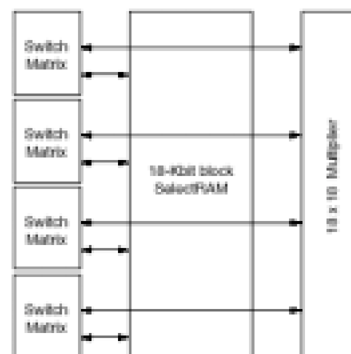
HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.co.uk
http://www.hunteng.co.uk
http://www.hunt-dsp.com

Implementing Multipliers in Xilinx Virtex II FPGAs

Rev 1.0 J.Maddocks 23-09-2003

Introduction

The Xilinx Virtex II FPGAs contain hardware multipliers. This allows multiplication to be done inside the FPGA without the use of large numbers of LUT's and also at a low power cost. These multipliers are ideally suited to performing operations such as DDC and Convolution. These multipliers are associated with a Block RAM as shown below. The multipliers have input data bus widths of 18 bits and an output data bus width of 36 bits. It is important to bear in mind when using these hardware multipliers several factors that are outlined in this document.



Common Rules

There are just a few general rules to bear in mind. When multiplying, the width of the answer will be the sum of the two input widths. For example, if the two input bus widths were 5 bits wide then the output bus width would need to be 10 bits to ensure no loss of data.
Also, signed data representation always uses the top bit to represent the sign information. That is, for a positive number the top bit, or sign bit will always be 0, and for a negative number the top bit will always be 1. When working with signed data and signed components you will need to correctly use the sign bits. The top bits of both the input data busses and output data busses of a signed multiplier must be set correctly. For example, if an 8-bit signed bus is used as an input to a 12x12 multiplier then it is important to extend the sign bit. In the 8 bit signed bus, bit 7 will be the sign bit but the 12x12 multiplier expects the sign bit at bit 11. The data must be sign extended as in the example below:

10100111 would become 111110100111
00100110 would become 000000100110

Instantiation

It is possible to instantiate two versions of this multiplier. The first is a pipelined version with registered outputs. The pipeline delay will be 1 cycle. The component declaration is as follows:

```
component MULT18X18S
  port (A : in STD_LOGIC_VECTOR (17 downto 0);
        B : in STD_LOGIC_VECTOR (17 downto 0);
        C : in STD_ULOGIC ;
        CE : in STD_ULOGIC ;
        P : out STD_LOGIC_VECTOR (35 downto 0);
        R : in STD_ULOGIC );
end component;
```

The inputs are 2's complement signed numbers as is the output. The output P is the result of A * B. C is the clock, CE is the clock enable and R is the reset.

The second possible instantiation of the mult18x18 is an un-pipelined version. The component instantiation for the un-pipelined version is:

```
component MULT18X18
  port (A : in STD_LOGIC_VECTOR (17 downto 0);
        B : in STD_LOGIC_VECTOR (17 downto 0);
        P : out STD_LOGIC_VECTOR (35 downto 0) );
end component;
```

The pipelined multiplier will run at higher speeds.

Inferring Multipliers

It is also possible to infer the un-pipelined mult18x18. In ISE 5.1 with service pack 2 or above it is possible to infer the pipelined multiplier. The simplest way to infer the un-pipelined multiplier is to write:

```
A <= B * C; where     A : signed(35 downto 0)
                      B : signed(17 downto 0)
                      C : signed(17 downto 0)
```

It is also possible to infer an unsigned multiplier but the two input bus widths must be one bit less e.g. 17 bits.

```
A <= B * C; where     A : unsigned(33 downto 0)
                      B : unsigned(16 downto 0)
                      C : unsigned(16 downto 0)
```

The fewer the number of bits on the input busses used the faster the multiplier will run. It is also possible to infer a multiplier using std_logic signals using the same principle.

```
A <= B * C; where     A : std_logic_vector(33 downto 0)
                      B : std_logic_vector (16 downto 0)
                      C : std_logic_vector (16 downto 0)
```

The following code will infer a pipelined multiplier

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

entity xcv2_mult18x18s is
Port (a : in std_logic_vector(16 downto 0);
      b : in std_logic_vector(16 downto 0);
    clk : in std_logic;
   prod : out std_logic_vector(33 downto 0));
end xcv2_mult18x18s;

architecture arch_xcv2_mult18x18s of xcv2_mult18x18s is
begin
process(clk)
begin
  if rising_edge(clk) then
    prod <= a*b;
  end if;
end process;
end arch_xcv2_mult18x18s;
```

To use the full 18 bits on the inputs the signals will need to be made signed

```
entity xcv2_mult18x18s is
Port (a : in signed(17 downto 0);
      b : in signed(17 downto 0);
    clk : in std_logic;
   prod : out signed(35 downto 0));
end xcv2_mult18x18s;
```

The pipelined multiplier will run at higher speeds although exactly how fast either multiplier will run will depend on each individual implementation and any placement and timing constraints attached to the instance.

Coregen IP Multipliers

The coregen IP multipliers have been found to produce the same results in terms of resource usage and limitations as instantiation or inference. They do have the benefit of offering some useful additional control signals. It is important to check the exact resource usage of the multiplier built using the CoreGen.

Performance and High Speed Multiplication

There are two types of multipliers used in Virtex II devices, standard and enhanced. The enhanced version offers greatly improved performance over the standard design. The following table summarises which devices have enhanced multipliers. The special markings are on the second line of the package marking.

Device	Enhanced mult. On ES	Special marking on ES	Enhanced mult. On Production	Special marking on production
2v40	N	N	Y	ALL C
2v80	Y	N/A	Y	N/A
2v250	Y	N/A	Y	N/A
2v500	Y	N/A	Y	N/A
2v1000	N	N	Y(Not all C)	**AMT or AGT**
2v1500	Y	N/A	Y	N/A
2v2000	Y	N/A	Y	N/A
2v3000	Y(Not all ES)	**AMT**	Y	ALL C
2v4000	N	N	Y(Not all C)	**AMT or AGT**
2v6000	N	N	Y(Not all C)	**AMT or AGT**
2v8000	Y	N/A	Y	N/A

Device Support and Special Markings

In order to access the enhanced multipliers the line

```
CONFIG STEPPING = "1";
```

needs to be added to the *.ucf file for the design. This is only possible with ISE 4.2i with service pack 2 or above.

Device	Possible Values	Default Value	Setting to access slower multiplier	Setting to access enhanced multiplier
2v40 **	ES,0,1	ES or 1	ES,0	1
2v80	ES,0,1	1	N/A *	1
2v250	ES,0,1	1	N/A *	1
2v500	ES,0,1	1	N/A *	1
2v1000	ES,0,1	0	ES,0	1
2v1500	ES,0,1	1	N/A *	1
2v2000	ES,0,1	1	N/A *	1
2v3000	ES,0,1	0	ES,0	1
2v4000	ES,0,1	0	ES,0	1
2v6000	ES,0,1	0	ES,0	1
2v8000	ES,0,1	1	N/A *	1

CONFIG STEPPING Values

Further information on this issue is available in the form of the Xilinx answer 14339 which can be found at:
http://support.xilinx.com/xlnx/xil_ans_display.jsp?iLanguageID=1&iCountryID=1&getPagePath=14339
The following table gives an indication of the exact performance that can be expected.

18 x 18 Pipelined Multiplier Performance

Device	Speed Grade	Unconstrained	Constrained	Units
Standard Design	-4	83	109	MHz
	-5	97	127	MHz
	-6	133	232	MHz
Enhanced Design	-4	103	180	MHz
	-5	116	205	MHz
	-6	133	232	MHz

The multipliers will run faster if fewer bits are used and the MSB's are tied to '0'. 200MHz+ performance is possible by following the Xilinx app note Xapp636 that describes how it is possible to improve the operation of the multipliers.
http://www.xilinx.com/ipcenter/catalog/search/reference/reference_xapp636_optimal_pipelining_vii-multipliers.htm

Another possible way of achieving high speeds at a cost would be to use two multipliers alternately, as while they wouldn't in fact work any faster the overall data rate through a multiplier component that accessed two multipliers would double.