COE 405, Term 062

Design & Modeling of Digital Systems

HW# 4 Solution

Due date: Saturday, May12, 2007

Q.1. It is required to model an **N-bit Serial Multiplier**. The VHDL Entity description of the n-bit multiplier is given below:

Entity multiplier is
Generic (n: Positive := 4);
Port(clk: IN std_logic; dataready: IN std_logic; A : IN std_logic_vector (n-1 downto 0); B : IN std_logic_vector (n-1 downto 0);
busy, done: OUT std_logic; result: OUT std_logic_vector (2*n-1 downto 0));
End multiplier;

The circuit receives two n-bit operands when the input *dataready* becomes 1. This causes the multiplication process to begin and the *busy* flag to become active. Using the **add-shift** method, the multiplier takes one or two clock cycles for each bit of the multiplicand. When the process is completed, *done* becomes 1 for one clock cycle and *busy* returns to 0. The circuit receives two operands from its *X* and *Y* inputs and produces the result on its 2n-bit *result* output.

(i) Develop a behavioral model of the n-bit Serial Multiplier. Divide your multiplier into a data-path and control unit and model each one separately using behavioral modeling style.

It is possible to model serial multiplier behaviorally without splitting the control unit and data path as shown below:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
Entity multiplier is
    Generic (n: Positive := 4);
    Port(clk: IN std_logic; dataready: IN std_logic; A : IN          std_logic_vector (n-1 downto 0); B
: IN          std_logic_vector (n-1 downto 0);
busy, done: OUT std_logic; result: OUT          std_logic_vector ( 2*n-1 downto 0));
End multiplier;

architecture Behavioral of multiplier is
signal ac1, ac2 : std_logic_vector (n-1 downto 0);
signal extra : std_logic_vector (n downto 0);
```

```
begin
process(clk)
variable state : integer range 1 to 5 := 1;
variable count : integer range 0 to n := 0;
begin
if (clk'event and clk = '1') then
case state is
when 1 =>
     count := 0;
     extra <= (others => '0');
     ac1 <= A; ac2 <= B;
     done <= '0';
     if dataready = '1' then
       state := state + 1;
     end if;
when 2 =>
     busy <= '1';
     if ac1(0) = '1' then
        state := 3;
     else state := 4;
     end if;
when 3 =>
     extra <= extra + ('0'&ac2);
     state := 4;
when 4 =>
     extra <= '0' & extra(n downto 1);
     ac1 <= extra(0) & ac1(n-1 downto 1);
     count := count + 1;
     if count = n then state := 5;
     else state := 2;
     end if;
when 5 =>
      done <= '1';
      busy <= '0';
      state := 1;
end case;
end if;
end process;
result <= extra(n-1 downto 0)&ac1;
end Behavioral;
```

Next, we will show separate modeling of the control unit and data path using behavioral modeling as required by the question.

```
Library IEEE;
use ieee.std_logic_1164.all;
Entity multiplier is
     Generic (n: Positive := 8);
     Port(clk: IN std_logic; dataready: IN std_logic; A : IN          std_logic_vector (n-1 downto 0); B
: IN          std_logic_vector (n-1 downto 0);
busy, done: OUT std_logic; result: OUT          std_logic_vector ( 2*n-1 downto 0));
End multiplier;
Architecture DF of multiplier is
Component DPath_Multiplier
Generic(n: Positive:= 8);
Port(LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, Clk: IN std_logic;
    CEN, AC1RB: OUT std_logic;
    A, B: in std_logic_vector(N-1 DownTo 0);
```

```
    result: out std_logic_vector(2*n-1 DownTo 0));
End Component;
Component CU_Multiplier
Port(clk, dataready, AC1RB, CEN: IN std_logic ;
    LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, Busy, Done: OUT
std_logic);
End Component;
Signal LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, CEN, AC1RB:
std_logic;
Begin
dpu: DPath_Multiplier Generic Map(N) PORT MAP (LD_AC1R, LD_AC2R, ClearExtraR,
LD_ExtraR, Shift, INC, CLRC, Clk, CEN, AC1RB, A, B, result);
cu: CU_Multiplier  PORT MAP (clk, dataready, AC1RB, CEN, LD_AC1R, LD_AC2R, ClearExtraR,
LD_ExtraR, Shift, INC, CLRC, Busy, Done);
End;

Library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

Entity DPath_Multiplier is
Generic(N: Positive:= 8);
Port(LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, Clk: IN std_logic;
    CEN, AC1RB: OUT std_logic;
    A, B: in std_logic_vector(N-1 DownTo 0);
    result: out std_logic_vector(2*N-1 DownTo 0));
End;
Architecture BV of DPath_Multiplier is
Constant LN: Positive:=N;
Signal AC1R, AC2R : std_logic_vector(N-1 DownTo 0);
Signal ExtraR: std_logic_vector(N DownTo 0);
Signal CounterR, t2: std_logic_vector(LN-1 DownTo 0);

Begin

AC1Reg: Process(clk)
    Begin
            if (clk'event and clk='0') Then
                    if (LD_AC1R='1') Then
                            AC1R <= A;
                    Elsif (Shift='1') Then
                            AC1R <= ExtraR(0) & AC1R(N-1 Downto 1);
                    End if;
            end if;
            AC1RB <= AC1R(0);
    end process;
AC2Reg: Process(clk)
    Begin
            if (clk'event and clk='0') Then
                    if (LD_AC2R='1') Then
                            AC2R <=  B;
                    end if;
            end if;
    end process;
ExtraReg: Process(clk)
    Begin
            if (clk'event and clk='0') Then
               if (LD_ExtraR='1') Then
```

```
                    ExtraR <= ExtraR + ('0'&AC2R);
                Elsif  (ClearExtraR='1') Then
                    ExtraR <=(N Downto 0 => '0');
                Elsif (Shift='1') Then
                    ExtraR <= ('0' & ExtraR(N Downto 1));
                End if;
            end if;
    end process;
CounterReg: Process(clk)
    Begin
            if (clk'event and clk='0') Then
                t2 <= (LN-2 Downto 0 =>'0')&'1';
                if (INC='1') Then
                    CounterR <=  CounterR + t2;
                elsif (CLRC='1' ) Then
                    CounterR <= (LN-1 Downto 0 => '0');
                end if;
            end if;
            if (CounterR=N-1) Then
                CEN <= '1' ;
            else
                CEN <= '0';
            end if;
    end process;

 result <= ExtraR(N-1 downto 0) & AC1R;
End  BV;

Library IEEE;
use ieee.std_logic_1164.all;
Entity CU_Multiplier is
Port(clk, dataready, AC1RB, CEN: IN std_logic ;
    LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, Busy, Done: OUT std_logic
:='0');
End;
Architecture BV of CU_Multiplier is
Type States is (s1, s2, s3, s4, s5);
Signal PS, NS : States;
Begin
reg: Process(Clk)
   Begin
      IF (Clk'EVENT and Clk = '1')  Then
            PS <= NS;
      End IF;
   End Process;

Transitions: Process(PS, dataready, AC1RB, CEN)
   Begin
      CASE PS is
            when  s1  =>
                IF dataready = '1' Then NS <= s2;
                else NS <= s1; End IF;
            when  s2  =>
                IF AC1RB = '1' Then NS <= s3;
                else NS <= s4; End IF;
            when  s3  =>
                NS <= s4;
            when  s4  =>
```

```
                    IF CEN = '1' Then NS <= s5;
                    else NS <= s2; End IF;
            when  s5  =>
                    NS <= s1;
     End CASE;
End Process;

    LD_AC1R  <= '1' when PS= s1 else '0';
    LD_AC2R  <= '1' when PS= s1 else '0';
    ClearExtraR <= '1' when PS= s1 else '0';
    CLRC <= '1' when PS= s1 else '0';
    LD_ExtraR  <= '1' when PS= s3 else '0';
    Shift  <= '1' when PS= s4 else '0';
    INC  <= '1' when PS= s4 else '0';
    Busy <= '1' when PS=s2 or PS=s3 or PS=s4 else '0';
    Done <= '1' when PS=s5 else '0';
End  BV;
```

(ii) Write a test bench for testing the 4-bit Serial Multiplier assuming that the input arguments are read from an input file and that the output will be stored in an output file. Use TEXTIO package for this purpose. Apply the following values for testing the correct operation of a 4-bit Serial Multiplier:

Input A	**Input B**
5	2
2	2
7	0
0	7
15	10
10	15
15	15
15	1
1	15
15	2
15	4
15	8

The output should be stored in the output file using the following format:

Input A	Input B	Result
5	2	5*2=10

The following test bench was developed and used to verify the correct functionality of the behavioral multiplier which produced the output given below:

```
Library IEEE;
```

```
use ieee.std_logic_1164.all;
USE STD.TEXTIO.ALL;

Entity multiplier_test is
End;

Architecture Test of multiplier_test is
	Component multiplier
	Generic (n: Positive := 4);
	Port(clk: IN std_logic; dataready: IN std_logic; A : IN std_logic_vector (n-1 downto 0);
	 B : IN   std_logic_vector (n-1 downto 0);
	busy, done: OUT std_logic; result: OUT       std_logic_vector ( 2*n-1 downto 0));
	End Component;


	Constant N: Positive :=4;
	Constant K: Positive :=30;
	Constant Period: Time := 100 ns;


	TYPE Integers IS ARRAY (NATURAL RANGE <>) of INTEGER;

	Procedure Int2Bin (Int: IN Integer; Bin : OUT std_logic_vector)  IS
		Variable Tmp: Integer;
		Constant size: Natural := Bin'length;
	Begin
		Tmp := Int;
		if (Tmp < 0) Then
			Tmp :=2**size+Tmp;
		End If;
		For I IN 0 To (Bin'Length - 1) Loop
			If ( Tmp MOD 2 = 1) Then
				Bin(I) := '1';
			Else Bin(I) := '0';
			End If;
			Tmp := Tmp / 2;
		End Loop;
	End Int2Bin;

	function Bin2Int(Bin: std_logic_vector) return integer is
		variable SUM: INTEGER:=0;
	begin

		For I IN 0 To (Bin'Length - 1) Loop
			if Bin(I)='1' then
				SUM := SUM + (2**I);
			end if;
		End Loop;
		return SUM;

	end Bin2Int;

	Signal clk, dataready, done, busy: std_logic :='0';
	Signal A, B: std_logic_vector(N-1 Downto 0);
	Signal Result: std_logic_vector(2*N-1 Downto 0);
	Signal First, Second: Integers(0 to K-1);
	Begin
	Process
```

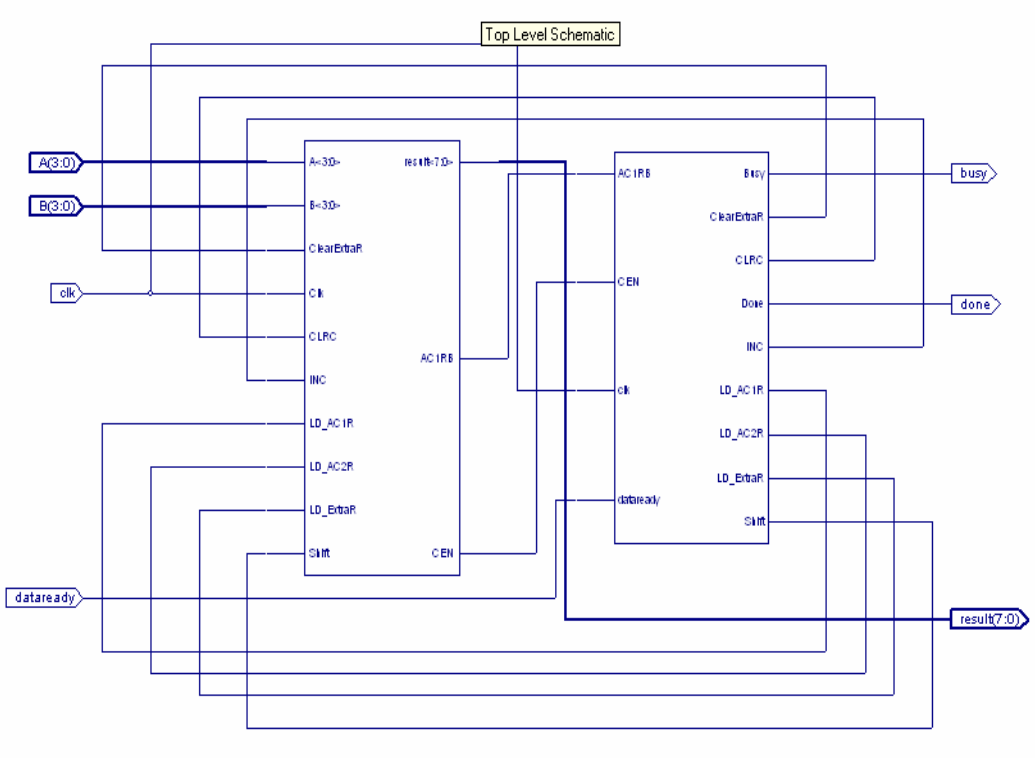
```
File Infile : Text IS IN "mul_input.txt";
Variable My_Line : Line;
Variable val: Integer;



File outFile: Text IS OUT "mul_output.txt";
Variable write_line: Line;
Variable Str: String(1 to 24);
Variable Str2: String(1 to 42);


Variable i,j: integer :=0;
Variable Buf: std_logic_vector(N-1 Downto 0);
Begin
        While Not ( Endfile(Infile) )  Loop

                Readline( Infile, My_Line);  -- read a line from the input file

                Read( My_Line, val);          -- read A value from the line
                First(i)<= val;
                Read( My_Line, val);          -- read B value from the line
                Second(i)<= val;

            i := i + 1;
        End Loop;
        wait for 0 ns;
        j:=0;
        Str :="Input A"&ht&ht&"Input B"&ht&ht&"Result";
            Write(write_line,Str);
            Writeline(outFile, write_line);
            Str2 :="-----------------------------------------";
            Write(write_line,Str2);
            Writeline(outFile, write_line);
        while (j < i) loop
          wait for period;
            if busy='1' Then wait until busy='0'; end if;
            dataready <= '1';
            Int2Bin (First(j), Buf);
            A<=Transport Buf ;
            Int2Bin (Second(j), Buf);
            B<=Transport Buf ;
            wait for 2*period;
            dataready <= '0';
            if done='0' Then wait until done='1';          end if;

            Write(write_line, First(j));
            Write(write_line,ht);
            Write(write_line,ht);
            Write(write_line, Second(j));
            Write(write_line,ht);
            Write(write_line,ht);
            Write(write_line, First(j));
            Write(write_line, '*');
            Write(write_line, Second(j));
            Write(write_line, '=');
            Write(write_line, Bin2Int(result));
            Writeline(outFile, write_line);
```

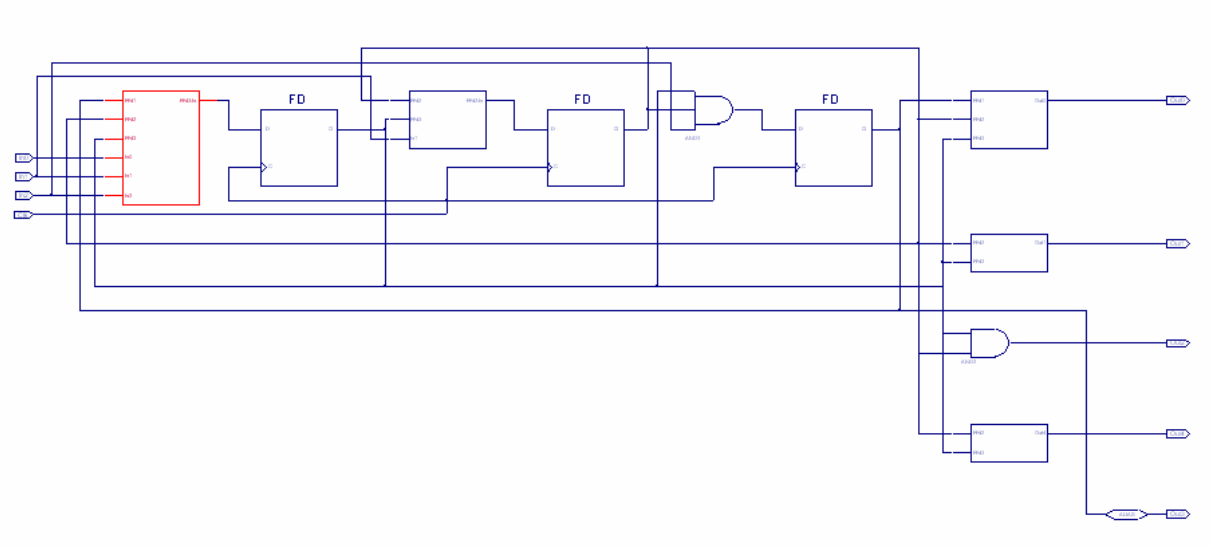
```
                    j := j + 1;
                    wait for 3*period;
                End loop;
                wait;
        End Process;

        clk <= not clk after period;
        CUT: multiplier Generic Map (N) Port Map (clk, dataready, A, B, busy, done, result);

End;
```

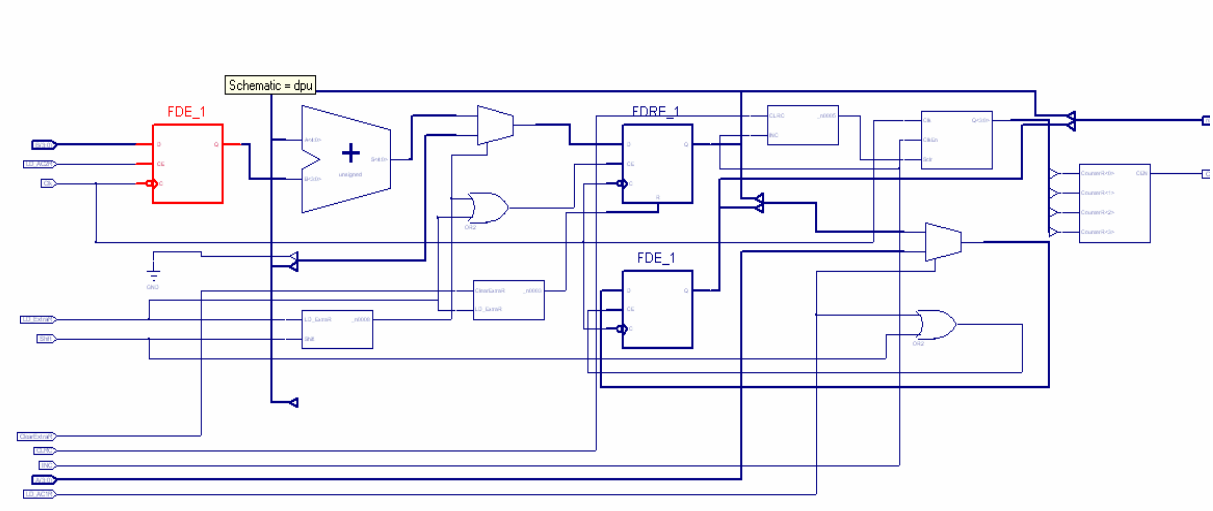
Input A	Input B	Result
5	2	5*2=10
2	2	2*2=4
7	0	7*0=0
0	7	0*7=0
15	10	15*10=150
10	15	10*15=150
15	15	15*15=225
15	1	15*1=15
1	15	1*15=15
15	2	15*2=30
15	4	15*4=60
15	8	15*8=120

(iii) Synthesize the modeled Serial Multiplier in (i) using Xilinx Project Navigator and report on the total equivalent gate count for design after mapping and the longest delay in the design based on Post-Map static timing report.

Both models given in part (i) are synthesizable. The results of the synthesis of the required model are shown below:


Top Level Schematic
A(3:0)
B(3:0)
clk
datareadyy
A<3:0>
B<3:0>
ClearExtraR
Clk
CLRC
INC
LD_AC1R
LD_AC2R
LD_ExtraR
Shift
result<7:0>
AC1R:8
CEN
AC1R:8
Busy
ClearExtraR
CLRC
CEN
Done
INC
clk
LD_AC1R
LD_AC2R
LD_ExtraR
dataready
Shift
busy
done
result(7:0)


FD
FD
FD



Total equivalent gate count for design: 358
Longest delay is 8.165 ns.

(iv) Develop a dataflow model of the n-bit Serial Multiplier. Divide your multiplier into a data-path and control unit and model each one separately using dataflow modeling style.

```
Library IEEE;
use ieee.std_logic_1164.all;
Entity multiplier is
    Generic (n: Positive := 8);
    Port(clk: IN std_logic; dataready: IN std_logic; A : IN          std_logic_vector (n-1 downto 0); B
: IN          std_logic_vector (n-1 downto 0);
busy, done: OUT std_logic; result: OUT          std_logic_vector ( 2*n-1 downto 0));
End multiplier;
Architecture DF of multiplier is
Component DPath_Multiplier
Generic(n: Positive:= 8);
Port(LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, Clk: IN std_logic;
    CEN, AC1RB: OUT std_logic;
    A, B: in std_logic_vector(N-1 DownTo 0);
    result: out std_logic_vector(2*n-1 DownTo 0));
End Component;
Component CU_Multiplier
Port(clk, dataready, AC1RB, CEN: IN std_logic ;
    LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, Busy, Done: OUT
std_logic);
End Component;
Signal LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, CEN, AC1RB:
std_logic;
Begin
dpu: DPath_Multiplier Generic Map(N) PORT MAP (LD_AC1R, LD_AC2R, ClearExtraR,
LD_ExtraR, Shift, INC, CLRC, Clk, CEN, AC1RB, A, B, result);
cu: CU_Multiplier  PORT MAP (clk, dataready, AC1RB, CEN, LD_AC1R, LD_AC2R, ClearExtraR,
LD_ExtraR, Shift, INC, CLRC, Busy, Done);
End;

Library IEEE;
```

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

Entity DPath_Multiplier is
Generic(N: Positive:= 8);
Port(LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, Clk: IN std_logic;
    CEN, AC1RB: OUT std_logic;
    A, B: in std_logic_vector(N-1 DownTo 0);
    result: out std_logic_vector(2*N-1 DownTo 0));
End;
Architecture DF of DPath_Multiplier is
Constant LN: Positive:=N;
Signal AC1R, AC2R : std_logic_vector(N-1 DownTo 0);
Signal ExtraR, t1       : std_logic_vector(N DownTo 0);
Signal CounterR, t2: std_logic_vector(LN-1 DownTo 0);
Signal AC1RE,AC2RE,ExtraRE, CounterRE : Boolean:=False ;

Begin
AC1RE <= LD_AC1R='1' or Shift='1' ;
AC2RE <= LD_AC2R='1';
ExtraRE <= LD_ExtraR='1' or ClearExtraR='1' or Shift='1' ;
CounterRE <= INC='1' or CLRC='1';

edge: Block(Clk='0' and not Clk'Stable)
Begin
AC1Reg: Block(AC1RE and Guard)
  Begin
    AC1R <= Guarded A when LD_AC1R='1' Else ExtraR(0) & AC1R(N-1 Downto 1);
    AC1RB <= AC1R(0);
 end Block AC1Reg;
AC2Reg: Block(AC2RE and Guard)
    Begin
            AC2R <= Guarded B;
    end Block AC2Reg;
ExtraReg: Block(ExtraRE and Guard)
 Begin
   t1 <= '0' & ExtraR(N Downto 1);
   ExtraR <= Guarded ExtraR+ ('0'&AC2R) when LD_ExtraR='1' Else (N Downto 0 => '0')when
ClearExtraR='1' Else t1;
 end Block ExtraReg;
CounterReg: Block(CounterRE and Guard)
    Begin
            t2 <= (LN-2 Downto 0 =>'0')&'1';
            CounterR <= Guarded CounterR + t2 when INC='1' else (LN-1 Downto 0 => '0');
            CEN <= '1' when CounterR=N else '0';
    end Block CounterReg;

End Block edge;
 result <= ExtraR(N-1 downto 0) & AC1R;
End  DF;


Library IEEE;
use ieee.std_logic_1164.all;
Entity CU_Multiplier is
Port(clk, dataready, AC1RB, CEN: IN std_logic ;
    LD_AC1R, LD_AC2R, ClearExtraR, LD_ExtraR, Shift, INC, CLRC, Busy, Done: OUT std_logic
:='0');
```

```
End;
Architecture DF of CU_Multiplier is
Type States is (s1, s2, s3, s4, s5);
Type State_Vector is Array (Natural Range <>) of States;
Function RF(V:State_Vector) Return States is
Begin
    Return V(V'Left);
end RF;
Signal PS:  RF States Register := s1;
Begin
edge: Block(Clk='1'  and  not Clk'Stable)
   Begin
S1B: Block(PS= s1 and Guard)
    Begin
            PS <= Guarded s2 when dataready='1' Else  s1;
    end Block S1B;
S2B: Block(PS= s2 and Guard)
    Begin
            PS <= Guarded s3 when AC1RB='1' Else  s4;
    end Block S2B;
S3B: Block(PS= s3 and Guard)
    Begin
            PS <= Guarded s4;
    end Block S3B;
S4B: Block(PS= s4 and Guard)
    Begin
            PS <= Guarded s5 when CEN='1' Else  s2;
    end Block S4B;
S5B: Block(PS= s5 and Guard)
    Begin
       PS <= Guarded s1;
    end Block S5b;
End Block edge;
    LD_AC1R  <= '1' when PS= s1 else '0';
    LD_AC2R  <= '1' when PS= s1 else '0';
    ClearExtraR <= '1' when PS= s1 else '0';
    CLRC <= '1' when PS= s1 else '0';
    LD_ExtraR  <= '1' when PS= s3 else '0';
    Shift  <= '1' when PS= s4 else '0';
    INC  <= '1' when PS= s4 else '0';
    Busy <= '1' when PS=s2 or PS=s3 or PS=s4 else '0';
    Done <= '1' when PS=s5 else '0';
End  DF;
```

(v) Verify the correct functionality of your dataflow model using the test bench developed in (ii).

Using the same test bench developed in (ii) the correct functionality of the dataflow multiplier is verified and the following output is obtained:

Input A	Input B	Result
5	2	5*2=10
2	2	2*2=4
7	0	7*0=0

0	7	0*7=0
15	10	15*10=150
10	15	10*15=150
15	15	15*15=225
15	1	15*1=15
1	15	1*15=15
15	2	15*2=30
15	4	15*4=60
15	8	15*8=120