TALLINN UNIVERSITY OF TECHNOLOGY

COURSE CODE: **IAS0600**

(DIGITAL SYSTEMS DESIGN WITH VHDL)

LABORATORY REPORT NO 5

(**TOPIC:** PARAMETERIZABLE ADDER AND MULTIPLIER)

Student: Dismas EZECHUKWU (**184603IVEM**)

Instructors:

Alexander Sudnitson (Associate Professor)

Dmitri Mihhailov (Research Scientist)

DATE: 17/12/2018

1. INTRODUCTION

A parameterizable Adder and multipliers are multipliers and adders where the number of bit of the inputs operands can be varied at will. In this laboratory work, a parameterizable ripple carry adder will be developed and it will be used as component to also develop a parameterizable multiplier. The implementation is shown graphically as columnar addition in figure 2. This is because the multiplication function can be reduced to a set of additions that is properly carried out.

The steps and task involved in the creation of this project work is itemized in the task section;

1.2 TASK

Implement a parameterizable multiplier using for/generate statement to describe repeating components/assignments. The size of the multiplier should be adjustable using parameter data width (possible values are 2, 3, 4, ... , 8). Following is the list of steps for the parameterizable multiplier design:

- Describe a parameterizable ripple-carry adder. It should be possible to adjust its size using parameter data width (possible values are 2, 3, 4, ... , 8).

- Optionally, add a carry-lookahead adder description to the previous design. In this case the adder design should have another adjustable parameter adder type (possible types are ripple carry, carry-lookahead).

- Use parameterizable adder as component to describe a parameterizable multiplier that implements multiplication using columnar addition. It should be possible to adjust its size using parameter data width (possible values are 2, 3, 4, ... , 8). Simulate and implement each design step on FPGA development board

2. BACKGROUND

Ripple Carry Adder

A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage [1].

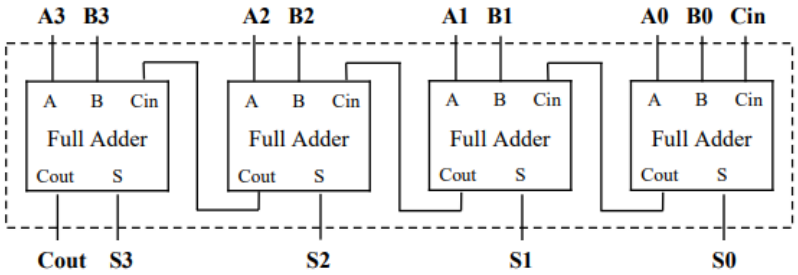

Figure 1. Ripple Carry Adder

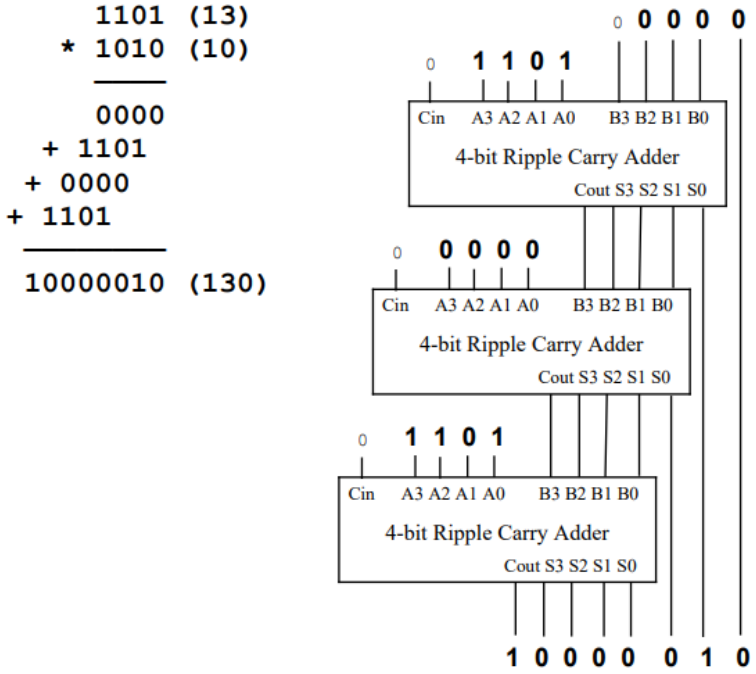

Figure 2. Multiplier, using adder as component

3. WORKFLOW

The laboratory work was carried out using the vivado software an it was simulated and implemented on the xillinx FPGA training board. Here, the ripple carry adder was first done and then the ripple carry adder was now used as a component to in the implementation of the parameterizable multiplier. The for/generate statement was used in both cases.

4. RESULTS AND DISCUSSION

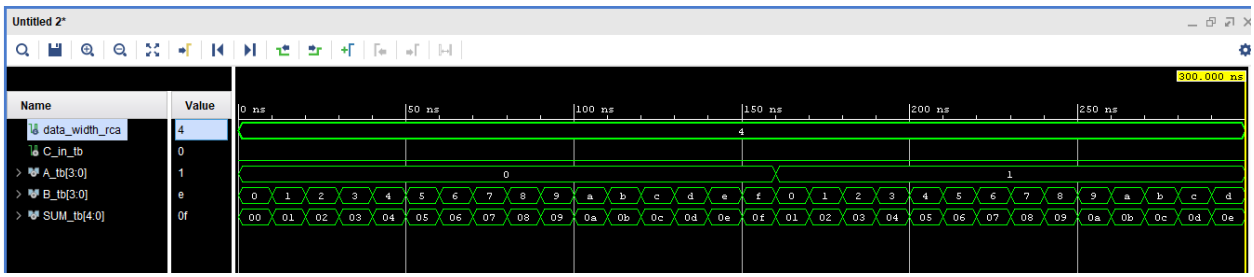

Figure 3. Simulation result of the ripple carry adder.

As seen in the Figure 3 above, the datawidth for this design is 4, “C_in_tb” contain the Carry-in, “A_tb” and “B-tb” are the inputs while SUM_tb is the sum of the inputs. The sum is 1 bit more that the inputs’ so as to accommodate the Carry-out as I didn’t make an extra carry-out port.

```
32          SUM=>SUM_tb);
33
34      process
35      begin
36      for k in 0 to (2**data_width_RCA)-1 loop
37        A_tb<=std_logic_vector(to_unsigned(k,data_width_RCA));
38        for h in 0 to (2**data_width_RCA)-1 loop
39          B_tb<=std_logic_vector(to_unsigned(h,data_width_RCA));
40          wait for 10 ns;
41          assert(SUM_tb<=std_logic_vector(unsigned(A_tb)+unsigned(B_tb)))
42          report "Testing at A("&integer'image(k)&")"&" and B("&integer'image(h)&") failed"
43          severity error;
44        end loop;
45      end loop;
46      wait;
```

Figure 4. Testbench of the ripple adder.

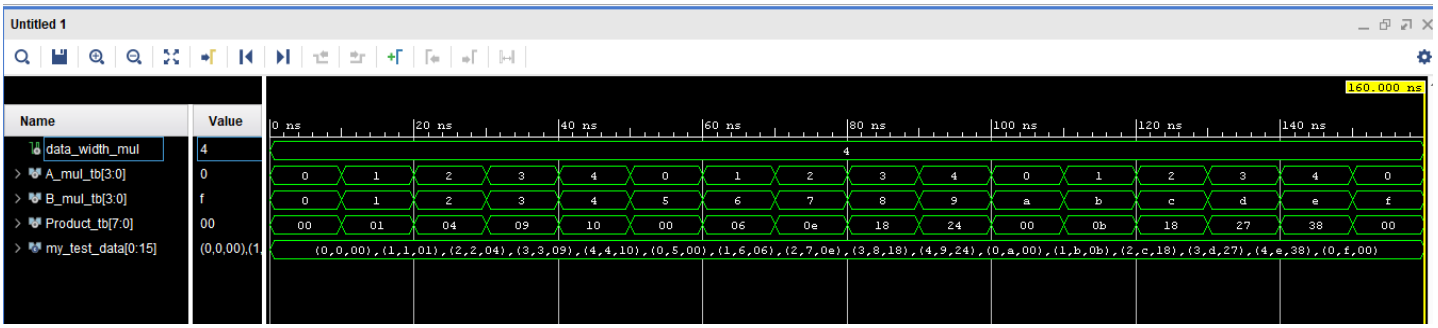

Figure 5. Simulation result of the ripple multiplier.

The Figure 5 above shows the simulation result of the multiplier, the data-width remain 4, the inputs also have 4 bits each, while the product is twice as number of bits as the inputs. The test data is a record which have been initialized with 16 different values for each member of the records. The code snippet for this is presented in figure 6 below.

```
30      type my_testings_val is array (natural range <>)  of my_testings;
31      constant my_test_data: my_testings_val:=(
32                  ("0000","0000","00000000"),--0 x 0 = 0
33                  ("0001","0001","00000001"),--1 x 1 = 1
34                  ("0010","0010","00000100"),--2 x 2 = 2
35                  ("0011","0011","00001001"),--3 x 3 = 9
36                  ("0100","0100","00010000"),--4 x 4 = 16
37                  ("0000","0101","00000000"),--0 x 5 = 0
38                  ("0001","0110","00000110"),--1 x 6 = 6
39                  ("0010","0111","00001110"),--2 x 7 = 14
40                  ("0011","1000","00011000"),--3 x 8 = 24
41                  ("0100","1001","00100100"),--4 x 9 = 36
42                  ("0000","1010","00000000"),--0 x 10(or a) = 0
43                  ("0001","1011","00001011"),--1 x 11(or b) = 11
44                  ("0010","1100","00011000"),--2 x 12(or c) = 24
45                  ("0011","1101","00100111"),--3 x 13(or d) = 39
46                  ("0100","1110","00111000"),--4 x 14(or e) = 56
47                  ("0000","1111","00000000") --0 x 15(or f) = 0
48                  );
49
50      begin
51      uut: Parameterizable_Multiplier port map(
52      A_mul=>A_mul_tb,
53      B_mul=>B_mul_tb,
54      Product=>Product_tb);
55
56      process
57      begin
58      for k in my_test_data'range loop
59          A_mul_tb<=my_test_data(k).A;
60          B_mul_tb<=my_test_data(k).B;
61          wait for 10 ns;
62          assert(Product_tb<=my_test_data(k).Product)
63          report "my_testings"&integer'image(k)&"failed"
64          severity error;
65      end loop;
66      wait;
67      end process;
```

Figure 6. Testbench of the multiplier.

The Figure 6 shows the testbench for the multiplier, it contain a record whose members have been initialized already with 16 different values.

```
-- Utilizing the Ripple Carry Adder to perform the multiply function
myProduct_sigg(0)<='0'&addings(0);
multiplication_block: for i in 1 to data_width_mul-1 generate
      n_bits_adder: component Ripple_Carry_Adder port map(
        C_in=>'0',  --completed
        B=>myProduct_sigg(i-1)((data_width_mul) downto 1),
        A=>addings(i), --completed
        SUM=>myProduct_sigg(i)
        );
end generate;
```

Figure 7. code snippet for the multiplier

The code section here shows the use of for/generate statement to implement the multiplier using the ripple carry adder as component. Here, the “C_in” signal is tied to the ground throughout the different instances of the for/generate statement while the previous value of the sum signal is used to feed in the B input.

```
begin
--generating the partial values to be added
ands: for j in 0 to data_width_mul-1 generate
   ands2: for k in 0 to data_width_mul-1 generate
    addings(j)(k) <= B_mul(j) and A_mul(k);
   end generate;
end generate;
```

Figure 8. code snippet of the multiplier.

Here the section of the code was used to get the values to be used in the columnar addition, these vales were gotten by used a nested for/generate statement while ANDing each member of the “B_mul” input with every member of the “A_mul” and the value is stored in the signal “addings” which is an array of std_logic_vectors.

```
Product((2*data_width_mul-1) downto data_width_mul-1)<=myProduct_sigg(data_width_mul-1);
```

Figure 9. Code section of the Product value assignment.

The above figure 9 shows how the first five (5) most significant bit of the final product was gotten

```
finally:for k in 0 to data_width_mul-2 generate
    Product(k)<=myProduct_sigg(k)(0);
end generate;
```

Figure 10. Code section of the Product value assignment.

The Figure 10 above shows how the remaining least significant bits of the product were gotten.

5. CONCLUSION

In this project, we have been able to realize a ripple carry adder using the for/generate statement. Also we also used this developed ripple carry adder as a component in the design of the multiplier. Both designs have been tested with different datawidth, simulated and implemented on the NEXYS 4 FPGA development board developed by Xillinx

6. REFERENCES

[1] Ripple Carry Adder, 2018. “http://www.circuitstoday.com/ripple-carry-adder”