Cyclone™ II DSP Block EDA Functional Description

Version 1.2
May 30, 2005

By
Altera Corporation

Table of Contents:

1 OVERVIEW 3
2 CYCLONE II DSP BLOCK DESCRIPTION 3
2.1 Features Supported 3
2.2 Block Organization 4
2.3 Operation Modes 5
3 CHIP LAYOUT 5
4 INSTANTIATING DSP BLOCKS 5
5 ESTIMATING DSP BLOCK USAGE 6
5.1 Counting Algorithm 6
5.2 Counting Example 7
6 USE OF DSP BLOCKS VERSUS LES 7
6.1 DSP Block Resource Balancing 7
7 INFERRING DSP BLOCKS 8

1 Overview

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

2 Cyclone II DSP Block Description

The Cyclone II DSP Block (also referred to as the MAC Block) is a block that implements the basic multipliers DSP function. This block is a strict subset of the Stratix® and Stratix II DSP Block.

2.1 Features Supported

The Cyclone II DSP block supports the following features:

- A base cell containing an 18bx18b multiplier features, with one full-precision (36 bits) 18bx18b multiplier output
- Capability to split the base cell containing one 18bx18b multiplier into a cell containing two 9bx9b multipliers, with two full-precision (18 bits each) 9bx9b multiplier outputs
- Input and output registers to improve block speed. Input register banks can be set/bypassed in 9bit chunks. Similarly, the output register banks can be set/bypassed in 18bit chunks.
- 2 dynamic input signals (SIGNX and SIGNY) have input registers that can be set/bypassed independently of the data register banks.

Comparing to the Stratix II DSP block, the following changes have been made for the Cyclone II DSP block:

- Block architecture changes:
  - The Cyclone II DSP block only contains a single 18bx18b multiplier instead of 4.
  - 1^{st} stage add/sub/accum and 2^{nd} stage adder units have been removed
  - Pipeline register unit has been removed
  - Output selection unit is removed
- 18bx18b multiplier features
  - Removed support for independent 52-bit multiply-accumulate operation of one 18bx18b multiplier.
  - Removed support for full-precision addition or subtraction result (37 bits each) of two 18bx18b multiplier.
  - Removed support for full precision addition result (38 bits) of four 18bx18b multiplier.
  - Removed support for 1.15 format rounding and saturation.
- 9bx9b multiplier features
  - Removed support for full-precision addition or subtraction results (19 bits each) of two 9bx9b multiplier outputs.
  - Removed support for full-precision addition result (20 bits) of four 9bx9b multiplier outputs.
  - Removed support for 36bx36b multiplier mode.

- Removed support for 2 built-in parallel 18-bit scan chain on input register for digital filter applications mode.

2.2 Block Organization

Figure 2-1 shows a simplified high-level organization of the DSP block for the 18bx18b-multiplier-based organization.

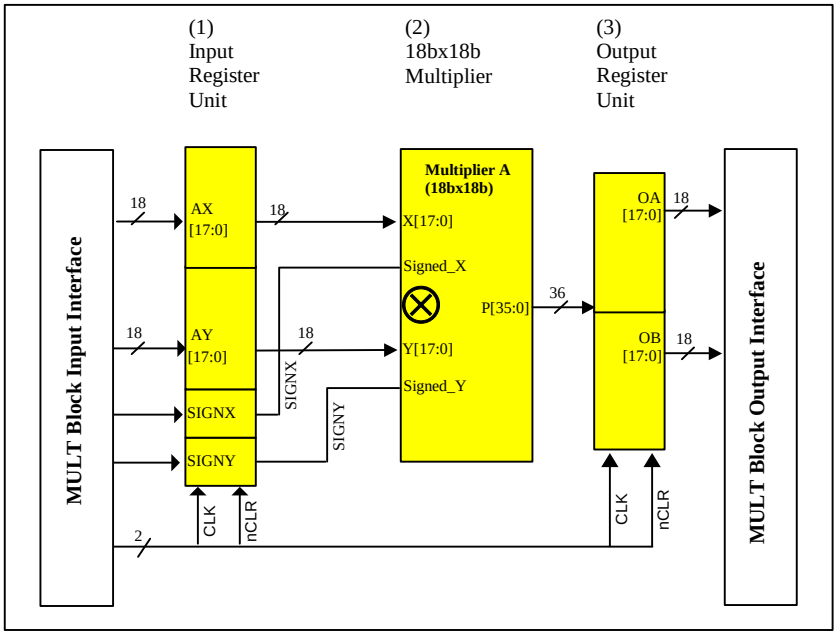

Figure 2-1: 18bx18b Multiplier Based Organization

The DSP block is consists of the following components:

- Input register unit
  The data registers are grouped in 9-bit banks that share the same clock and clear resources.
- Base 18bx18b multipliers
  The two 18b-wide input buses to a base 18bx18b multiplier are the "multiplicand-input-bus" (X[17:0]) and the "multiplier-input-bus" (Y[17:0]). These two input buses are logically interchangeable.
- Output register unit
  The registers are grouped in 2 18-bit banks that share the same clock and clear resources.

There is 1 set of independent clock and clear - CLK and nCLR. This set of clock and clear is available to each register bank.

Figure 2-2 shows a simplified high-level functional diagram of the DSP block for the 9bx9b-multiplier-based organization.

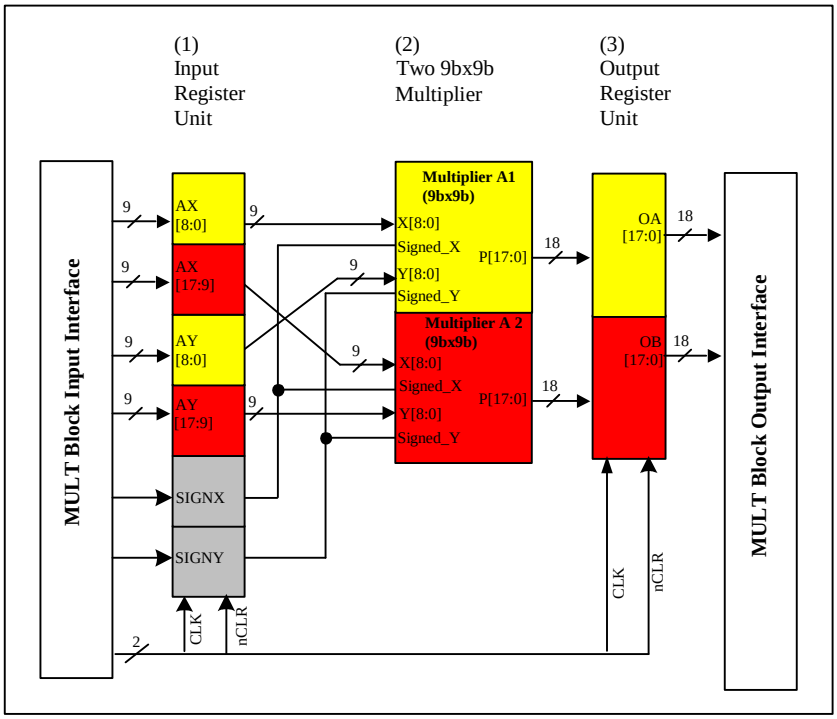

Figure 2-2: 9bx9b Multiplier Based Organization

2.3 Operation Modes

There are a total of 2 modes of operation for the DSP block to address a wide range of DSP applications. Table 2-1 defines these 2 modes as well as lists the number of data inputs and data outputs for these modes.

Table 2-1: DSP Block Modes and Data I/O Count

Mode	Description	# Data Input	# Data Output
	18 bit wide operations		
1)	**1 individual 18bx18b multipliers**	36	36
	9 bit wide operations		
2)	**2 individual 9bx9b multipliers**	36	36

3 Chip Layout

The number and layout of the DSP blocks on each Cyclone II device will be provided at a later time.

4 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.

The same megafunctions used to implement Stratix DSP blocks are used to implement Cyclone II DSP blocks, and these megafunctions are backwards compatible with Stratix devices. As a result, a megafunction instantiated for Stratix designs can also be compiled for Cyclone II designs, with adders and accumulators implemented efficiently using regular logic elements.

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.
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.

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

5 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.

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

9x9	**18x18**
1	2

5.1 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 width.

2. Divide the number by 2 to give the number of DSP blocks used in this mode, round up in case of partial DSP blocks.
3. Add the number of DSP blocks used for each width

5.2 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 multipliers
b) Ten 8x8 multipliers
c) Five 16x16 multipliers

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

a) 9 DSP elements for 9bx9b mode
b) 10 DSP elements for 9bx9b mode
c) 10 DSP elements for 18bx18b mode (= 5 x 2)

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

1) 9bx9b mode: 19 DSP elements (= 9 + 10)
2) 18bx18b mode: 10 DSP elements

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

1) 9bx9b mode: 19 / 2 = 10 DSP blocks needed (after rounding up)
2) 18bx18b mode: 10 / 2 = 5 DSP blocks needed

Adding the number of DSP blocks needed:

10 + 5 = 15 DSP blocks needed for this design.

This means 15 DSP blocks will be used for this design. The number of DSP elements that will be reported as used for this design is 29 (note that this number does not include the inflation done for the 9x9 modes). As in Stratix devices, the concept of a DSP element representing the actual used portion of a DSP block is still the same.

6 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.

6.1 DSP Block Resource Balancing

Making the decision between using DSP blocks and LEs is performed by DSP block resource balancing in the Quartus II software. 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 family. 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.

7 Inferring DSP Blocks

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

1) Simple multipliers (i.e. just the '*' operator)
2) Multipliers feeding adders
3) A multiplier feeding an accumulator

As with Stratix designs, it is recommended that the megafunctions be used to specify the inferred DSP functions instead of inferring the WYSIWYGs directly. For multipliers with adders and accumulators, ALTMULT_ACCUM and ALTMULT_ADD can be used, or as an alternative, synthesis tools can implement its own adder/accumulator logic whiling inferring LPM_MULT.