COE 405 **Dr. Aiman El-Maleh**

HW#3 Solution

Q.1. It is required to model in VHDL a parametrizable **n-bit *Ripple Carry Adder***.

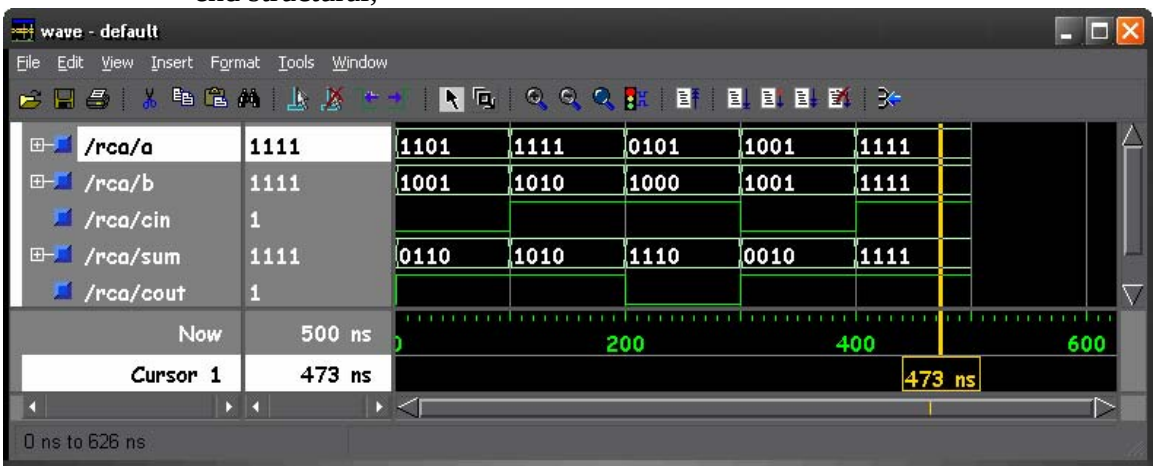
(i) Describe the entity of the n-bit adder using GNERIC for passing the adder size.

```
entity FA is
   port (a, b, cin    : in BIT;
         sum, cout  : out BIT);
end FA;

-- description of adder using concurrent signal assignments
architecture behave of FA is
begin
   sum <= (a xor b) xor cin;
   cout <= (a and b) or (cin and a) or (cin and b);
end behave;
                ==============

entity RCA is
   generic(N : integer := 4);
   port (a    : in BIT_vector(N downto 1);
         b    : in BIT_vector(N downto 1);
         cin  : in BIT;
         sum  : out BIT_vector(N downto 1);
         cout : out BIT);
end RCA;
```

(ii) Model architecture for the n-bit adder using GENERATE statement.

```
-- structural implementation of the N-bit adder
Use work.HW3.all;
architecture structural of RCA is
          FOR ALL : n1 USE ENTITY WORK.FA;
         signal carry : BIT_vector(0 to N);
 begin
   carry(0) <= cin;
   cout <= carry(N);

   -- instantiate a single-bit adder N times
   gen: for I in 1 to N generate
```

```
                add: n1 port map(
                        a => a(I),
                        b => b(I),
                        cin => carry(I - 1),
                        sum => sum(I),
                        cout => carry(I));
            end generate;
        end structural;
```



(iii) Define a package called HW3 where you store all used components, functions and procedures inside the package and use the package when needed.

```
-------------------------------------------------
--                  PACKAGE DECLERATION
-------------------------------------------------
Package HW3 is
Type integers IS ARRAY (0 to 9) OF INTEGER;

COMPONENT n1 port (a : in BIT;b : in BIT; cin : in BIT;sum : out BIT;cout : out BIT); END
COMPONENT ;
component RCA
        generic(N : integer);
        port (a    : in Bit_vector(N downto 1);
            b    : in Bit_vector(N downto 1);
            cin  : in Bit;
            sum  : out Bit_vector(N downto 1);
            cout : out Bit);
   end component;

Procedure Int2Bin (Int: IN Integer; Bin : OUT BIT_VECTOR);
Procedure Bin2Int (Bin : IN BIT_VECTOR;  Int: OUT Integer);
Procedure Apply_Data ( Signal Target: OUT Bit_Vector;                Constant Values: IN Integers;
        Constant Period: IN Time) ;

END HW3;

-------------------------------------------------
```

```
--                      PACKAGE BODY
-------------------------------------------------

Package Body HW3 IS

Procedure Int2Bin (Int: IN Integer; Bin : OUT BIT_VECTOR)  IS
        Variable Tmp: Integer;
Begin
        Tmp := Int;
        For I IN 1 To (Bin'Length) Loop
                If ( Tmp MOD 2 = 1) Then
                        Bin(I) :='1' ;
                Else Bin(I) := '0' ;

                End If;
                Tmp := Tmp / 2;
        End Loop;
End Int2Bin;

Procedure Bin2Int (Bin : IN BIT_VECTOR;  Int: OUT Integer)  IS
        Variable Result: Integer;
Begin
        Result := 0;
        For I  IN Bin'RANGE  Loop
                If Bin(I) = '1' Then
                        Result := Result + 2**I;
                End If;
        End Loop;

        Int := Result / 2;
End Bin2Int;

Procedure Apply_Data (
                Signal Target: OUT Bit_Vector ;
                Constant Values: IN Integers;
                Constant Period: IN Time) IS

                Constant n : INTEGER := target'length;
                VARIABLE j: INTEGER;
                VARIABLE tmp, pos : INTEGER :=0;
                Variable Buf: Bit_Vector (N Downto 1);
Begin
                For I IN 0 To (integers'length -1) Loop
                        tmp := values (i);
                        j:= 1;
                        WHILE j <= N LOOP
                          IF(tmp MOD 2 = 1) THEN
                             buf (j) := '1';
                          else buf(j) :='0';
                          end if;
                          tmp:= tmp /2;
                          j:= j +1;
                          END LOOP;
                          target <= TRANSPORT buf AFTER i * period;
                          END LOOP;
End Apply_Data;
```

```
end HW3;
```

(iv) Write a test bench for testing the n-bit ripple carry adder assuming that the input arguments will be specified using the procedure Apply_Data given in the book..

```
entity testHW3 is
end;

--------------------------------------------------------------------------
-- testbench for 4-bit adder
--------------------------------------------------------------------------
Use work.HW3.all;
architecture RCA4 of testHW3 is
FOR ALL : RCA USE ENTITY WORK.RCA;
   constant N : integer := 4;
   signal a    : Bit_vector(N downto 1);
   signal b    : Bit_vector(N downto 1);
   signal cin  : Bit;
   signal sum  : Bit_vector(N downto 1);
   signal cout : Bit;
begin
a1: RCA generic map (n)
          port map ( a, b, cin, sum, cout);

          apply_data(a, 00&05&06&08&10&11&09&03&15&14, 100 ns);
          apply_data(b, 01&06&07&02&05&01&09&11&01&00, 100 ns);
          cin <= '1', '0' AFTER 500 ns;
END RCA4;
```

(v) Write a test bench for testing the n-bit ripple carry adder assuming that the input arguments will be read as integers from an input file and that the output will be stored as integer in an output file. Use TEXTIO package for this purpose. **Hint**: assignments scheduled on signals inside a process take effect after encountering a wait statement. Use the statement **wait for 0 ns** as many times as needed for this purpose.

```
entity testHW32 is
end;

--------------------------------------------------------------------------------
-- testbench for 4-bit adder
--------------------------------------------------------------------------------
Use work.HW3.all;
USE STD.TEXTIO.ALL;

architecture RCA4 of testHW32 is
constant N : integer := 4;

   signal a    : Bit_vector(N downto 1);
   signal b    : Bit_vector(N downto 1);
   signal cin  : Bit;
   signal sum  : Bit_vector(N downto 1);
   signal cout : Bit;
   FOR a1: RCA USE ENTITY WORK.RCA;

begin
        Process
                File Infile : Text IS IN "example1.txt" ;
                File Outfile: Text IS OUT "Outfile1.txt" ;
                Variable Out_Line, My_Line : Line;
                Variable Int_Val, Int_Value : Integer;
                Variable Buf1: Bit_Vector (N Downto 1);
                Variable Buf2: Bit_Vector (N Downto 1);
                Variable BufCin : Bit_Vector (1 Downto 1);
                Begin
                        While Not (Endfile(Infile)) Loop

                                Readline( Infile, My_Line);  -- read a line from the input file
                                Read( My_Line, Int_Val);         -- read a value from the line
                                Int2Bin (Int_Val, BufCin);

                                cin <= BufCin(1);
                                wait for 0 ns;
                                wait for 0 ns;

                                --Readline( Infile, My_Line);  -- read a line from the input file
                                Read( My_Line, Int_Val);         -- read a value from the line
```

```
                                Int2Bin (Int_Val, Buf1);

                                a <= Buf1;
                                wait for 0 ns;
                                wait for 0 ns;


                                --Readline( Infile, My_Line);  -- read a line from the input file
                                Read(My_Line, Int_Val);          -- read a value from the line
                                Int2Bin (Int_Val, Buf2);

                                b <= Buf2;
                                wait for 0 ns;
                                wait for 0 ns;

                                -- To let the input settle
                                wait for 100 ns;


                                Bin2Int (sum , Int_Val);
                                Write( Out_Line, Int_Val);                    -- write the squared value to
the line
                                Write( Out_Line, cout);             -- write the squared value to the line
                                WriteLine( Outfile, Out_Line);      -- write the line to the output file

                        End Loop;
                        wait;
                End Process;
                a1: RCA generic map (n)
                        port map ( a, b, cin, sum, cout);
END RCA4;
```

==========

THE INPUT FILE " example1.txt" :

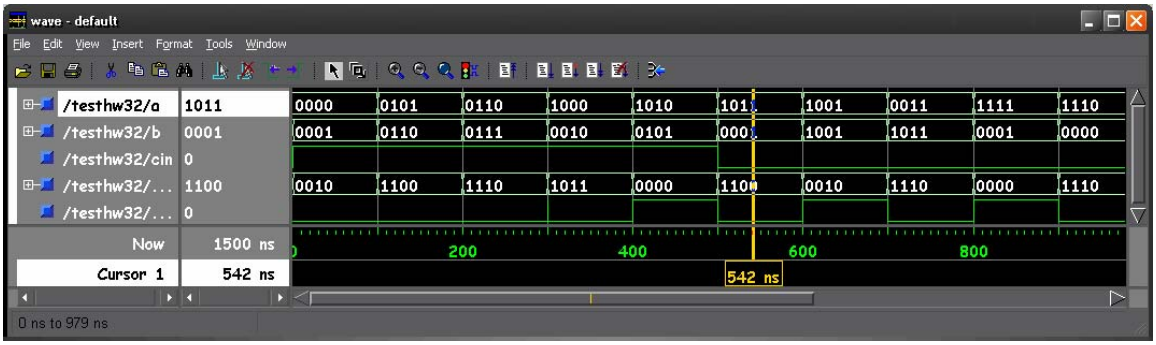
```
1 0 1
1 5 6
1 6 7
1 8 2
1 10 5
0 11 1
0 9 9
0 3 11
0 15 1
0 14 0
```

THE OUTPUT FILE " Outfile1.txt" :

```
20
120
140
110
01
120
21
```

140
01
140

The Last digit from each line is the carry out.



Q.2. It is required to model in VHDL a parametrizable **n-bit *Array Multiplier***.
(i) Describe the entity of the n-bit array multiplier using GNERIC for passing the adder size. Note that while the multiplier inputs are n-bits each, the multiplier output is 2n bits.

```
entity AMul IS
generic (         N : INTEGER := 4);
port (    A, B : IN BIT_VECTOR(N downto 1);
          SUM : OUT  BIT_VECTOR(2*N downto 1));
end AMul;
```

(ii) Model architecture for the n-bit array multiplier using GENERATE statement. Use the n-bit ripple carry adder modeled in Q1 in your solution.

```
USE WORK.HW3.ALL;
architecture Mine of AMul is
component RCA
          generic(N : integer);
          port (a,b    : in Bit_vector(N downto 1);
               cin  : in Bit;
               sum  : out Bit_vector(N downto 1);
               cout : out Bit);
    end component;

type  MULARRAY  is array  (N+1 downto 1) of BIT_VECTOR((2*N+1) downto 1);
signal ANDING_BITS, FIRST, SECOND : MULARRAY;
```

```
signal TEMP_CIN, TEMP_COUT : BIT_VECTOR ((N+1) downto 1);
constant SIZE : INTEGER := 2*N+1;

BEGIN

FIRST_NUMBER_BITS: FOR I IN 1 TO N generate
begin
        SECOND_NUMBER_BITS: FOR J IN  1 TO N generate
        begin
                ANDING_BITS(I)(J+I-1) <= A(J) and B(I);
        end generate;
end generate;

-- first row
FIRST(1) <= ANDING_BITS(1);

TEMP: FOR I IN 1 TO N generate
begin
        SECOND(I) <= ANDING_BITS(I+1);
        SUMMING: RCA generic map (SIZE) port map (FIRST(I), SECOND(I),
        TEMP_CIN(I), FIRST(I+1), TEMP_COUT(I));
        TEMP_CIN(I+1) <= TEMP_COUT(I);
end generate;

sum <= FIRST(N+1)(2*N downto 1);
end mine;
```

(iii) Modify the test bench developed for the adder in Q1(v) to test the n-bit multiplier. Assume that the input arguments will be read as integers from an input file and that the output will be stored as integer in an output file. Use TEXTIO package for this purpose.

```
entity testMul is
end;

--------------------------------------------------------------------------
-- testbench for 4-bit adder
--------------------------------------------------------------------------
Use work.HW3.all;
USE STD.TEXTIO.ALL;

architecture mul of testMul is
constant N : integer := 4;

   signal a    : Bit_vector(N downto 1);
   signal b    : Bit_vector(N downto 1);
```

```
   signal product  : Bit_vector(2*N downto 1);
   FOR a1: AMul USE ENTITY WORK.AMul;

begin
        Process
                File Infile : Text IS IN "example2.txt" ;
                File Outfile: Text IS OUT "Outfile2.txt" ;
                Variable Out_Line, My_Line : Line;
                Variable Int_Val, Int_Value : Integer;
                Variable Buf1: Bit_Vector (N Downto 1);
                Variable Buf2: Bit_Vector (N Downto 1);
                Begin
                        While Not (Endfile(Infile)) Loop

                                Readline( Infile, My_Line);  -- read a line from the input
        file
                                Read( My_Line, Int_Val);            -- read a value from the
        line
                                Int2Bin (Int_Val, Buf1);

                                a <= Buf1;
                                wait for 0 ns;
                                wait for 0 ns;


                                --Readline( Infile, My_Line);  -- read a line from the input
        file
                                Read(My_Line, Int_Val);             -- read a value from the
        line
                                Int2Bin (Int_Val, Buf2);

                                b <= Buf2;
                                wait for 0 ns;
                                wait for 0 ns;

                                wait for 100 ns;


                                Bin2Int (product , Int_Val);
                                Write( Out_Line, Int_Val);          -- write the squared
        value to the line
                                WriteLine( Outfile, Out_Line);      -- write the line to the
        output file
                End Loop;
                wait;
         End Process;
```

```
        a1: AMul generic map (n)
                port map ( a, b, product);
END mul;
```

THE INPUT FILE " example2.txt" :

15 15
9 13
8 4
12 3
6 6
1 1
15 0
11 11
13 12
10 10

THE OUTPUT FILE " Outfile2.txt" :

225
117
32
36
36
1
0
121
156
100