Cycle 13

Tables 28 and 29 show the implementation results of the two 16-bit input, 15-bit constant coefficient hybrid multiplication example shown in Figure 19 for Stratix II and Stratix devices, respectively.

Table 28. Two Input, 15-Bit Constant Coefficient Hybrid Multiplication Implementation Results Using Stratix II Devices	
Device	EP2S15F484C3
Utilization	ALUTs: 126/12,480 (1%) M512 RAM blocks: 1/104 (<1%)
Latency *(1)*	12 clock cycles
Throughput	33 megasamples per second
Performance	266.0 MHz

Note to Table 28:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

Table 29. Two Input, 15-Bit Constant Coefficient Hybrid Multiplication Implementation Results Using Stratix Devices

Device	EP1S10F484C5
Utilization	Logic cells: 170/10,570 (2%) M512 RAM blocks: 1/94 (1%)
Latency (1)	12 clock cycles
Throughput	25 megasamples per second
Performance	198.0 MHz

Note to Table 29:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**hybrid_fixed.zip**) for the design described in Tables 28 and 29 from the Design Examples section of the Altera web site (**www.altera.com**).

Tables 30 and 31 show the implementation results for a hybrid variable coefficient multiplication example for Stratix II and Stratix devices, respectively.

Table 30. Two Input, 15-Bit Variable Coefficient Hybrid Multiplication Implementation Results Using Stratix II Devices

Device	EP2S15F484C3
Utilization	ALUTs: 213/12,480 (2%) M512 RAM blocks: 1/104 (<1%)
Latency (1)	12 clock cycles
Throughput	32 megasamples per second
Performance	254.0 MHz

Note to Table 30:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

Table 31. Two Input, 15-Bit Variable Coefficient Hybrid Multiplication Implementation Results Using Stratix Devices

Device	EP1S10F484C5
Utilization	Logic cells: 240/10,570 (2%) M512 RAM blocks: 1/94 (1%)
Latency *(1)*	12 clock cycles
Throughput	25 megasamples per second
Performance	202.0 MHz

Note to Table 31:
(1) Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**hybrid_var.zip**) for the design described in Tables 30 and 31 from the Design Examples section of the Altera web site at (**www.altera.com**).

Fully Variable Multipliers

The fully variable multiplier mode allows you to implement a soft multiplier in which both the input and the coefficient can vary every clock cycle. The partial product values, which are stored in the RAM blocks, are calculated based on the algebraic expansion of the following equation:

$$(a + b)^2 - (a - b)^2 \quad = \quad a^2 + 2ab + b^2 - (a^2 - 2ab + b^2)$$

$$= \quad 4ab$$

therefore:

$$ab = ((a + b)^2 \, / \, 4) - ((a - b)^2 \, / \, 4)$$

Where a and b are both variable inputs to the multiplier.

Figure 21 shows the RAM LUT implementation of the fully variable multiplier calculated using these equations. Two unique RAM blocks are required, to store the $(a + b)^2/4$ and $(a - b)^2/4$ precalculated values, respectively. The address inputs of $(a + b)$ for the former and $(a - b)$ for the latter RAM block are precalculated in logic prior to the RAM block. The final result of the multiplication is obtained by subtracting the result of the $(a - b)$ RAM block by the result from the $(a + b)$ RAM block. The fully variable multiplier can accept a new input every clock cycle, and takes three clock cycles to compute the final multiplication result.

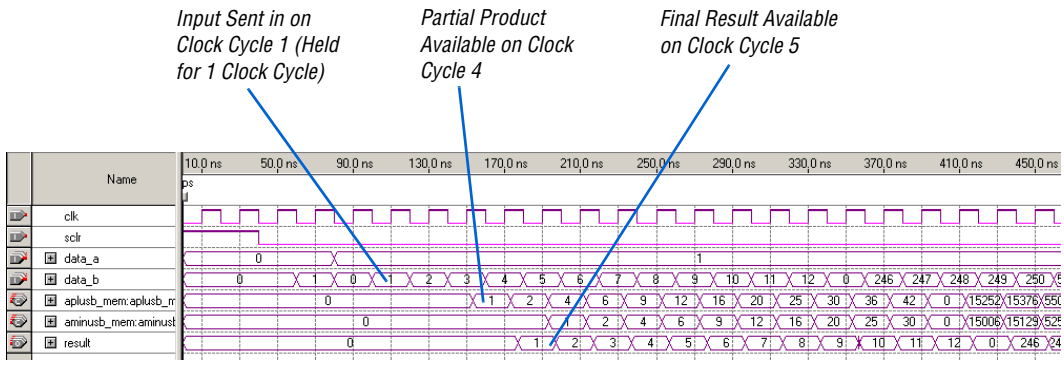*Figure 21. 8-Bit Fully Variable Multiplier Implementation Using M4K RAM Blocks as LUTs*
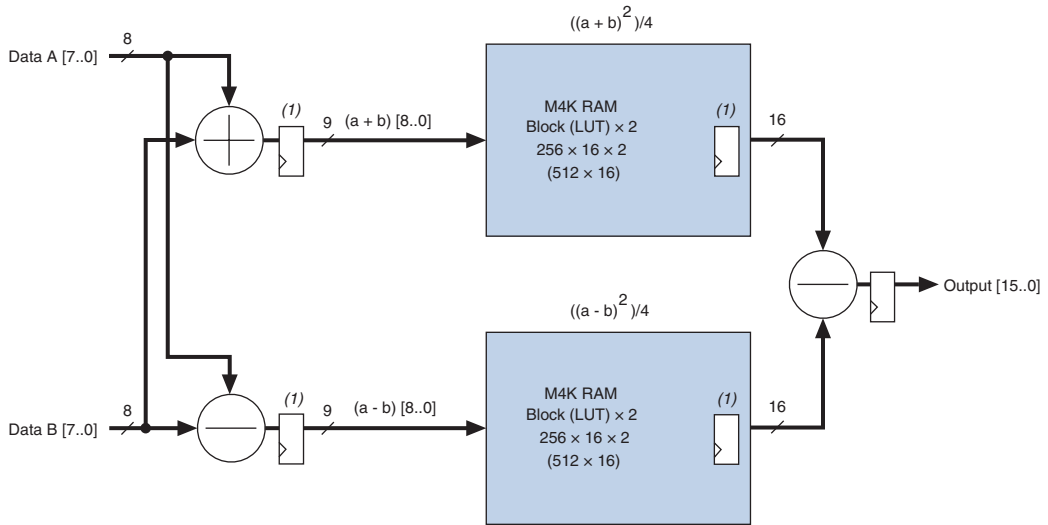
Note to Figure 21:
(1)   This is an optional pipeline register to increase system performance.

Figure 21 shows an implementation for two 8-bit data inputs. 8-bit inputs result in 16-bit outputs and 9-bit addresses per partial product RAM block. Therefore, for each partial product, two M4K RAM blocks are required in a 256×16 configuration ($2^9 = 512$ addresses). In this multiplier mode, the size of the inputs directly affects the total number of RAM blocks required.

Figure 22 shows the simulation results for the example shown in Figure 21.

Figure 22. Fully Variable Multiplier Simulation Results

Tables 32 and 33 show the implementation results of the 8-bit fully variable multiplier example shown in Figure 21 for Stratix II and Stratix devices, respectively. The fully variable multiplication mode is ideal for low-resolution multiplication in which the input and coefficient bit widths are not too large. Larger input and coefficient bit widths require a significant amount of memory block resources compared with other variable soft multiplier modes of the same size.

Table 32. 8-Bit Fully Variable Multiplier Implementation Results Using Stratix II Devices	
Device	EP2S15F484C3
Utilization	ALUTs: 36/12,480 (<1%) M4K RAM blocks: 4/78 (5%)
Latency *(1)*	Four clock cycles
Throughput	351 megasamples per second
Performance	351.0 MHz

Note to Table 32:

(1)   Latency is the number of clock cycles required to complete a single multiplication computation.

Table 33. 8-Bit Fully Variable Multiplier Implementation Results Using Stratix Devices

Device	EP1S10F484C5
Utilization	Logic cells: 34/10,570 (1%) M4K RAM blocks: 4/60 (6%)
Latency *(1)*	Four clock cycles
Throughput	291 megasamples per second
Performance	291.0 MHz

Note to Table 33:
(1)   Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**fully_var.zip**) for the design described in Tables 32 and 33 from the Design Examples section of the Altera web site at (**www.altera.com**).

Implementing Multipliers Using DSP Blocks or Logic Resources

Altera provides three Quartus II megafunctions for implementing various multiply, multiply-accumulate, and multiply-add functions using DSP blocks or logic resources:

- The `lpm_mult` megafunction performs multiply functions only.
- The `altmult_add` megafunction performs multiply or multiply-add functions.
- The `altmult_accum` megafunction performs multiply-accumulate functions only.

See Quartus II Online Help for instructions on using the megafunctions and the MegaWizard Plug-In Manager.

Firm Multipliers

Stratix II, Stratix, and Stratix GX firm multipliers use a combination of DSP blocks and logic resources. Cyclone II firm multipliers use a combination of embedded multipliers and logic resources. Firm multipliers allow you to increase the utilization efficiency of the DSP blocks or embedded multipliers within your Stratix II, Stratix, Stratix GX, or Cyclone II device. Stratix II, Stratix, and Stratix GX DSP blocks support 9×9, 18×18, and 36×36 multipliers. Cyclone II device embedded multipliers support 9×9 and 18×18 multipliers. If you implement a multiplier of a different size, some DSP blocks or embedded multipliers may be partially used. For example, a 12×9 multiplier uses two 9×9 DSP blocks or embedded multipliers because the 12-bit input exceeds the maximum requirement of a single 9×9 DSP block or embedded multiplier. The first 9×9 DSP block or embedded multiplier is fully utilized but the second 9×9 DSP block or embedded multiplier is

partially used. Instead of partially utilizing a DSP block or embedded multiplier for the remaining logic, you can use logic resources to implement it, freeing the DSP block or embedded multiplier for other use. This method is particularly useful if your design requires a lot of DSP blocks or embedded multipliers but has logic resources available.

To implement a firm 12×9 multiplier, split up the 12-bit input and decompose the multiplication into smaller, partial products that can be implemented in DSP blocks or embedded multipliers and logic resources. To maximize DSP block or embedded multiplier usage, split the 12-bit input into two sections: a 9-bit section that is multiplied using the DSP block or embedded multiplier and a 3-bit section that is multiplied using logic resources. If the 9-bit section consists of LSBs, it becomes an unsigned value while the 3-bit section becomes a signed value and vice versa.

When deciding whether to select the 3-bit section from the MSB or the LSB of the 12-bit input, keep in mind that an adaptive logic module (ALM) or LE multiplier is more resource efficient when implemented as a signed multiplier than as an unsigned multiplier. If the 9-bit input is unsigned, the 3-bit section is chosen from the MSB so that the ALM or LE multiplier performs signed multiplication. If the 9-bit input is signed, you can choose the 3-bit section from the MSB or LSB because either implementation results in a signed multiplier implemented in ALMSs or LEs.

Figure 23 shows the decomposition of the 12×9 firm multiplier.

*Figure 23. Decomposition of the 12 × 9 Multiplier*
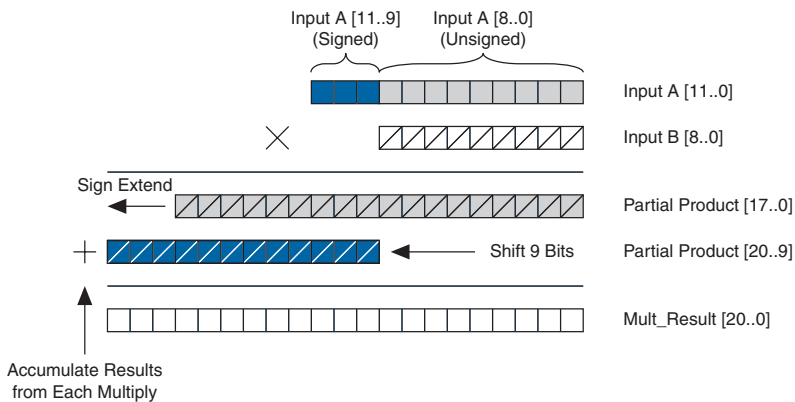
Based on this decomposition, you can build the circuit for the firm multiplier using three main blocks:

- DSP block or embedded multiplier—Built using either the `lpm_mult` or `altmult_add` megafunctions
- ALM- or LE-based multiplier—Built using either the `lpm_mult` or `altmult_add` megafunctions
- End-stage adder—Built using the `lpm_add_sub` megafunction

The DSP block or embedded multiplier multiplies the 9-bit input by the 9-bit LSB section of the 12-bit input. The ALM- or LE-based multiplier multiplies the 9-bit input with the 3-bit MSB section of the 12-bit input. The result of both multipliers is the partial products of the decomposition. The results of the partial products are weighted prior to being summed in the end-stage adder. This weighting and addition restores the bit-alignment of the partial products to ensure proper result values. Based on Figure 23, the 9×3 multiplication partial product is weighted by a shift to the left of nine bits. The 12-bit end-stage adder has to accommodate the 12-bit result of the 9×3 multiplication and the nine MSBs of the 9×9 multiplication, sign extended.

Figure 24 shows the circuit of the 12×9 firm multiplier.

Figure 24. 12 × 9 Firm Multiplier Circuit   *Note (1)*
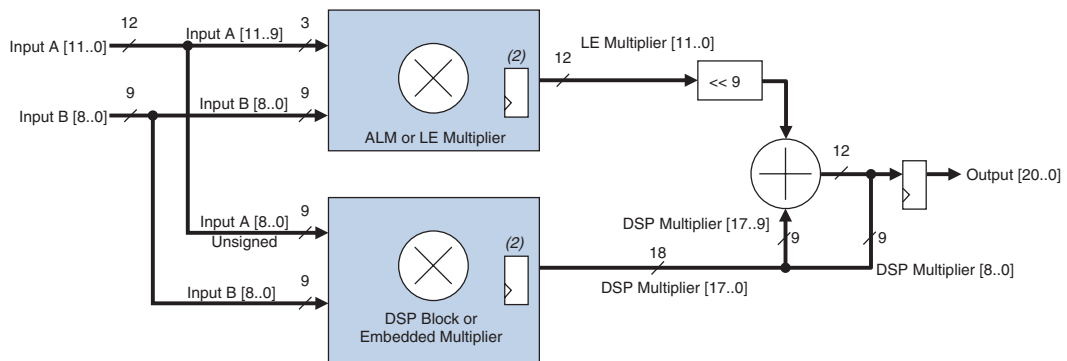
Notes to Figure 24:
(1) Using the `altmult_add` megafunction to implement the multipliers allows you to mix signed and unsigned inputs.
(2) This is an optional pipeline register to increase system performance.

Figure 25 shows the simulation results for the example shown in Figure 24.

Figure 25. 12 × 9 Firm Multiplier Simulation Results
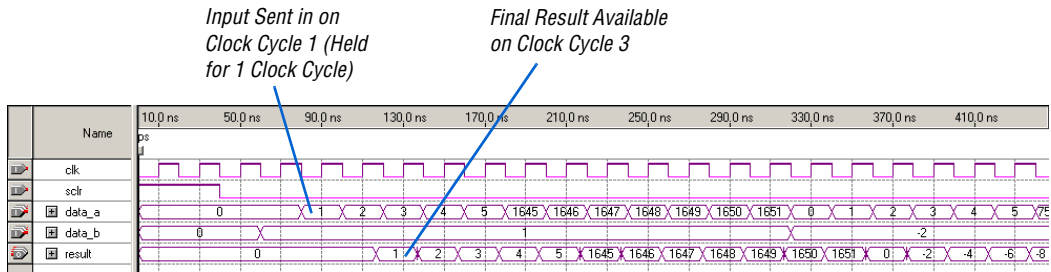
Tables 34 and 35 show the implementation results for the 12 × 9 firm multiplier circuit example shown in Figure 24 for Stratix II and Stratix devices, respectively.

Table 34. 12 × 9 Firm Multiplier Implementation Results Using Stratix II Devices   *Note (1)*

Device	EP2S15F484C3
Utilization	Logic cells: 60/12,480 (<1%) DSP block 9-bit elements: 1/96 (1%)
Latency *(2)*	2 clock cycles
Throughput	317 megasamples per second
Performance	317.0 MHz

Notes to Table 34:
(1) The `altmult_add` megafunction implements both the ALM or LE and DSP block multipliers.
(2) Latency is the number of clock cycles required to complete a single multiplication computation.

Table 35. 12 × 9 Firm Multiplier Implementation Results Using Stratix Devices   *Note (1)*

Device	EP1S10F484C5
Utilization	Logic cells: 68/10,570 (1%) DSP block 9-bit elements: 1/48 (2%)
Latency *(2)*	2 clock cycles
Throughput	274 megasamples per second
Performance	274.0 MHz

Notes to Table 35:
(1) The `altmult_add` megafunction implements both the ALM or LE and DSP block multipliers.
(2) Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**12x9_firm_mult.zip**) for the design described in Tables 34 and 35 from the Design Examples section of the Altera web site (**www.altera.com**).

The example shown in Figure 24 is suitable when only one of the multiplier inputs exceeds the 9-bit input width of a single DSP block or embedded multiplier. When both multiplier inputs exceed 9-bits, as in the case of a 12 × 12 multiplier, the multiplication must be decomposed into three partial products instead of two. The 12-bit inputs must be sectioned to maximize the use of the 9 × 9 DSP blocks or embedded multipliers and the utilization efficiency of implementing signed multiplication in logic resources. Therefore, both inputs should be sectioned into a 3-bit MSB section and a 9-bit LSB section.

Figure 26 shows the decomposition of the 12 × 12 multiplier.

*Figure 26. Decomposition of the 12 × 12 Multiplier*
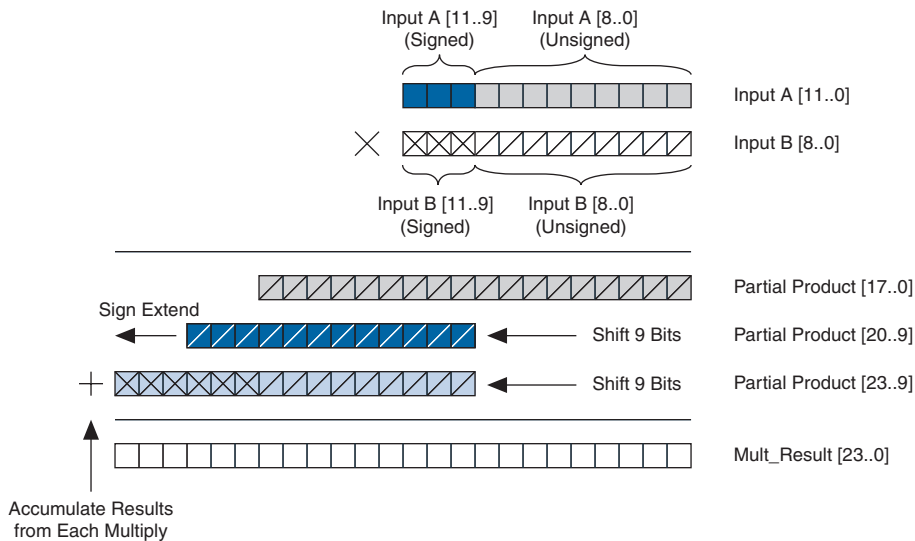
The circuit for the firm multiplier can now be extracted from the decomposition. The firm multiplier circuit consists of five main blocks:

- One DSP block multiplier or embedded multiplier—Built using either the `lpm_mult` or `altmult_add` megafunctions
- Two ALM- or LE-based multipliers—Built using either the `lpm_mult` or `altmult_add` megafunctions
- Two adders—Built using the `lpm_add_sub` megafunction

The DSP block or embedded multiplier multiplies the two 9-bit LSB sections of the 12-bit inputs. The first ALM- or LE-based multiplier multiplies the 9-bit LSB section of one 12-bit input with the 3-bit MSB section of the other 12-bit input. The other ALM- or LE-based multiplier multiplies the 3-bit MSB of one 12-bit input with the entire 12-bits of the other input. The results of these three multipliers are the three partial products of the decomposition. The results of these partial products are summed in two stages (using two adders) prior to producing the final output.

Figure 27 shows the two adder stages within the final circuit of the 12 × 12 firm multiplier.

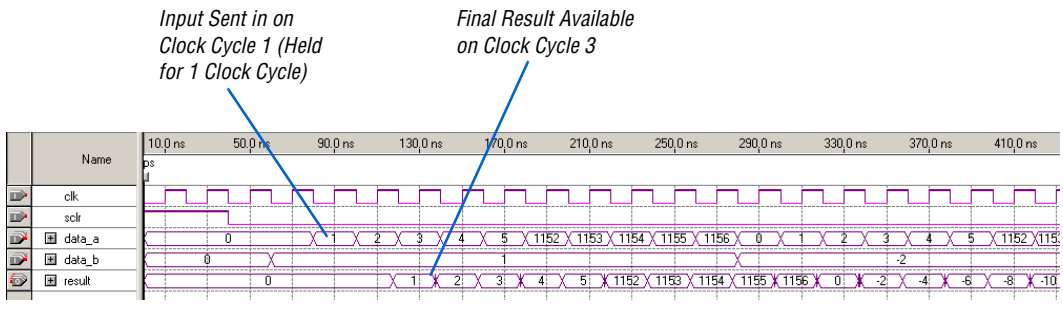*Figure 27. 12 × 12 Firm Multiplier Circuit   Note (1)*
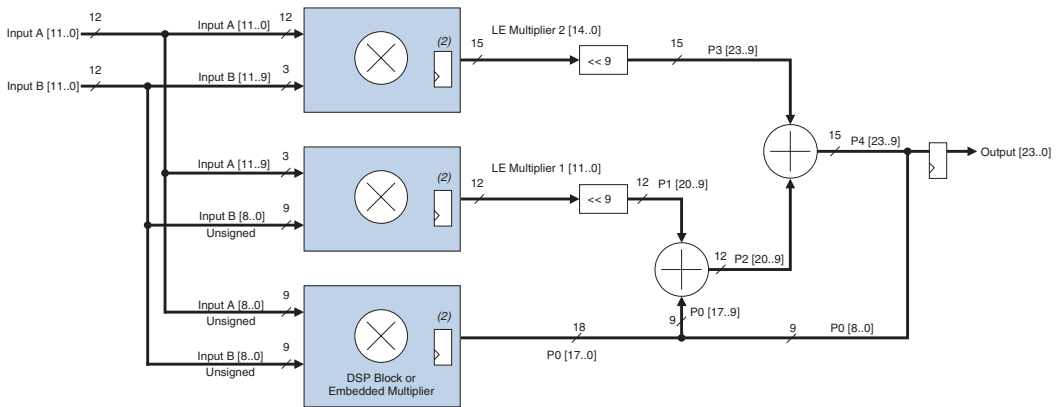
Notes to Figure 27:
(1)   Using the `altmult_add` megafunction to implement the multipliers allows you to mix signed and unsigned inputs.
(2)   This is an optional pipeline register to increase system performance.

Figure 28 shows the simulation results for the example shown in Figure 27.

Figure 28. 12 × 12 Firm Multiplier Simulation Results

Tables 36 and 37 show the implementation results for the 12 × 12 firm multiplier example shown in Figure 27 for Stratix II and Stratix devices, respectively.

Table 36. 12 × 12 Firm Multiplier Implementation Results Using Stratix II Devices Note (1)

Device	EP2S15F484C3
Utilization	ALUTs: 126/12,480 (1%) DSP block 9-bit elements: 1/96 (1%)
Latency *(2)*	2 clock cycles
Throughput	238 megasamples per second
Performance	238.0 MHz

Notes to Table 36:
(1) The `altmult_add` megafunction implements both the ALM or LE and DSP block multipliers.
(2) Latency is the number of clock cycles required to complete a single multiplication computation.

Table 37. 12 × 12 Firm Multiplier Implementation Results Using Stratix Devices Note (1)

Device	EP1S10F484C5
Utilization	Logic cells: 144/10,570 (1%) DSP block 9-bit elements: 1/48 (2%)
Latency *(2)*	2 clock cycles
Throughput	180 megasamples per second
Performance	180.0 MHz

Notes to Table 37:
(1) The `altmult_add` megafunction implements both the ALM or LE and DSP block multipliers.
(2) Latency is the number of clock cycles required to complete a single multiplication computation.

You can download the files (**12x12_firm_mult.zip**) for the design described in Tables 36 and 37 from the Design Examples section of the Altera web site (**www.altera.com**).

Conclusion

Stratix II, Stratix, and Stratix GX DSP blocks and Cyclone II embedded multipliers are designed for implementing DSP applications. However, you can also use Stratix II, Stratix, and Stratix GX TriMatrix blocks (M512 or M4K RAM blocks) or Cyclone II and Cyclone M4K RAM blocks for designs that need more multipliers than are available using DSP blocks or embedded multipliers alone. For example, using soft multipliers, you can increase the number of 16 × 16 multipliers in a Stratix E1S80 device by a factor of more than 7 (see Table 14 on page 8). Another example is that the fully variable soft multiplier is an ideal implementation for applications

requiring smaller multipliers with frequently varying coefficients. Other soft multiplier modes are more resource efficient and better suited for applications that do not require frequent coefficient updates. The firm multiplier allows you to balance the use of DSP block or embedded multipliers with ALM- or LE-based multipliers, allowing more efficient use of the Stratix II, Stratix, and Stratix GX DSP blocks and Cyclone II embedded multipliers.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com

I.S. EN ISO 9001