

## Introduction

Altera® Stratix™ devices have dedicated digital signal processing (DSP) blocks optimized for DSP applications. DSP blocks are ideal for implementing DSP applications requiring high data throughput, such as finite impulse response (FIR) filters, complex FIR filters, infinite impulse response (IIR) filters, fast Fourier transform (FFT) functions, direct cosine transform (DCT) functions, and correlators.

This document provides design guidelines for using the Stratix DSP blocks in the Synplify® version 7.1 software for Altera.

## DSP Blocks in Stratix Devices

The DSP blocks in Stratix devices consists of multipliers, adders, subtractors, accumulators, and summation units. Optional pipeline, input, and output registers are also available.

You can configure a DSP block in one of four operation modes as shown in [Table 1](#).

<b>Table 1. Operation Modes for DSP Block</b>			
<b>DSP Block Mode</b>	<b>Number &amp; Size of Multipliers per DSP Block <i>Note (1)</i></b>		
	<b>9 × 9-bit</b>	<b>18 × 18-bit</b>	<b>36 × 36-bit</b>
Simple multiplier	8 multipliers with 8 product outputs	4 multipliers with 4 product outputs	1 multiplier with 1 product output
Multiply-accumulator	2 multiply and accumulate (34 bit)	2 multiply and accumulate (52 bit)	N/A
Two-multipliers adder	4 sums of 2 multiplier products each	2 sums of 2 multiplier products each	N/A
Four-multipliers adder	2 sums of 4 multiplier products each	1 sum of 4 multiplier products each	N/A

**Note to Table 1:**

- (1) DSP blocks can be combined to create larger sized functions, if necessary.



For details on DSP block features and operation modes, refer to *AN 214: Using DSP Blocks in Stratix Devices*.

## DSP Block Megafunctions in the Quartus II Software

The following Altera megafunctions are used with DSP block modes:

- `lpm_mult`
- `altmult_accum`
- `altmult_add`

You can instantiate these megafunctions in the design or have a third-party synthesis tool, such as the Synplify software, infer the appropriate megafunction by recognizing a multiplier, multiply accumulator (MAC), or multiply-adder in the design. The Altera Quartus® II version 2.0 software maps the functions to the DSP blocks in the device during place-and-route.



For details on these megafunctions, refer to *AN 214: Using DSP Blocks in Stratix Devices*.

## Inferring DSP Blocks

Based on the VHDL or Verilog coding style, the Synplify software can infer the correct megafunctions.

### Simple Multipliers

The `lpm_mult` megafunction implements the DSP block in the simple multiplier mode. In this mode:

- The DSP block includes registers for the input and output stages and an intermediate pipeline stage.
- Signed and unsigned arithmetic are supported.

[Table 2](#) shows the `LPM_PIPELINE` values corresponding to the latency and registers used.



The latency or number of pipeline stages in the design is passed to the Quartus II software using the `LPM_PIPELINE` parameter. The Synplify software automatically sets this parameter based on the design.

**Table 2. `lpm_mult` Megafunction Latency Values**

LPM_PIPELINE Value	DSP Block Registers Used		Latency (1)
	Input	Output	
0	-	-	0
1	✓	-	1
2	✓	✓	2

**Note to Table 2:**

- (1) The Synplify software maps input and output register stages to DSP blocks. Additional pipeline registers are mapped to logic elements (LEs).



The `lpm_mult` megafunction does not support using the input registers as shift registers. Hence, the Synplify software does not infer `lpm_mult` megafunctions with input registers being used as shift registers.

Figures 1 and 2 show sample Verilog and VHDL code, respectively, for inferring the `lpm_mult` megafunction in the Synplify software.



Associated with this application note are sample code text files (`synplify_verilog.txt` and `synplify_vhdl.txt`) in the Literature section of the Altera web site available at <http://www.altera.com>. The sample code in these text files use the input, output, and pipeline registers along with the simple multiplier.

**Figure 1. Verilog HDL Code for Inferring lpm\_mult (Unsigned 8 × 8 Multiplier with Pipeline=2)**

```
module lpm_mult_8_inreg_pipereg_unsigned (out, clk, a, b);

//Port Declarations
output [15:0] out;
input      clk;
input  [7:0] a;
input  [7:0] b;

//Register Declarations
reg      [7:0] a_reg;
reg      [7:0] b_reg;
reg      [15:0] out;

//Wire Declarations
wire     [15:0] mult_out;

assign mult_out = a_reg * b_reg;

always@(posedge clk)

begin
    a_reg <= a;
    b_reg <= b;
    out <= mult_out;
end

endmodule
```

**Figure 2. VHDL Code for Inferring *lpm\_mult* (Unsigned 8×8 Multiplier with Pipeline=2)**

```
library ieee ;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_signed.all;

entity unsigned_mult2 is
  port (
    a: in std_logic_vector (7 downto 0) ;
    b: in std_logic_vector (7 downto 0) ;
    clk : in std_logic;
    aclr : in std_logic;
    result: out std_logic_vector (15 downto 0)
  ) ;
end unsigned_mult2;

architecture rtl of unsigned_mult2 is
  signal a_int, b_int : std_logic_vector (7 downto 0);
  signal pdt_int : unsigned (15 downto 0);

begin

  process (clk, aclr)
  begin
    if ( aclr = '1' ) then
      a_int <= (others => '0');
      b_int <= (others => '0');

      elsif (clk'event and clk = '1') then
        a_int <= a;
        b_int <= b;
        pdt_int <= unsigned(a_int) * unsigned(b_int);
      end if;
    end process;

    result <= std_logic_vector(pdt_int);
  end rtl ;
```

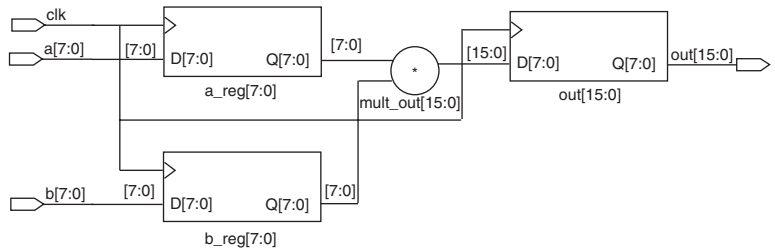
---

Figure 3 provides the RTL view of the code in Figures 1 and 2 after synthesis as seen in HDL Analyst® in the Synplify software

---

**Figure 3. HDL Analyst View of *lpm\_mult* Megafunction (Unsigned 8×8)**

**Multiplier with Pipeline=2)**



**Multiply Accumulators**

The `altmult_accum` megafunction implements the DSP block in the multiply-accumulator mode. In this mode:

- The DSP block includes registers for the input and output stages and an intermediate pipeline stage.
- The output registers are required for the accumulator.
- The input and pipeline registers are optional.
- Signed and unsigned arithmetic are supported.

Table 3 shows the registers used and the corresponding latency values for the `altmult_accum` megafunction.



Unlike the `lpm_mult` megafunction, the `altmult_accum` megafunction does not use the `LPM_PIPELINE` parameter to pass the latency value to the Quartus II software. Instead, the Synplify software automatically infers the required set of registers based on the design.

DSP Block Registers Used			Latency (1), (2)
Input	Pipeline	Output	
-	-	✓	1
✓	-	✓	2
✓	✓	✓	3

**Notes to Table 3:**

- (1) A latency larger than 3 requires the use of logic element (LE) registers external to the DSP block.
- (2) The `altmult_accum` megafunction always uses the output registers. Hence, the `altmult_accum` megafunction always has a latency of at least one clock cycle.



If the design requires input registers to be used as shift registers, use the black-boxing method to instantiate the `altmult_accum` megafunction.

Figures 4 and 5 show sample Verilog and VHDL code, respectively, for inferring the `altmult_accum` megafunction in the Synplify software. These examples use only the required output registers, not the input or pipeline registers.



Associated with this application note are sample code text files (`synplify_verilog.txt` and `synplify_vhdl.txt`) in the Literature section of the Altera web site available at <http://www.altera.com>. The sample code in these text files use the input and pipeline registers with the multiply-accumulator.

---

**Figure 4. Verilog HDL Code for Inferring `altmult_accum` (Unsigned  $8 \times 8$  Multiplier & 32-bit Accumulator)**

```
module altmult_acc_8_outreg_unsigned (dataout, dataa, datab,
    clk, aclr, clken);

//Port Declarations
input    [7:0]    dataa;
input    [7:0]    datab;
input    clk;
input    aclr;
input    clken;

output   [31:0]   dataout;

//Register Declarations
reg      [31:0]   dataout;

//Wire Declarations
wire     [15:0]   multa;
wire     [31:0]   adder_out;

assign multa = dataa * datab;
assign adder_out = multa + dataout;

always @(posedge clk or posedge aclr)
begin
    if(aclr)
    begin
        dataout <= 0;
    end

    else if(clken)
    begin
        dataout <= adder_out;
    end
end
endmodule
```

**Figure 5. VHDL Code for Inferring `altmult_accum` (Unsigned  $8 \times 8$  Multiplier & 16-bit Accumulator)**

```
library ieee;
USE ieee.std_logic_1164.all;

USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

entity usigaltnmult_accum1 is
generic (size : integer := 4);
port (
    a: in std_logic_vector (7 downto 0);
    b: in std_logic_vector (7 downto 0);
    clk : in std_logic;
    accum_out: inout std_logic_vector (15 downto 0)
);

end usigaltnmult_accum1;

architecture synplify of usigaltnmult_accum1 is
    signal a_int, b_int : unsigned (7 downto 0);
    signal pdt_int : unsigned (15 downto 0);
    signal adder_out : unsigned (15 downto 0);
begin
    a_int <= unsigned (a);
    b_int <= unsigned (b);
    pdt_int <= a_int * b_int;
    adder_out <= pdt_int + unsigned(accum_out);
    process (clk)
    begin
        if (clk'event and clk = '1') then
            accum_out <= std_logic_vector (adder_out);
        end if;
    end process;
end synplify;
```

---

## Multiplier Adders

The Synplify software can infer multiplier adders and map them to either the two-multiplier adder mode or the four-multiplier adder mode of the DSP blocks. The Synplify software maps the HDL code to the correct `altmult_add` function. In these modes:

- The DSP block includes registers for the input and output stages and an intermediate pipeline stage.
- Signed and unsigned arithmetic are supported.

Table 4 shows the registers used and the corresponding latency values for the `altmult_add` megafunction.



Unlike the `lpm_mult` megafunction, the `altmult_add` megafunction does not use the `LPM_PIPELINE` parameter to pass the latency value to the Quartus II software. Instead, the Synplify software infers the required set of registers based on the design.

DSP Block Registers Used			Latency (1)
Input	Pipeline	Output	
-	-	-	0
✓	-	-	1
✓	-	✓	2
✓	✓	✓	3

**Note to Table 4:**

- (1) A latency larger than 3 requires the use of logic element (LE) registers external to the DSP block.

The `altmult_add` megafunction allows you to choose any of the register sets based on the HDL coding style. Figures 6 and 7 show sample Verilog and VHDL code, respectively, for inferring `altmult_add`. These samples do not use any of the register sets.



Associated with this application note are sample code text files (`synplify_verilog.txt` and `synplify_vhdl.txt`) in the Literature section of the Altera web site available at <http://www.altera.com>. The sample code in these text files use the input and pipeline registers with the `altmult_add` megafunction.

**Figure 6. Verilog HDL Code for Inferring `altmult_add` (Unsigned  $16 \times 16$  Multiply & 32-bit Result)**

```
module altmult_add_16_unsigned ( dataa, datab, datac, datad,
                               result);

    // Port Declaration
    input  [15:0] dataa;
    input  [15:0] datab;
    input  [15:0] datac;
    input  [15:0] datad;

    output [32:0] result;

    // Wire Declaration
    wire  [31:0] mult0_result;
    wire  [31:0] mult1_result;

    // Implementation
    // Each of these can go into one of the 4 mults in a DSP block
    assign mult0_result = dataa * datab;
    assign mult1_result = datac * datad;

    // This adder can go into the adder in a DSP block
    assign result = (mult0_result + mult1_result);

endmodule
```

**Figure 7. VHDL Code for Inferring `altmult_add` (Unsigned  $8 \times 8$  Multiply & 16-bit Result)**

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

entity unsignedmult_add is
  port (
    a: in std_logic_vector (7 downto 0);
    b: in std_logic_vector (7 downto 0);
    c: in std_logic_vector (7 downto 0);
    d: in std_logic_vector (7 downto 0);
    result: out std_logic_vector (15 downto 0)
  );
end unsignedmult_add;

architecture rtl of unsignedmult_add is
  signal a_int, b_int, c_int, d_int : unsigned (7 downto 0);
  signal pdt_int, pdt2_int : unsigned (15 downto 0);
  signal result_int: unsigned (15 downto 0);
begin
  a_int <= unsigned (a);
  b_int <= unsigned (b);
  c_int <= unsigned (c);
  d_int <= unsigned (d);
  pdt_int <= a_int * b_int;
  pdt2_int <= c_int * d_int;
  result_int <= pdt_int + pdt2_int;
  result <= std_logic_vector(result_int);
end rtl ;
```

## Resource Balancing

While mapping multipliers to DSP blocks, the Synplify software performs resource balancing for optimum performance.

Stratix devices have a fixed number of DSP blocks, which implies a fixed number of embedded multipliers. If the design uses more multipliers than are available, then the Synplify software automatically maps the extra multipliers to LEs.

If a design has more multipliers than are available in the DSP blocks, the Synplify software maps the multipliers in the critical paths to DSP blocks. Next, any wide multipliers, which may or may not be in the critical paths, are mapped to DSP blocks. Smaller multipliers and/or multipliers that are not in the critical paths may then be implemented in LEs. This ensures that the design fits successfully in the device.

## Controlling DSP Block Inferencing

Multipliers, accumulators, and adders can be implemented in DSP blocks or in logic elements in Stratix devices. The user can control this implementation through attribute settings in the Synplify software.

As shown in [Table 5](#) a signal level attribute in the Synplify software controls the implementation of the multipliers in the DSP blocks or LEs.

**Table 5. Attribute Settings for DSP Block in the Synplify Software**

Level	Attribute Name	Value	Description
Signal	syn_multstyle	lpm_mult	LPM function inferred and multipliers implemented in DSP block
		logic	LPM function not inferred and multipliers implemented in LEs by the Synplify software

### Signal Level Attribute

You can control the implementation of individual multipliers by using the `syn_multstyle` attribute as shown below:

```
<signal_name> /* synthesis syn_multstyle = "logic" */
```

where `signal_name` is the name of the signal.



This setting applies to wires only; it cannot be applied to registers.

[Figures 8](#) and [9](#) show sample Verilog and VHDL code, respectively, using the `syn_multstyle` attribute.

**Figure 8. Signal Attributes for Controlling DSP Block Inference in Verilog HDL**

```

module mult(a,b,c,r,en);

input [7:0] a,b;
output [15:0] r;
input [15:0] c;
input en;
wire [15:0] temp /* synthesis syn_multstyle="logic" */;

assign temp = a*b;
assign r = en ? temp : c;
endmodule

```

**Figure 9. Signal Attributes for Controlling DSP Block Inference in VHDL Code**

```

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

entity onereg is port (
    r : out std_logic_vector(15 downto 0);
    en : in std_logic;
    a : in std_logic_vector(7 downto 0);
    b : in std_logic_vector(7 downto 0);
    c : in std_logic_vector(15 downto 0)
);
end onereg;

architecture beh of onereg is

signal temp : std_logic_vector(15 downto 0);
attribute syn_multstyle : string;
attribute syn_multstyle of temp : signal is "logic";

begin
temp <= a * b;
    r <= temp when en='1' else c;
end beh;

```

## Black Boxing DSP Blocks



In addition to inferring the DSP block through HDL code, the Synplify software also supports black boxing of the DSP blocks. Using black-boxing techniques, you can instantiate a Quartus II software-generated DSP megafunction, such as `lpm_mult`, `altmult_add`, or `altmult_accum`. You can customize these megafunctions in the Quartus II software and take advantage of all the DSP block features through the MegaWizard Plug-In Manager

Refer to AN 231: *Designing with Stratix Devices in Synplify* for more details on black boxing DSP blocks in the Synplify software.

## Guidelines for Using DSP Blocks

In addition to the guidelines mentioned earlier in this application note, use the following guidelines while designing with DSP blocks in the Synplify software:

- Follow the black-boxing technique when using the input registers as shift registers for the `altmult_accum` megafunction.
- The `altmult_add` megafunction is the most versatile megafunction because it allows maximum individual control of the input, pipeline, and output registers. Hence, use the `altmult_add` megafunction instead of `lpm_mult` megafunction if you need more control of the registers available in the DSP blocks.
- The `altmult_add` megafunction can implement the same functionality as the `lpm_mult` megafunction.
- To access all the different control signals for the DSP block, such as `sign_A`, `sign_B`, and `dynamic_addnsub`, use the black-boxing technique.
- While performing signed operations, ensure that the specified data width of the output port matches the data width of the expected result. Otherwise the sign bit may be lost or data will be incorrect because the sign is not extended. For example, if the data widths of input A and B are `width_a` and `width_b`, respectively, then the maximum data width of the result can be  $(width\_a + width\_b + 2)$  for the four-multipliers adder mode. Thus, the data width of the output port should be less than or equal to  $(width\_a + width\_b + 2)$ .
- While using the accumulator, the data width of the output port should be equal to or greater than  $(width\_a + width\_b)$ . The maximum width of the accumulator can be  $(width\_a + width\_b + 16)$ . Accumulators wider than this are implemented in LEs.
- If the input and output registers use different control signals, then follow the black-boxing technique for instantiating the DSP block in your design to implement these registers in the DSP block. Otherwise, these registers may be implemented in LEs.
- If the design uses more multipliers than available in a particular Stratix device, you might get a no fit error in the Quartus II software. In such cases, use the attribute settings in the Synplify software to control the mapping of multipliers in your design to DSP blocks or LEs.

## Sample HDL Code

Two text files available on the Application Note section of <http://www.altera.com> have sample Verilog and VHDL code examples that infer the DSP block in the different operation modes.

Table 6 is a list of the modes with sample code available in the text files.

**Table 6. Sample Verilog & VHDL Code in Text Files**

Megafunction	Unsigned Operations	Signed Operations	Latency			
			0	1	2	3
lpm_mult (a*b)	✓	✓	✓	✓	✓	N/A
altmult_add (a*b + c*d)	✓	✓	✓	✓	✓	✓
altmult_accum (a*b+x)	✓	✓	- (1)	✓	✓	✓

Note to Table 6:

(1) Latency of 0 is not supported as the output registers are required in this mode.



The syntax is slightly different for signed and unsigned representations in the sample code.

## Conclusion

This application note has shown how to use the Synplify software with the DSP blocks available in Stratix devices. The DSP block operation modes are ideal for implementing DSP applications.



101 Innovation Drive  
 San Jose, CA 95134  
 (408) 544-7000  
<http://www.altera.com>  
 Applications Hotline:  
 (800) 800-EPLD  
 Literature Services:  
[lit\\_req@altera.com](mailto:lit_req@altera.com)

Copyright © 2002 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Synplify is a registered trademark of Synplicity. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services

