Stratix II DSP Block EDA Functional Description

Version 1.2
March 22, 2004

by
Altera Corporation

Author:	Altera Corporation
System:	
Subsystem:	
Keywords:	Stratix II, MAC, DSP block, Multiplier

Table of Contents:

1 Overview

This document is intended for EDA tool developers working with the Stratix II DSP blocks. It is a companion to the "Stratix II MAC WYSIWYG Description" and should be used in conjunction with it. This document provides information about the Stratix II DSP block to allow EDA vendors to infer these blocks optimally.

2 Stratix II DSP Block Description

The Stratix II DSP Block (also referred to as the MAC Block) is a block that implements basic DSP functions, such as multipliers, multiply-accumulators, multiply-adders, and input shift registers. This block is a complete superset of the Stratix DSP Block, and so can implement all features that can be implemented in a Stratix DSP Block.

As in the Stratix architecture, each DSP Block is composed of four 18-bit multipliers that can feed adders. Some of these adders can be configured to implement an accumulator. These adders cannot be used independent of the multipliers – the multipliers must always be used. The four 18-bit multipliers can be configured to instead implement eight 9-bit multipliers, or a single 36-bit multiplier without any external logic.

The Stratix II DSP Block adds several new features not available in the Stratix DSP Block:

1. Built-in rounding and saturation support for Q1.15 format numbers
2. Accumulator is loadable with separate load data bus
3. Input shift registers are dynamically loadable
4. Ability to dynamically switch between some modes
5. Ability to mix multiple modes in a single DSP block

The Stratix II MAC WYSIWYG document describes the various configurations and their options in more detail. It also describes the Q1.15 number format.

2.1 DSP Block Chip Layout

On a Stratix II device, there are two, three, or four columns of DSP blocks. The smaller members have two columns, and the largest member has four columns. Additionally, the DSP block is four LABs tall, compared to the Stratix architecture where it was eight LABs tall.

As a result, there are more DSP blocks in the Stratix II devices compared to the Stratix devices- from 12 to 96, depending on device, compared to a maximum of 22 in the Stratix architecture.

2.2 Input Shift Register (Scan Chain)

Input shift registers are commonly used in DSP filter applications, such as a FIR filter. It can be shown that a 4-tap FIR Filter can be implemented in a single DSP block using this feature.

As in the Stratix family, the scan chain of the Stratix II DSP block is a shift register implemented out of the input registers of the multipliers. The output of each multiplier's input register can be configured to feed the input of the next multiplier through dedicated routing. The last multiplier in a DSP block can feed the first multiplier in the DSP block immediately below it in the same column.

There exists the ability for the shift register to feed non-dedicated routing, however use of this ability is not recommended since it introduces several fitting restrictions. Instead, if using a scan-chain, care must be taken to ensure that it will not require use of the non-dedicated routing path.

The scan chains can be used in all modes except the 36-bit multiplier mode. For this case, external shift registers will have to be implemented.

New for the Stratix II family is the ability to dynamically select whether a particular multiplier operand is fed by its regular data input, or its dedicated scan-chain input. This allows the implementation of a dynamically loadable shift-register: the shift register can operate as normal using the scan-chain path, and can be loaded in parallel by using the data input path. The sourcea and sourceb signals control which source feeds the multiplier input, and the scanina and scaninb ports allow specifying the dedicated scan-chain path. However, as in the Stratix architecture, if not dynamically selecting the input source, then the scan-chain outputs should feed the data inputs (i.e. they do not have to feed the scanin inputs).

2.3 Mixing Modes

In the Stratix family, a single DSP block could implement only DSP functions operating of the same mode. However, for the Stratix II family, functions operating in different modes can be mixed in the same DSP block. This minimizes wasted unused space in any given DSP block.

The following table summarizes which modes can be implemented in each section of a DSP block.

Mode	Section
Multiplier Only (9-bit, 18-bit)	per quarter
Multiplier Only (36-bit)	full block
Multiply-Accumulator	per half
Two-Multipliers Adder	per half
Four-Multipliers Adder	full block
Dynamic	full block

Only the section indicated for that mode needs to be in the same mode. For modes that take a full block, the entire DSP block needs to be in the same mode. But for modes that are constrained only to half a block, the other half of the block can be in a different mode (either half a block in one mode, or two modes that take up quarter of a block). Similarly, for modes that take up only a quarter block, the other three quarters of the block can be in a different mode (either modes that take up a quarter and/or a mode that takes up half of a block).

For example, a single Stratix II DSP block can implement two 9-bit multipliers, one 18-bit multiplier, and one 18-bit multiply-accumulator. In the Stratix architecture, this would have required three separate DSP blocks.

One caveat to note here is that although 9-bit multipliers could take up one eighth of a DSP block, they're still implemented in sections as per quarter, per half, of full block depending on the mode. For example, a single four-multiplier-adder with 9-bit multipliers is still implemented in a full block section, which means it can only be mixed with another four-multiplier-adder with 9-bit multipliers.

The Quartus II fitter will automatically take advantage of this ability when feasible. However, this slightly changes the method used to estimate the number of DSP blocks used, as described later.

3 Instantiating DSP Blocks

DSP blocks can be instantiated by either instantiating the MAC WYSIWYGs directly, or by instantiating one of three megafunctions that implement DSP blocks. As with the Stratix family, we recommend instantiating the megafunctions instead of the WYSIWYGs directly due to the complexity of some of the DSP modes.

The same megafunctions used to implement Stratix DSP blocks are used to implement Stratix II DSP blocks, and these megafunctions are backwards compatible with the Stratix device. As a result, a megafunction instantiated for Stratix devices can also be compiled for Stratix II devices.

The megafunctions are as follows:

1. The **LPM_MULT** megafunction that implements a single multiplier.
    - Does not provide full control of implementation since there are no parameters to specify exactly which registers can be used, or to allow dynamic sign control signals, or different sign representations for each operand. For these cases, it is better to instead use an ALTMULT_ADD with NUMBER_OF_MULTIPLIERS=1.
2. The **ALTMULT_ACCUM** megafunction that implements a multiplier feeding an accumulator.
    - This megafunction can implement a multiplier and an accumulator of any width by adding any necessary external logic.
    - ***[NEW for the Stratix II Family]*** Provides the ability to load the upper bits of the accumulator using external data. In the Stratix architecture, only the multiplier result could be loaded into the accumulator. This feature will only be supported for cases that fit in a single DSP block and no stitching with external logic will be performed (e.g. multiplier widths larger than 18-bits cannot use this feature). Note that this supports only loading the upper bits of the accumulator – the lower bits will need to be loaded using the multiplier (by performing a multiply-by-1 operation). To load the full accumulator bits, external logic (not implemented by this megafunction) will be needed to MUX the lower bits with one of the multiplier operands.
    - ***[NEW for the Stratix II Family]*** Provides the ability to perform rounding and saturation for Q1.15 format numbers in the accumulator and the multiplier. This feature will only be supported for cases that fit in a single DSP block.
    - ***[NEW for the Stratix II Family]*** Provides the ability to dynamically select the multiplier input source to implement loadable shift-registers.
3. The **ALTMULT_ADD** megafunction that implements one or more multipliers feeding an adder.

- If NUMBER_OF_MULTIPLIERS=1, then this megafunction will implement a single multiplier and will bypass the adder. Unlike lpm_mult, this provides full control of all registers and dynamic controls.
- ***[NEW for the Stratix II Family]*** Provides the ability to perform rounding and saturation for Q1.15 format numbers in the multipliers, and rounding for the first level adders. This feature will only be supported for cases that fit in a single DSP block.
- ***[NEW for the Stratix II Family]*** Provides the ability to dynamically select the multiplier input sources to implement loadable shift-registers.

All three megafunctions support the ability to be implemented in regular logic elements through the use of the DEDICATED_MULTIPLIER_CIRCUITRY parameter.

There is currently no megafunction that instantiates the DSP Block in dynamic mode. WYSIWYGs must be used to instantiate a DSP block in dynamic mode.

4 Estimating DSP Block Usage

This section describes how to estimate the usage of DSP Blocks in a particular design. In most cases, this estimate will be the same as the actual usage and only differs in cases where signal conflicts may result in higher usage.

As in Stratix devices, a DSP block 9-bit element (DSP element) is used to describe the smallest component in a DSP block. A DSP element is essentially a 9-bit multiplier, where one of these elements is needed to build a 9-bit multiplier, and two of these are needed to build an 18-bit multiplier. Four 9x9 multipliers are not needed to build an 18x18 since the hardware itself supports 18x18 multipliers, and 9x9 multipliers are derived by splitting each of the 18x18 multipliers in half.

By counting the number of these elements used for every mode and width, it is possible to determine the number of DSP blocks required since 8 DSP elements can fit in a single DSP block. In Stratix devices, multipliers of different mode or width could not be combined in the same DSP block. However, in Stratix II devices, mixing modes is possible, and thus better utilization of the MAC blocks is possible.

The following table shows the DSP element usage for every possible mode and width combination.

Mode	Width		
	9x9	**18x18**	**36x36**
Multiplier Only	1	2	8
Multiply-Accumulator	n/a(1)	4	n/a(2)
Two-Multipliers Adder	2	4	n/a(2)
Four-Multipliers Adder	4	8	n/a(2)
Dynamic	n/a(3)	8	n/a(3)

(1) The multiply-accumulator mode only supports 18-bit multipliers. All multipliers smaller than 18-bits (including those smaller than 9-bits) will be use up an 18-bit multiplier.

(2) Multipliers larger than 18x18 and up to 36x36 can only be implemented as independent multipliers since an entire DSP block is used to implement 36x36 multipliers. For all other

modes, the accumulator or adder will be implemented in LEs while only the multiplier portion is implemented in a DSP block set to 36x36 mode.

(3) In Dynamic mode, only 18-bit based multipliers can be used. However, a 36-bit multiplier can be implemented using the four 18-bit multipliers.

4.1 Counting Algorithm

To determine the number of DSP blocks required, the following algorithm can be used:

1. First determine the number of DSP elements used for each mode-width combination.
2. Since 9-bit multipliers must be implemented in the entire quarter of a DSP block, any 9-bit based mode's element count needs to be incremented until it is divisible by the number of DSP elements used for the 18-bit based mode (i.e. 9x9 mult-only mode should be divisible by 2, 9x9 two-mult mode should be divisible by 4, 9x9 four-mult mode should be divisible by 8).
3. Divide the number by 8 to give the number of partial DSP blocks used in this mode.
4. Add the number of partial DSP blocks used for each mode-width combination
5. Round up this value to get the number of DSP blocks used.

4.2 Counting Example

The following is an example showing how to count the number of DSP blocks used.

Suppose a design has the following DSP functions:

a) Nine 7x5 simple multipliers (not feeding an adder or accumulator)
b) Ten 8x8 simple multipliers
c) Five 16x16 simple multipliers
d) Three sets of two 14x12 multipliers feeding an adder (i.e. 6 multipliers feeding 3 adders)
e) Three sets of four 8x8 multipliers feeding an adder (i.e. 12 multipliers feeding 3 adders)

Each of these will lead to the following DSP element counts:

a) 9 DSP elements for 9x9 Multiplier Only Mode
b) 10 DSP elements for 9x9 Multiplier Only Mode
c) 10 DSP elements for 18x18 Multiplier Only Mode (= 5 x 2)
d) 12 DSP elements for 18x18 Two-Multipliers Adder Mode (= 3 x 4)
e) 12 DSP elements for 9x9 Four-Multipliers Adder Mode (= 3 x 4)

So for each mode-width combination, the following lists the number of DSP elements needed:

1) 9x9 Multiplier Only Mode: 19 DSP elements (= 9 + 10)
2) 18x18 Multiplier Only Mode: 10 DSP elements
3) 18x18 Two-Multipliers Adder Mode: 12 DSP elements
4) 9x9 Four-Multipliers Adder Mode: 12 DSP elements

For the 9x9-based modes, we need to make sure the count is divisible by their 18x18 counterpart:

1) 9x9 Multiplier Only Mode: 20 DSP elements (= 19 + 1 since 19 is not divisible by 2)
2) 18x18 Multiplier Only Mode: 10 DSP elements
3) 18x18 Two-Multipliers Adder Mode: 12 DSP elements
4) 9x9 Four-Multipliers Adder Mode: 16 DSP elements (= 12 + 4 since 12 is not divisible by 8)

For each mode-width combination, the following lists the number of partial DSP blocks needed

1) 9x9 Multiplier Only Mode: 20 / 8 = 2.5 DSP blocks needed
2) 18x18 Multiplier Only Mode: 10 / 8 = 1.25 DSP blocks needed
3) 18x18 Two-Multipliers Adder Mode: 12 / 8 = 1.5 DSP blocks needed
4) 9x9 Four-Multipliers Adder Mode: 16 / 8 = 2 DSP blocks needed

Adding the number of DSP blocks needed for each mode-width combination:

2.5 + 1.25 + 1.5 + 2 = 5.25 DSP blocks needed for this design.

This means 6 DSP blocks will be used for this design (rounding up to get a whole number). The number of DSP elements that will be reported as used for this design is 53 (note that this number does not include the inflation done for the 9x9 modes).

This counting method differs from the Stratix architecture since it takes into account the ability for a single DSP block to implement more than one mode. However, as in Stratix architecture, the concept of a DSP element representing the actual used portion of a DSP block is still the same.

5 Use of DSP Blocks Versus LEs

Since the DSP blocks are essentially blocks of logic, they can be interchanged with LEs to implement the same function. The advantages for using a DSP block over LEs are the large size savings, and the high performance advantage. However, since there is a limited number of DSP blocks in any given device, some multipliers may need to be implemented using regular LEs.

5.1 DSP Block Resource Balancing

Making the decision between using DSP blocks and LEs is performed by DSP block resource balancing in Quartus II. If not enough DSP blocks are available in the selected device, resource balancing will convert some of them into LEs based on the estimated number of LEs needed to implement each multiplier.

The functionality of this feature will be the same as the Stratix devices– the only change is that it now supports the new features of the Stratix II architecture. As before, resource balancing will honor all user parameters, and also allows further user control through entity assignments.

Synthesis tools may want to perform their own resource balancing to get a better estimate of resources and timing.

6 Inferring DSP Blocks

As in the Stratix architecture, synthesis tools could infer the following elements:

1) Simple multipliers (i.e. just the '*' operator)
2) Multipliers feeding 2 to 4 adders
3) A multiplier feeding an accumulator
4) Shift registers on multiplier inputs (i.e. scan-chains)

5) Registers on the inputs and outputs of multipliers, and outputs of adders/accumulators

Since the Stratix II DSP block is backwards compatible with the Stratix DSP block, no change is necessary to infer these elements in the Stratix II family compared to the Stratix family.

For Stratix II DSP blocks, the following new elements could also be inferred:

1) Multipliers feeding loadable accumulators:
    a. When only the upper bits of the accumulator are being loaded
    b. When the entire accumulator is being loaded, requiring external logic to MUX between the multiplier inputs and the load data bits.
2) Dynamically loadable input shift registers
    a. This is considered lower priority since it is not expected to be part of many designs.

The other new Stratix II features are not expected to be inferable. This includes the rounding and saturation features for Q1.15 format numbers, and the dynamic mode. The new mixed mode feature is handled by the Quartus II fitter.

As with the Stratix family, it is recommended that the megafunctions be used to specify the inferred DSP functions instead of inferring the WYSIWYGs directly due to the complexity of some of the modes.

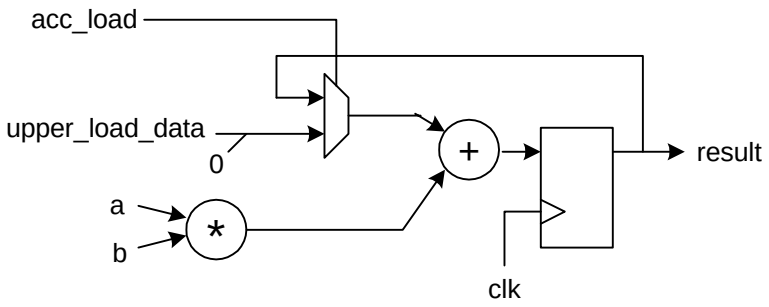
6.1 Examples

6.1.1 Multiply-Accumulator Loading Upper Bits

The following is a Verilog design example, its corresponding high-level netlist, and the resulting VQM file that can then be fed to Quartus II. This example is a multiply-accumulator with the upper bits of the accumulator being loaded.

```
module upper_loadable_macc (a, b, upper_load_data, acc_load, clk, result);

      input [15:0] a;  // mult A input
      input [15:0] b;  // mult B input
      input [31:0] upper_load_data;  // upper load data
      input acc_load;  // load control signal
      input clk;
      output [47:0] result;  // accumulator output
      reg [47:0] result;

      wire [31:0] mult_result = a * b; // multiplier

      always @(posedge clk)
            if (acc_load)
                  // load accumulator with upper_data and mult_result
                  result <= (upper_load_data << 16) + mult_result;
            else
                  // accumulate
                  result <= result + mult_result;
endmodule
```

The resulting high-level netlist will approximately be as follows:



The VQM file generated could be as follows:

```
module upper_loadable_macc (a, b, upper_load_data, acc_load, clk, result);

      input [15:0] a;
      input [15:0] b;
      input [31:0] upper_load_data;
      input acc_load;
      input clk;
      output [47:0] result;

      altmult_accum mult_accum_0 (
                  .dataa(a),
                  .datab(b),
                  .accum_sload(acc_load),
                  .accum_sload_upper_data(upper_load_data, 16'h0000),
                  .clk(1'b0, 1'b0, 1'b0, clk),
                  .result(result)
            );
      defparam mult_add_0.WIDTH_A = 16;
      defparam mult_add_0.WIDTH_B = 16;
      defparam mult_add_0.WIDTH_RESULT = 48;
      defparam mult_add_0.WIDTH_UPPER_DATA = 32;
      defparam mult_add_0.OUTPUT_REG = CLOCK0;
      defparam mult_add_0.INPUT_REG_A = "UNREGISTERED";
      defparam mult_add_0.INPUT_REG_B = "UNREGISTERED";
      defparam mult_add_0.MULTIPLIER_REG = "UNREGISTERED";
      defparam mult_add_0.ACCUM_SLOAD_REG = "UNREGISTERED";
      defparam mult_add_0.ACCUM_SLOAD_PIPELINE_REG = "UNREGISTERED";
      defparam mult_add_0.INTENDED_DEVICE_FAMILY = "Stratix II";

endmodule /* upper_loadable_macc */
```

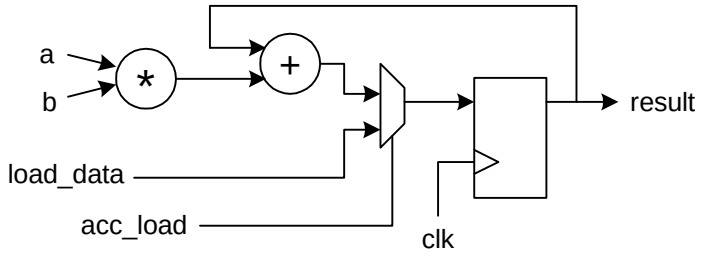
6.1.2 Multiply-Accumulator Loading All Bits

The following is a Verilog design example, its corresponding high-level netlist, and the resulting high-level netlist that can be converted to a VQM file for Quartus. This example is a multiply-accumulator with all the bits of the accumulator being loaded externally, thus requiring external logic in LEs in addition to the DSP block.

```
module loadable_macc ( a, b, load_data, acc_load, clk, result);

      input [15:0] a;  // mult A input
      input [15:0] b;  // mult B input
```

```
        input [47:0] load_data;  // load data
        input acc_load;  // load control signal
        input clk;
        output [47:0] result;  // accumulator output
        reg [47:0] result;

        wire [31:0] mult_result = a * b; // multiplier

        always @(posedge clk)
              if (acc_load)
                    // load full accumulator with load_data
                    result <= load_data;
              else
                    // accumulate
                    result <= result + mult_result;
endmodule
```

The resulting high-level netlist will approximately be as follows:



The high-level netlist generated with an altmult_accum and external logic is as follows: