Kyle Keislar

ECE524/L FPGA/ASIC Design and Optimization Using VHDL Lab



Lab 6

Arithmetic

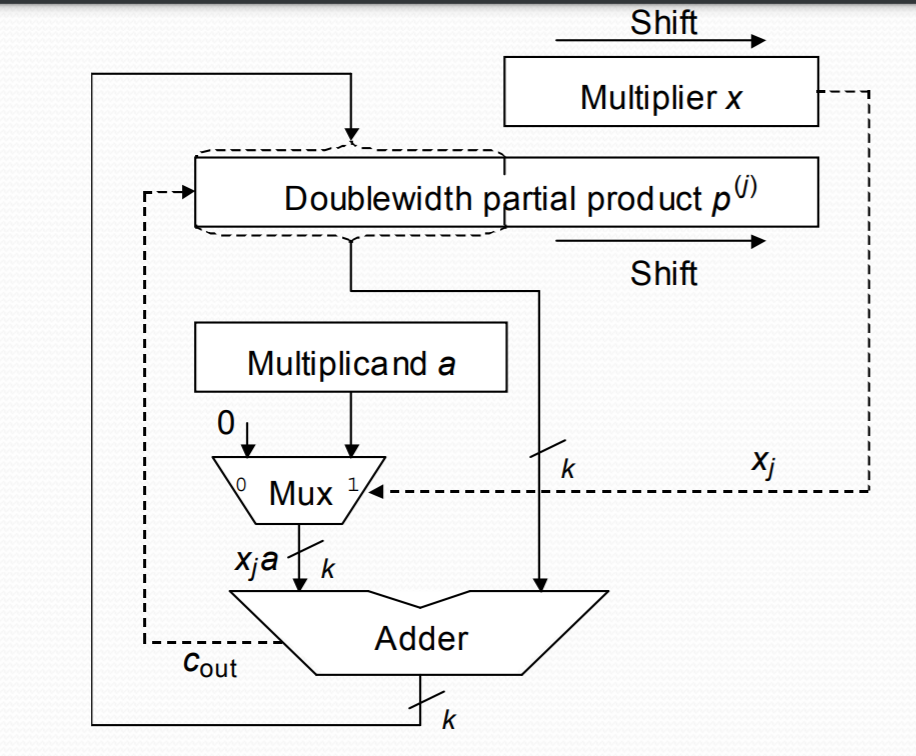
11/10/2020

## 

## Introduction & Problem Statements

In this experiment a shift-right unsigned radix 4 multiplier is designed. The multiplier handles inputs of predefined lengths whose operands are a multiplicand of 10 bits and a multiplier of 6 bits. This kind of multiplier is also know as a “High Raddix” multiplier as it computes the partial products of the multiplicand by a bit pair of the multiplier. A general high level block diagram for a fundamental radix 2 multiplier is shown below in Figure 6.0.A.

Fig 6.0.A: General Schema of the High Raddix multiplier

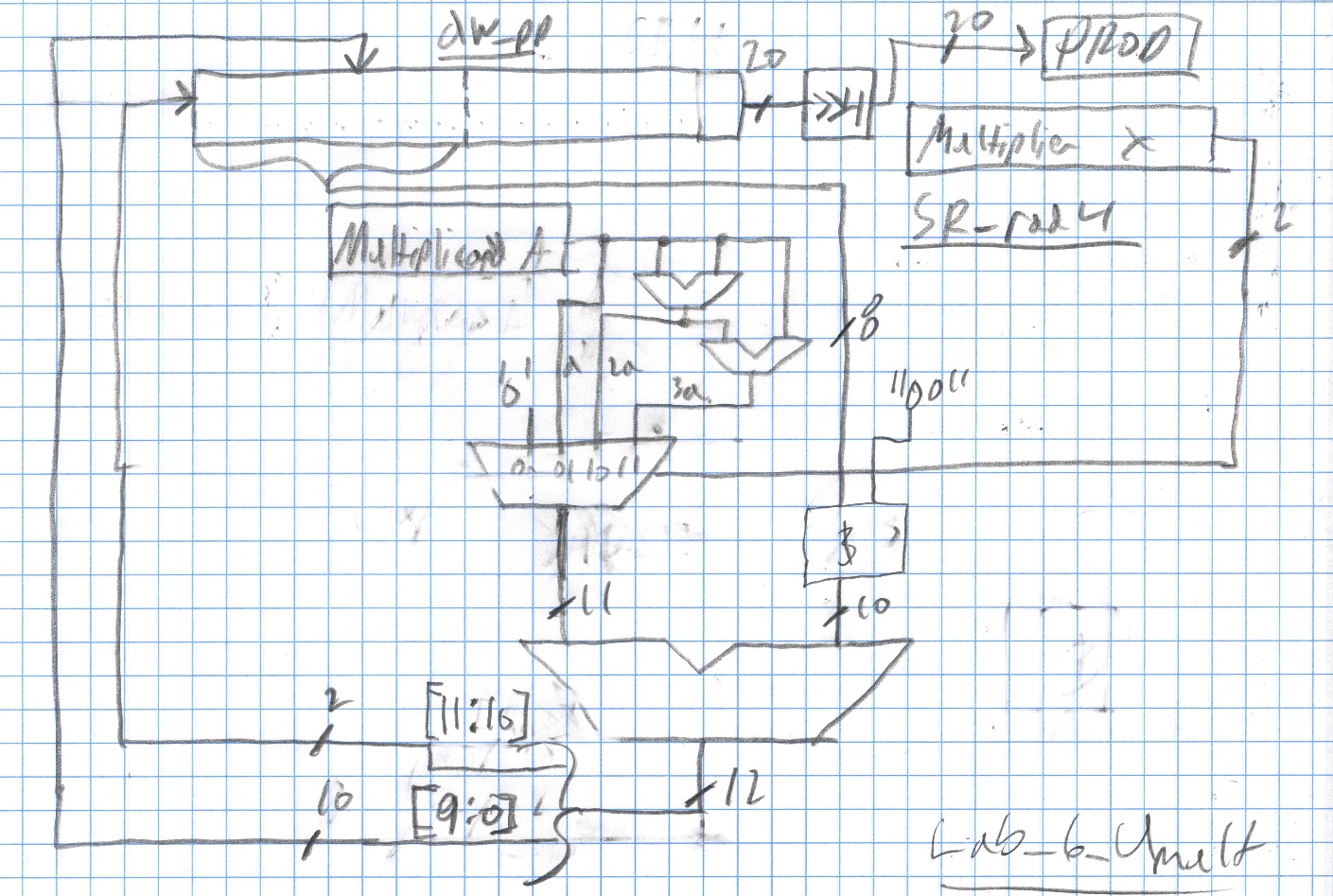


The detailed block diagram for the multiplier designed in this experiment is shown below as Fig 6.0.B in the Procedure section below. The proposed design diagram references its general counterpart above and is comprised of two modules and one top level. A testbench was also written to demonstrate the functionality of the unsigned multiplier.

## Procedure

**Task 1**: Provide the detailed block diagram of your design.

Fig 6.0.B: General Schema of the High Raddix multiplier



**Task 2**: Provide VHDL source code and VHDL testbench for the design.

The simulated design files for Task 2 can be seen in the Appendix below as items: A.2 *Lab\_6\_Umult.vhd*, A.3 *dw\_pp.vhd* and A.4 *SR\_rad4.vhd*. The test bench utilized for Task 2, can be found as A.1 *tb\_Lab\_6\_Umult.vhd*.

## Results (Data):

## *Waveforms graphed from generated & simulated data:*

Fig 6.1: Radix 4 Multiplier Test Vector products 1,2

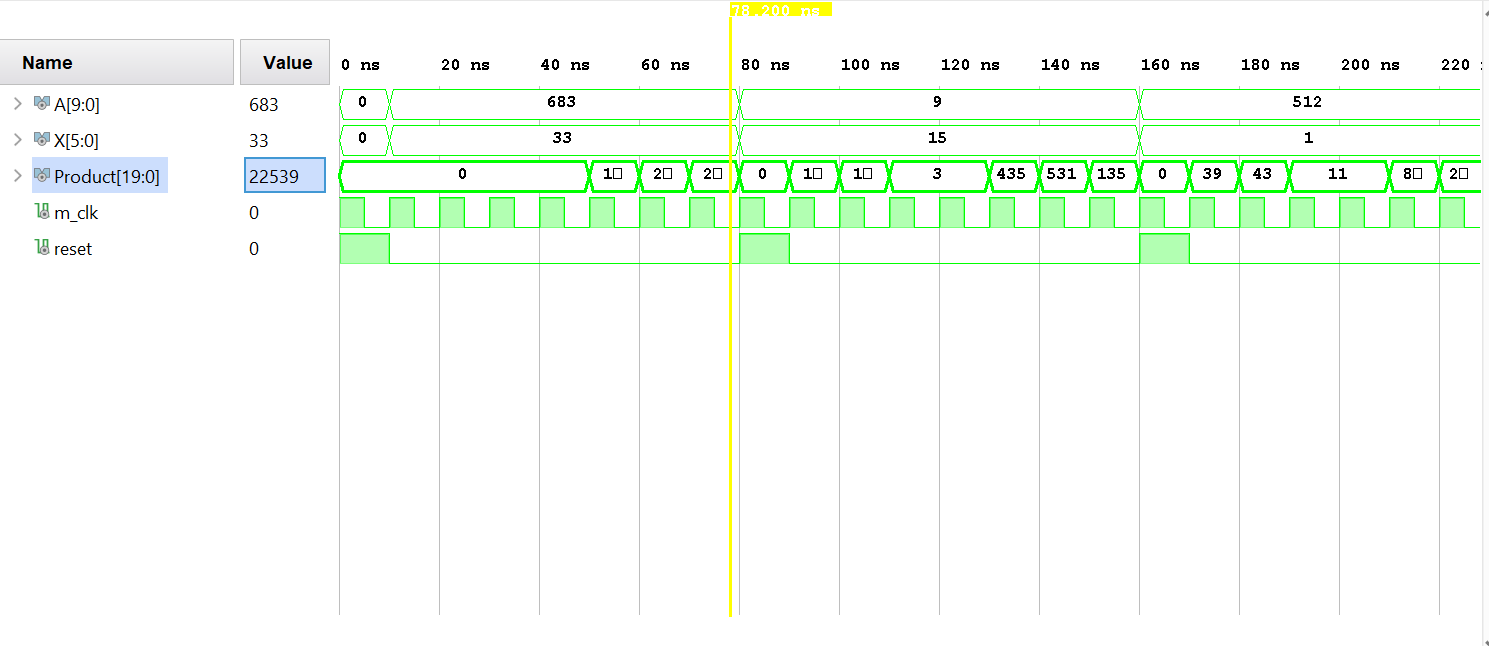


Fig 6.2: Radix 4 Multiplier Test Vector products 3,4

## 

## Analysis:

The simulated design for this experiment successfully demonstrated the functionality of the radix 4 unsigned multiplier proposed in Fig 6.0.B above.

Figure 6.1 and 6.2 show 4 collective test vector cases with different 10 bit multiplicands and 6 bit multipliers. In the first figure, the test vectors for the first two pairs and their products are shown. The first multiplier input, x, and multiplicand input a are 638 and 33 respectively. The expected result is 638 \* 33 = 22,539. This value can be seen in the simulation waveform above after the design latency of 7 clock pulses following the reset signal. The next case illustrates the multiplier functionality with a smaller multiplier whose binary signal contains the bit pair “11”. Here the multiplicand of 9 is multiplied by 15 and the result of 135 can be seen 7 clock cycles after the inputs are reset at 150 ns.

In the second figure, the test vectors for the last two pairs and their products are shown. The first multiplier and multiplicand pair is 512 and 1. The expected result is 512 and can be seen after the design latency at 230 ns. This test vector was chosen to illustrate a fundamental property of multiplication when one of the operands is 1 and the other is non-zero. The next case illustrates the multiplier behavior when one of the operands is 0. Here, the multiplier is 0 but the multiplicand is increased to its maximum value of 1023. The result is 0 as expected and can be seen present in the waveform at 310 ns.

These results confirm the basic functionality of the multiplier for the minimum and maximum bounds of testing. This design could be improved greatly by utilizing additional FPGA resources such as a DSP block. This method was not pursued in this experiment as no realized design was implemented. This would also reduce the latency of the design as multiple additions could be performed in a single clock cycle without utilizing any additional resources. More extensive testing will be required to utilize this multiplier in future designs.

APPENDIX

1. *tb\_Lab\_6\_Umult.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** ieee**.**numeric\_std**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** tb\_Lab\_6\_Umult **is**

-- Port ( );

**end** tb\_Lab\_6\_Umult **;**

**architecture** Behavioral **of** tb\_Lab\_6\_Umult **is**

--CONSTANT DEFINITIONS

**CONSTANT** X\_W**:** INTEGER **:=** 6**;** --multiplier width

**CONSTANT** A\_W**:** INTEGER **:=** 10**;** --multiplicand width

**CONSTANT** RADDIX**:** INTEGER **:=**4**;** --raddix used to compute TB\_RAD\_BITS

**CONSTANT** TB\_RAD\_BITS**:** INTEGER **:=** 2**;**

--signal defintions

**signal** A**:** STD\_LOGIC\_VECTOR**(**A\_W**-**1 **DOWNTO** 0**)** **:=** **(Others** **=>**'0'**);**

**signal** X**:** STD\_LOGIC\_VECTOR**(**X\_W**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

**signal** Product**:** STD\_LOGIC\_VECTOR**((**2**\***A\_W**)-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

--CLOCK process signal definitiions

**CONSTANT** CP**:** TIME **:=** 10ns**;**--clock period

**SIGNAL** PULSE**:** TIME**:=**CP**\***0.5**;**--pulse width

**signal** m\_clk**,**reset**:** std\_logic **:=** '0'**;** --master clock, hold initial values

--Component Definitions

**COMPONENT** Lab\_6\_Umult **is**

**GENERIC(**A\_WIDTH**:** INTEGER**:=**4**;**-- shift register bit width

X\_WIDTH**:** INTEGER**:=**4**;**

RAD\_BITS**:** INTEGER **:=**2**);** --Raddix of multiplier

**Port** **(** a**:** **in** STD\_LOGIC\_VECTOR **(**A\_WIDTH**-**1 **downto** 0**);** --multipler, multiplicand

X**:** **in** STD\_LOGIC\_VECTOR**(**X\_WIDTH**-**1 **downto** 0**);**

clk**,**rst **:** **in** STD\_LOGIC**;**

P **:** **out** STD\_LOGIC\_VECTOR **((**A\_WIDTH**\***2**)-**1 **downto** 0**));** --proudct

**end** **COMPONENT;**

**begin**

uut**:**Lab\_6\_Umult

**Generic** **Map** **(** A\_WIDTH **=>** A\_W**,**

X\_WIDTH **=>** X\_W**,**

RAD\_BITS **=>** TB\_RAD\_BITS**)**

**Port** **Map** **(** a**=>**A**,**

x**=>**X**,**

clk**=>**m\_clk**,**

rst**=>**reset**,**

P**=>** Product**);**

--Clock Processs

m\_clock**:process** --free running clock

**begin**

m\_clk **<=** '1'**;**

**wait** **for** PULSE **;**

m\_clk **<=** '0'**;**

**wait** **for** PULSE**;**

**end** **process;**

--Clock Processs

rst**:process** --fholds initilized values to show 0 behavior

**begin**

reset**<=** '1'**;**

**wait** **for** CP **;**

reset **<=** '0'**;**

**wait** **for** CP**\***7**;**

reset**<=** '1'**;**

**wait** **for** CP **;**

reset **<=** '0'**;**

**wait** **for** CP**\***7**;**

reset**<=** '1'**;**

**wait** **for** CP **;**

reset **<=** '0'**;**

**wait** **for** CP**\***7**;**

reset**<=** '1'**;**

**wait** **for** CP **;**

reset **<=** '0'**;**

**wait** **for** CP**\***7**;**

**wait;**

**end** **process;**

TVs**:process**

**begin**

**wait** **for** CP**;**

A**<=** "1010101011"**;** --683 in binary

X**<=** "100001" **;** -- 33 in Binary

**wait** **for** CP**\***7**;**

--Expected Result:

--683 \* 33 = 22,539

A**<=** "0000001001"**;** --9 in binary

X**<=** "001111" **;** -- 15 in Binary

**wait** **for** CP**\***7**;**

**wait** **for** CP**;**

--Expected Result:

--9 \* 15 = 135

A**<=** "1000000000"**;** --512 in binary

X**<=** "000001" **;** -- 15 in Binary

**wait** **for** CP**\***7**;**

**wait** **for** CP**;**

--Expected Result:

--512 \* 1 = 512

A**<=** "1111111111"**;** --1023 in binary

X**<=** "000000" **;** -- 0 in Binary

**wait** **for** CP**\***7**;**

**wait** **for** CP**;**

--Expected Result:

--1023 \* 0 = 0

**wait;**

**end** **process;**

**end** Behavioral**;**

1. *Lab\_6\_Umult.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**use** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;** --Included for convient signal incrementation

**entity** Lab\_6\_Umult **is**

**GENERIC(**A\_WIDTH**:** INTEGER**:=**4**;**-- shift register bit width

X\_WIDTH**:** INTEGER**:=**4**;**

RAD\_BITS**:** INTEGER **:=**2**);** --Raddix of multiplier

**Port** **(** a**:** **in** STD\_LOGIC\_VECTOR **(**A\_WIDTH**-**1 **downto** 0**);** --multipler, multiplicand

X**:** **in** STD\_LOGIC\_VECTOR**(**X\_WIDTH**-**1 **downto** 0**);**

clk**,**rst **:** **in** STD\_LOGIC**;**

P **:** **out** STD\_LOGIC\_VECTOR **((**A\_WIDTH**\***2**)-**1 **downto** 0**));** --proudct

**end** Lab\_6\_Umult**;**

**architecture** Behavioral **of** Lab\_6\_Umult **is**

--CONSTANTS

**CONSTANT** CONCAT**:** STD\_LOGIC\_VECTOR**(**RAD\_BITS**-**1 **downto** 0**)** **:=** **(Others** **=>** '0'**);**

**signal** zero\_mux**:** STD\_LOGIC\_VECTOR**(**11 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

**signal** a\_mux**:** STD\_LOGIC\_VECTOR**(**11 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

**signal** two\_a\_mux**:** STD\_LOGIC\_VECTOR**(**11 **downto** 0**):=** **(Others** **=>**'0'**);**

**signal** three\_a\_mux**:** STD\_LOGIC\_VECTOR**(**11 **downto** 0**):=** **(Others** **=>**'0'**);**

--SIGNALS

**signal** X\_SR\_in**:**STD\_LOGIC\_VECTOR**(**X\_WIDTH**-**1 **DOWNTO** 0**):=** **(Others** **=>**'0'**);**

**signal** temp\_x**,**X\_SR\_out**:**STD\_LOGIC\_VECTOR**(**X\_WIDTH**-**1 **DOWNTO** 0**):=** **(Others** **=>**'0'**);** --Shift Register Output for Multplier

**signal** mult\_sel**:** STD\_LOGIC\_VECTOR**(**RAD\_BITS**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --input from shift Register to Adder MUX select lines

**signal** partial\_add**:** STD\_LOGIC\_VECTOR**(**A\_WIDTH**-**1 **DOWNTO** 0**):=** **(Others** **=>**'0'**);** --Next partial product to be summated

**signal** partial\_in**,**next\_partial**:** STD\_LOGIC\_VECTOR**(**A\_WIDTH**-**1 **DOWNTO** 0**):=** **(Others** **=>**'0'**);** --Current partial sum to be stored in doublewidth partial product register

**signal** Add\_in**:** STD\_LOGIC\_VECTOR**(**11 **DOWNTO** 0**):=** **(Others** **=>**'0'**);**-- selected input based off of shifted in RAD 4, two bit pairs form X

**signal** Sum\_out**:** STD\_LOGIC\_VECTOR**(**11 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --output sum to drive c\_out, partial\_in

**signal** c\_out**,**count**:** STD\_LOGIC\_VECTOR**(**RAD\_BITS**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --carryout from adder

--COMPONENTS

--Rad4 Shift REg

**COMPONENT** SR\_rad4 **is**

**GENERIC(**WIDTH**:** INTEGER**:=**4**;**-- shift register bit width

RAD\_BITS**:** INTEGER**:=**2**);** --Amount of bits to shift for Raddix

**Port** **(** Reset**,**clk**:** **in** STD\_LOGIC**;** --0 => SR 1 => SL

SR\_in**:** **in** STD\_LOGIC\_VECTOR **(**WIDTH**-**1 **downto** 0**);**

SR\_out**:** **out** STD\_LOGIC\_VECTOR **(**WIDTH**-**1 **downto** 0**);**

SR\_mult\_out**:** **out** STD\_LOGIC\_VECTOR**(**RAD\_BITS**-**1 **downto** 0**));**

**end** **COMPONENT;**

--Double Width Partial Product Reg

**COMPONENT** dw\_pp **is**

**GENERIC(**A\_WIDTH**:** INTEGER**:=**4**;**

X\_WIDTH**:** INTEGER**:=**4**;**

RAD\_BITS**:** INTEGER**:=**2**);** -- shift register bit width

**Port** **(** clk**:** **in** STD\_LOGIC**;**

Reset **:** **in** STD\_LOGIC**;**

PP\_in**:** **in** STD\_LOGIC\_VECTOR **(**A\_WIDTH**-**1 **DOWNTO** 0**);**

carry\_in**,**count**:** **in** STD\_LOGIC\_VECTOR**(**RAD\_BITS**-**1 **downto** 0**)** **;**

Prod **:** **out** STD\_LOGIC\_VECTOR **((**A\_WIDTH**\***2**)-**1 **downto** 0**)**

**);**

**end** **COMPONENT;**

**begin**

--Component Instantiation

DBL\_P**:** dw\_pp

**Generic** **Map(** A\_WIDTH **=>** A\_WIDTH**,**

X\_WIDTH **=>** X\_WIDTH**,**

RAD\_BITS**=>**RAD\_BITS**)**

**Port** **Map(**

clk **=>** clk**,**

Reset **=>** rst**,**

PP\_in **=>** partial\_in**,**

carry\_in **=>** c\_out**,**

count**=>**count**,**

Prod **=>** P**);**

SR\_4**:** SR\_rad4

**Generic** **Map(**WIDTH**=>**X\_WIDTH**,**

RAD\_BITS**=>**RAD\_BITS**)** -- shift register bit width

**Port** **Map** **(** Reset**=>**rst**,**

clk**=>**clk**,**

SR\_in**=>**x\_SR\_in**,**

SR\_out**=>**X\_SR\_out**,**

SR\_mult\_out**=>**mult\_sel**)** **;**

Init**:process(**rst**,**a**)**

**begin**

--after reset, define multiplicand values based on A-input

**if(**rst'**event** **AND** rst**=**'0'**)** **then**

a\_mux **<=** "00"**&**a**;**--'0'& std\_logic\_vector(unsigned(a));

two\_a\_mux**<=** '0'**&**a**&** '0'**;**

three\_a\_mux**<=** std\_logic\_vector**((**"00" **&** UNSIGNED**(**a**))** **+** **(**'0'**&** UNSIGNED**(**a**)&**'0'**));**

**end** **if;**

**end** **process;**

Comp\_sum\_shift**:** **process(**clk**)**

**begin**

**if(**rst**=**'1' **)** **then**

Sum\_out**<=(Others=>**'0'**);**

X\_SR\_in**<=(Others=>**'0'**);**

count **<=** **(Others** **=>**'1'**);**

**elsif** **(**rst'**event** **AND** rst**=**'0'**)** **then**

X\_SR\_in**<=**x**;**

**else**

**if(**clk'**event** **AND** clk**=**'1'**)** **then**

--Find next partial product to add

partial\_in**<=**Sum\_Out**(**9 **downto** 0**);**

--compute next partial sum to store in doublewidth

**if(**mult\_sel**=**"00"**)** **then**

Sum\_out**<=**zero\_mux**+(**'0'**&**partial\_add**);**

**elsif** **(**mult\_sel**=**"01"**)** **then**

Sum\_out**<=**a\_mux**+(**'0'**&**partial\_add**);**

**elsif(**mult\_sel**=**"10"**)** **then**

Sum\_out**<=**two\_a\_mux**+(**'0'**&**partial\_add**);**

**elsif(**mult\_sel**=**"11"**)** **then**

Sum\_out**<=**three\_a\_mux**+(**'0'**&**partial\_add**);**

**end** **if;**

count**<=**count**+**1**;**

--shift in next multiplier bit pair

X\_SR\_in**<=** "00" **&** X\_SR\_in**(**X\_WIDTH**-**1 **downto** RAD\_BITS**);**

**end** **if;**

**end** **if;**

**end** **process;**

--Adder Multiplexer

--Sum\_out<=(Add\_in)+('0'&partial\_add);

partial\_add**<=** "00"**&**Sum\_out**(**9 **downto** 2**);**

c\_out**<=**Sum\_out**(**11 **downto** 10**);**

**with** mult\_sel **select** Add\_in **<=**

zero\_mux **when** "00"**,**

a\_mux **when** "01"**,**

two\_a\_mux **when** "10"**,**

three\_a\_mux **when** "11"**,**

Add\_in **when** **others;**

**end** Behavioral**;**

1. *Dw\_pp.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**use** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;** --Included for convient signal incrementation

**entity** dw\_pp **is**

**GENERIC(**A\_WIDTH**:** INTEGER**:=**4**;**

X\_WIDTH**:** INTEGER**:=**4**;**

RAD\_BITS**:** INTEGER**:=**2**);** -- shift register bit width

**Port** **(** clk**:** **in** STD\_LOGIC**;**

Reset **:** **in** STD\_LOGIC**;**

PP\_in**:** **in** STD\_LOGIC\_VECTOR **(**A\_WIDTH**-**1 **DOWNTO** 0**);**

carry\_in**,**count**:** **in** STD\_LOGIC\_VECTOR**(**RAD\_BITS**-**1 **downto** 0**)** **;**

Prod **:** **out** STD\_LOGIC\_VECTOR **((**A\_WIDTH**\***2**)-**1 **downto** 0**)**

**);**

**end** dw\_pp**;**

**architecture** Behavioral **of** dw\_pp **is**

**signal** temp\_p**:** STD\_LOGIC\_VECTOR **((**2**\***A\_WIDTH**)-**1 **downto** 0**):=** **(Others** **=>** '0'**);**

**signal** prev\_carry**:** STD\_LOGIC\_VECTOR**(**RAD\_BITS**-**1 **downto** 0**):=** **(Others** **=>** '0'**);**

**signal** nxt\_prod**:** STD\_LOGIC\_VECTOR **((**A\_WIDTH**\***2**)-**1 **downto** 0**):=** **(Others** **=>** '0'**);**

**begin**

**process(**clk**)**

**begin**

**if(**reset**=**'1'**)** **then**

Prod**<=** **(Others** **=>**'0'**);**

**else**

**if(**clk'**event** **AND** clk**=**'1'**)** **then**

Prod**<=**nxt\_prod**;**

prev\_carry**<=**carry\_in**;**

**case** count **is**

**when** "01" **=>**

temp\_p**(**3 **downto** 2**)** **<=** pp\_in**(**1 **downto** 0**);**

**when** "10" **=>**

temp\_p**(**5 **downto** 4**)** **<=** pp\_in**(**1 **downto** 0**);**

**when** "11" **=>**

temp\_p**(**7 **downto** 6**)** **<=** pp\_in**(**1 **downto** 0**);**

**when** **others** **=>**

temp\_p**<=**temp\_p**;**

**end** **case;**

**end** **if;**

**end** **if;**

**end** **process;**

nxt\_prod**(**15 **downto** 0**)<=** prev\_carry**&**pp\_in**&**temp\_p**(**7 **downto** 4**);**

**end** Behavioral**;**

1. *SR\_rad4.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**use** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;** --Included for convient signal incrementation

**entity** SR\_rad4 **is**

**GENERIC(**WIDTH**:** INTEGER**:=**4**;**-- shift register bit width

RAD\_BITS**:** INTEGER**:=**2**);** --Amount of bits to shift for Raddix

**Port** **(** Reset**,**clk**:** **in** STD\_LOGIC**;** --0 => SR 1 => SL

SR\_in**:** **in** STD\_LOGIC\_VECTOR **(**WIDTH**-**1 **downto** 0**);**

SR\_out**:** **out** STD\_LOGIC\_VECTOR **(**WIDTH**-**1 **downto** 0**);**

SR\_mult\_out**:** **out** STD\_LOGIC\_VECTOR**(**RAD\_BITS**-**1 **downto** 0**));**

**end** SR\_rad4**;**

**architecture** Behavioral **of** SR\_rad4 **is**

--signal SR\_nxt: STD\_LOGIC\_VECTOR (WIDTH-1 downto 0);

**signal** temp**,**shift**:** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**):=** **(Others** **=>** '0'**);**

**begin**

**process** **(**clk**,**Reset**)**

**begin**

**if** **(**Reset **=** '1'**)** **then**

SR\_out **<=** **(others** **=>** '0'**);**

SR\_mult\_out **<=** **(others** **=>**'0'**);**

**end** **if;**

**if(rising\_edge(**clk**)** **AND** Reset **=**'0'**)** **then**

SR\_mult\_out **<=** SR\_in**(**RAD\_BITS**-**1 **downto** 0**);**

**for** i **in** 0 **to** WIDTH**-(**RAD\_BITS**+**1**)** **loop**

SR\_out**(**i**)** **<=** Sr\_in**(**i**+(**RAD\_BITS**));**

**end** **loop;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**