COE-561: Digital System Design and Synthesis
Assignment 1 Solution

Q1)

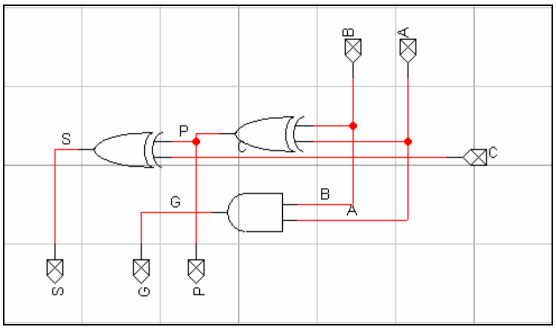
The basic building block of the CLA adder is the partial full adder (PFA). The PFA and the carry path circuitry can be viewed as working in parallel to generate the sum and the carry respectively.

<u>Partial Full Adder:</u>

This block takes three bits as input (A , B , C) and generates three outputs:

$G_i = A_i \cdot B_i$ Called the *generate* function
$P_i = A_i \oplus B_i$ called the *propagate* function
$S_i = A_i \oplus B_i \oplus C_i$ called the *sum* function

This is shown below. The output of the first two functions, G*i* and P*i*, will be used later to generate the carry at each stage of the carry look ahead adder (CLA).



In order to construct a 4-bit CLA, four PFAs are needed to generate the signals that will be used in the functions below.

$S_0 = A0 \oplus B0 \oplus Cin$
$S_1 = A1 \oplus B1 \oplus C1$
$S_2 = A2 \oplus B2 \oplus C2$
$S_3 = A3 \oplus B3 \oplus C3$

$C_1 = G_0 + C_{in}\ P_0$
$C_2 = G_1 + G_0\ P_1 + C_{in}\ P_1\ P_0$
$C_3 = G_2 + G_1\ P_2 + G_0\ P_1\ P_2 + P_2\ P_1\ P_0\ C_{in}$
$C_{out} = G_3 + G_2\ P_3 + G_1\ P_3\ P_2 + G_0\ P_3\ P_2\ P_1 + P_3\ P_2\ P_1\ P_0\ C_{in}$

Overflow occurs when the number of bits is insufficient to accommodate the sum. In our case, this occurs when the sum requires more than 4 bits.
The overflow is detected by the signal OV which is described by the following Boolean function:

$OV = C3 \oplus Cout$

COE-561: Digital System Design and Synthesis
Assignment 1 Solution

(i)

The code that describes this 4-bit CLA *entity* is shown below:

```
Entity CLADDER4 is

     port (A,B: in bit_vector(3 downto 0);
          Cin: in bit;
          Sum: out bit_vector(3 downto 0);
          Cout, OV: out bit);
end;
```

(ii)

The code for the **concurrent** architecture of the CLA entity is shown below:

```
Architecture concurrent of CLADDER4 is

      Signal C1, C2, C3, C4: bit;
      Signal P, G: bit_vector(3 downto 0);

begin
      Sum(0) <= A(0) xor B(0) xor Cin;
      Sum(1) <= A(1) xor B(1) xor C1;
      Sum(2) <= A(2) xor B(2) xor C2;
      Sum(3) <= A(3) xor B(3) xor C3;

      P(0) <= A(0) xor B(0);
      P(1) <= A(1) xor B(1);
      P(2) <= A(2) xor B(2);
      P(3) <= A(3) xor B(3);
      G(0) <= A(0) and B(0);
      G(1) <= A(1) and B(1);
      G(2) <= A(2) and B(2);
      G(3) <= A(3) and B(3);

      C1 <= G(0) or (Cin and P(0));
      C2 <= G(1) or (G(0) and P(1)) or (Cin and P(1) and P(0));
      C3 <= G(2) or (G(1) and P(2)) or (G(0) and P(1) and P(2))
            or (P(2) and P(1) and P(0) and Cin);
      C4 <= G(3) or (G(2) and P(3)) or(G(1) and P(3) and P(2))
            or (G(0) and P(3) and P(2) and P(1)) or (P(3) and P(2) and
            P(1) and P(0) and Cin);
      OV <= C3 xor C4;
      Cout <= C4;
end concurrent;
```

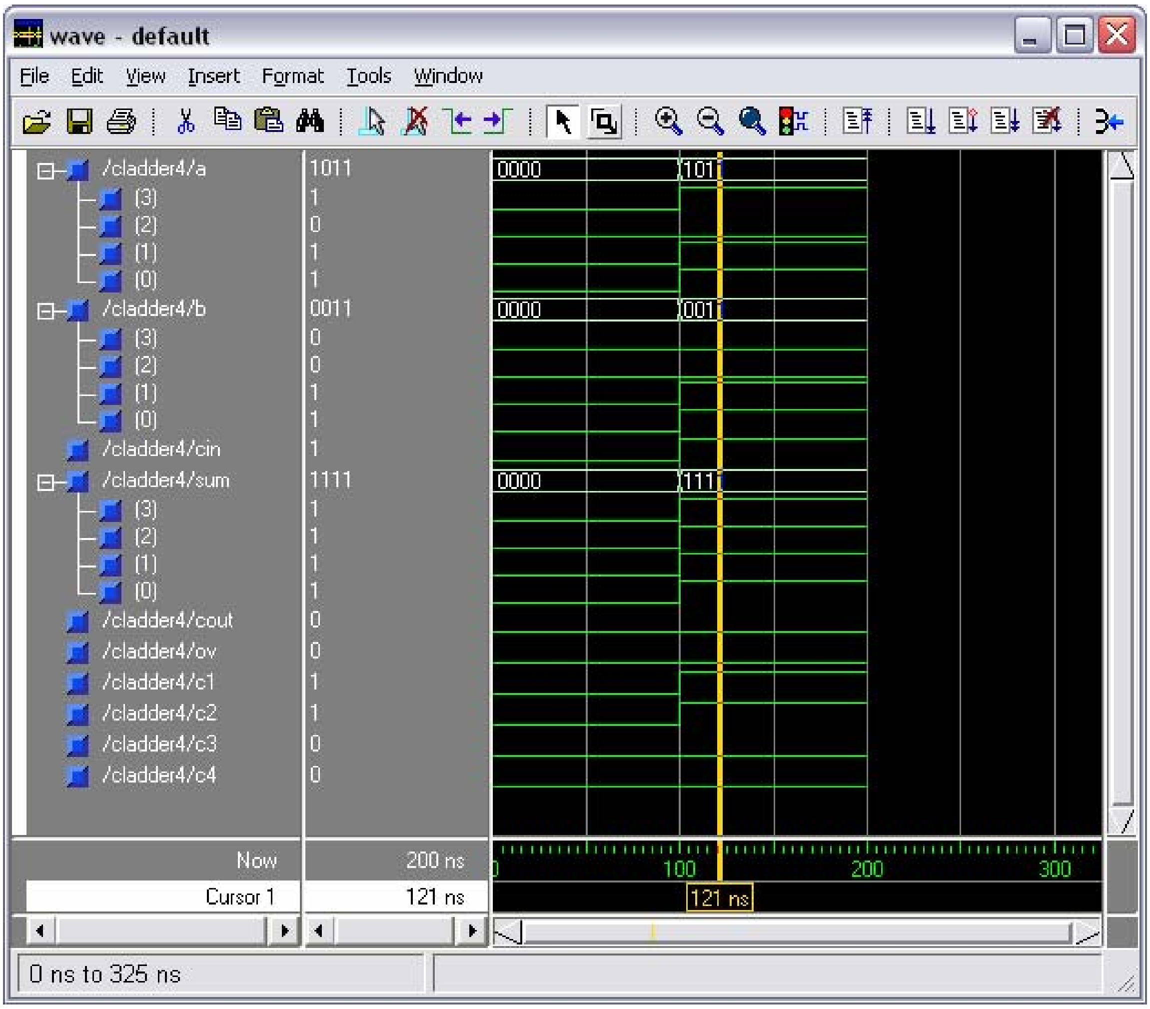
COE-561: Digital System Design and Synthesis
Assignment 1 Solution

Simulation results:

Normal Operation:

Addition of the two numbers -5 and 3 with the Cin set to 1 results in -1 ; A = 1011 , B = 0011 and Cin = 1 results in the following signals which are shown in the simulation snapshot below:

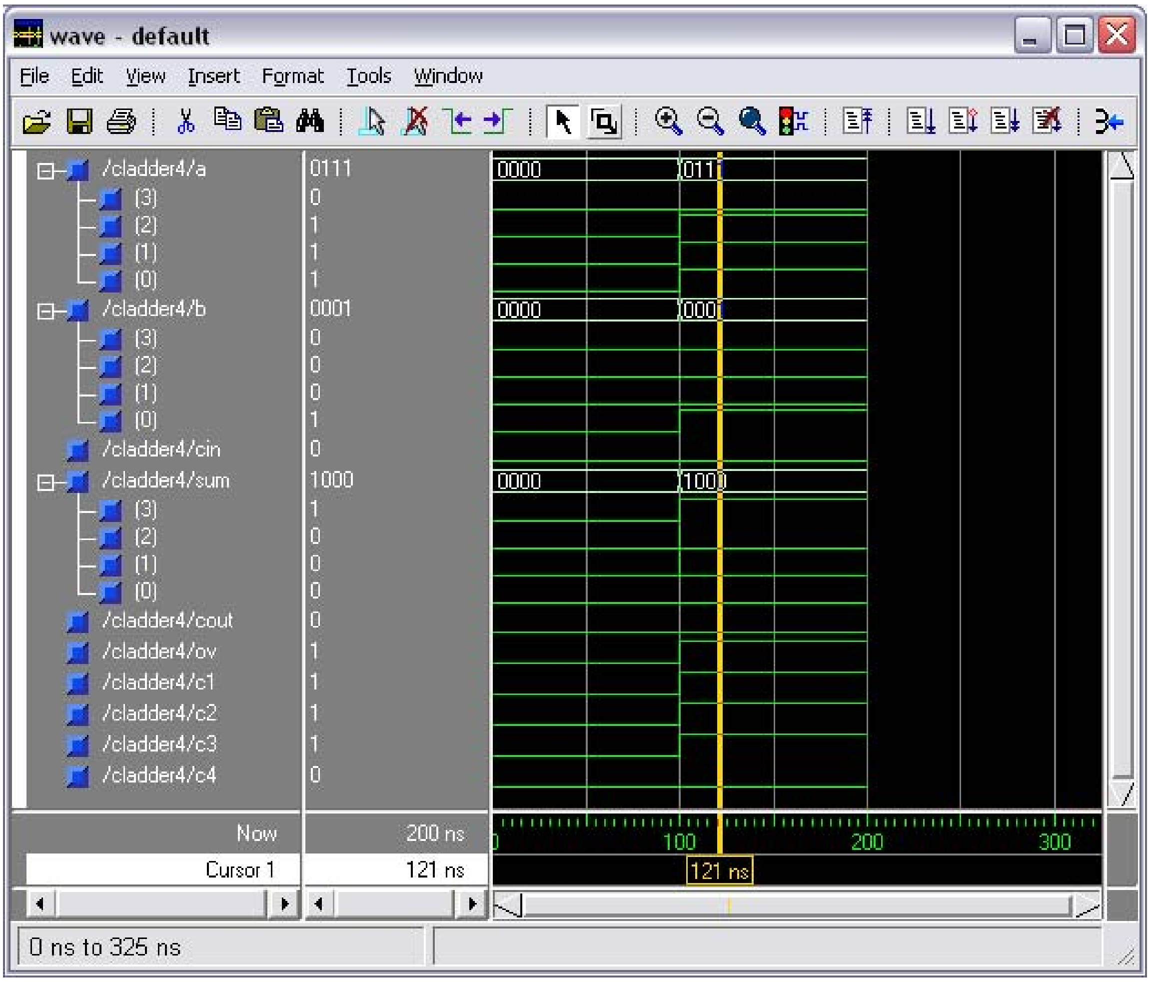
$C_1 = 1$, $C_2 = 1$, $C_3 = 0$, $C_{out} = 0$
$S_0 = 1$, $S_1 = 1$, $S_2 = 1$, $S_3 = 1$



COE-561: Digital System Design and Synthesis
Assignment 1 Solution

Overflow bit:

Addition of the numbers A = 0111 and B = 0001 results in an overflow because the result cannot be represented as a 4-bit two's complement number. The simulation snapshot is shown below:



(iii)

The code for an n-bit CLA is shown below:

```
entity CLADDER is

     Generic(n :positive :=8);
     port(A, B: in bit_vector(n-1 downto 0);
     Cin: in bit;
     Sum : out bit_vector(n-1 downto 0);
     Cout, OV: out bit);

end entity CLADDER;
```

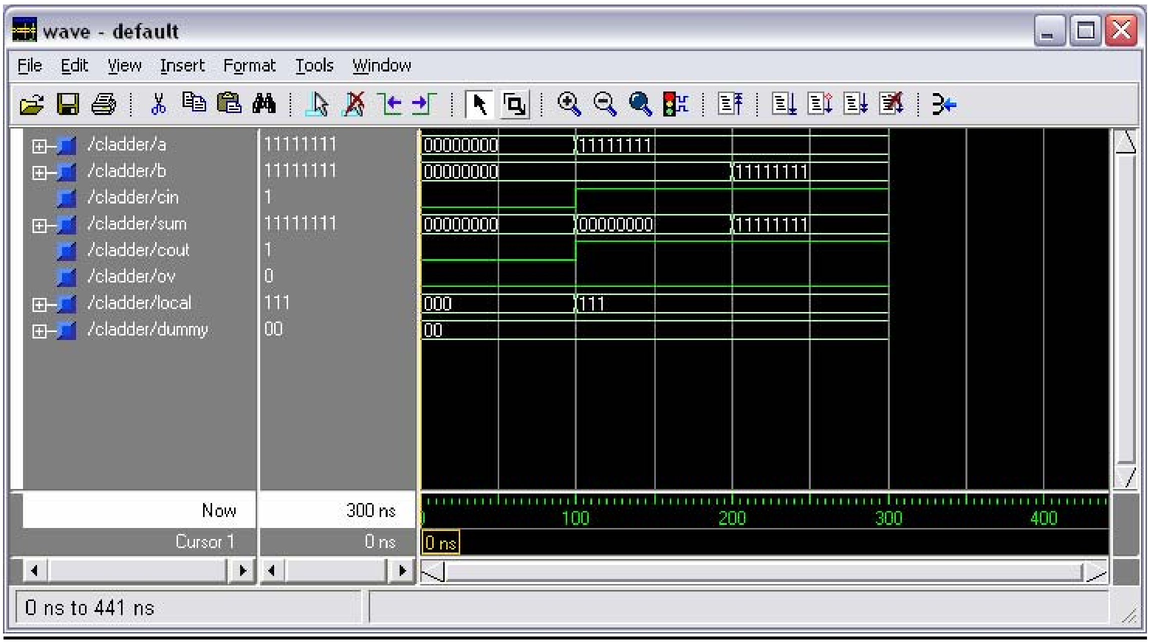
(iv)

Following is the code for the <u>Structural</u> architecture of the n-bit CLA:

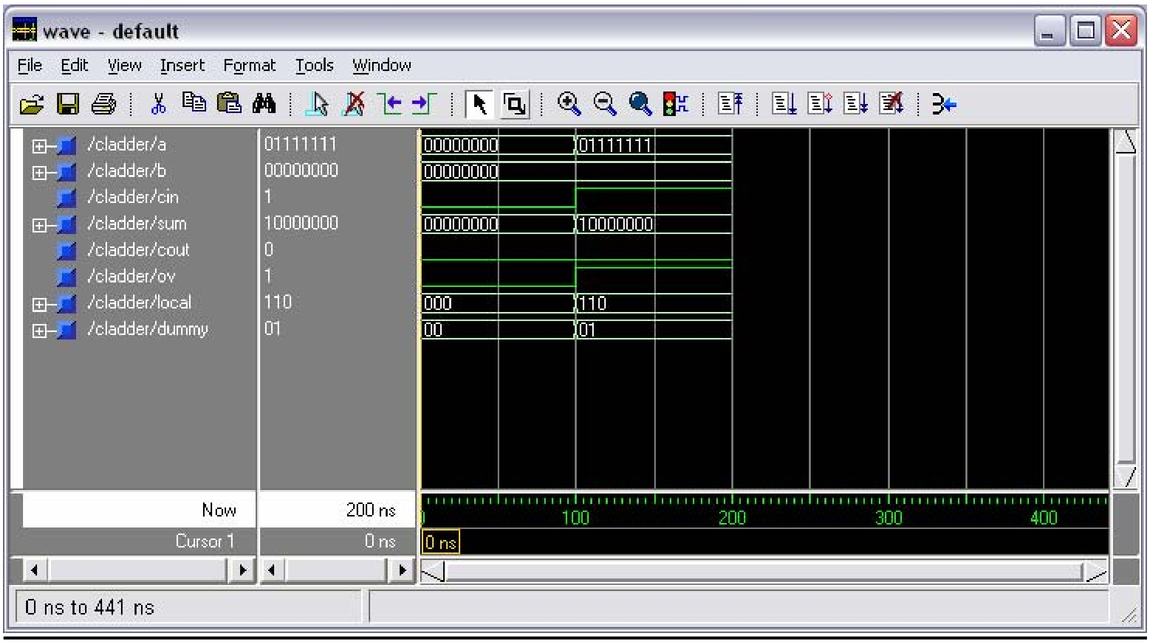
```
Architecture Structural of CLADDER is

     component CLADDER4
          port(A, B: in bit_vector(3 downto 0);
               Cin: in bit;
               Sum: out bit_vector(3 downto 0);
               Cout, OV: out bit);

     End Component;
     Signal local: bit_vector(0 to n/4);
     Signal dummy: bit_vector(1 to n/4);

begin

     local(0) <= Cin;
     g1: for i in 1 to n/4 generate
          g2: CLADDER4 port map(A((i*4)-1 downto (i-1)*4),
          B((i*4)-1 downto (i-1)*4), local(i-1), sum((i*4)-1
          downto (i-1)*4),local(i),dummy(i));

     end generate;
     ov<= dummy(n/4);
     Cout <= local(n/4);

End Structural;
```

COE-561: Digital System Design and Synthesis
Assignment 1 Solution

Simulation snapshot:



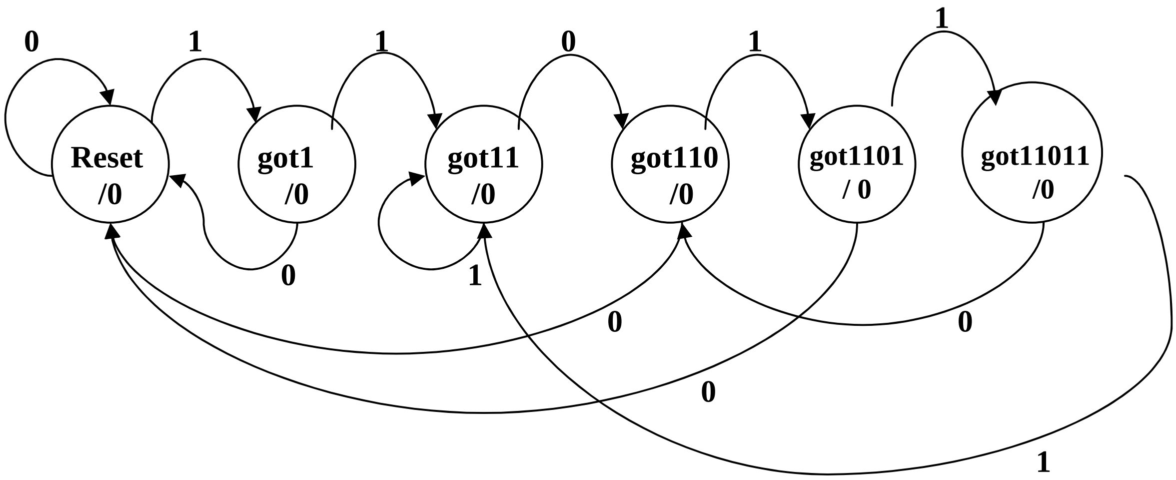
Overflow case:



COE-561: Digital System Design and Synthesis
Assignment 1 Solution

Q2)i)

The design of the finite state machine that detects the sequence 11011 assuming overlapping sequence detection is shown below:



(ii)

the code for the above sequence detector assuming a rising-edge triggered system is shown below:

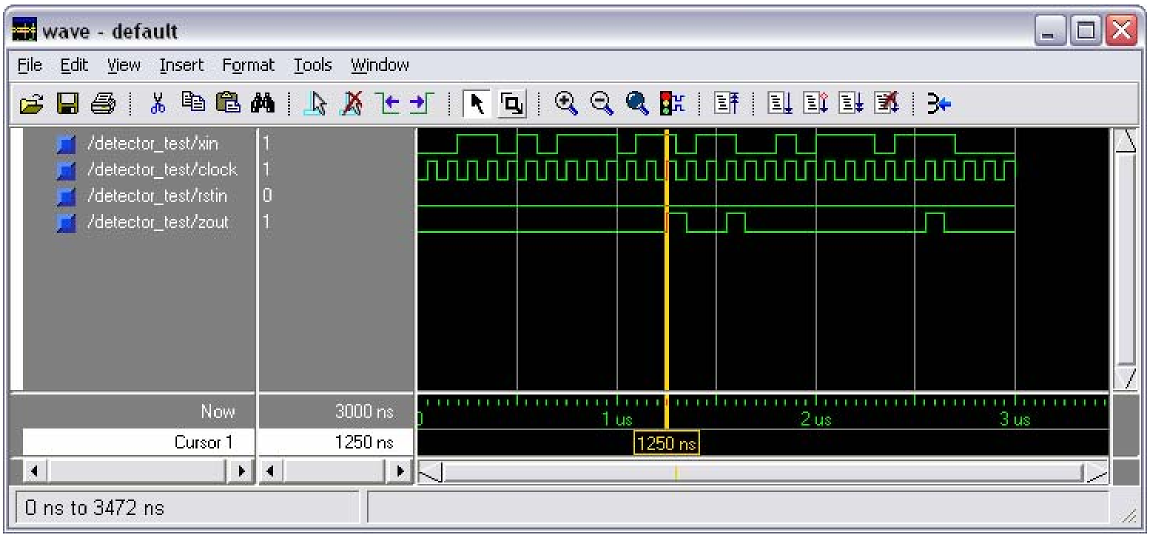
```
Entity detector_11011 is
     Port(x, clk, rst: in bit;
          z: out bit);
End detector_11011;

Architecture behave of detector_11011 is
     Type state is(reset, got1, got11, got110, got1101,
                   got11011);
     signal current: state :=reset;
begin
     Process(clk, rst)
     begin
          if(rst = '1') then
               current <= reset;
          end if;
```

```
if(clk = '1' and clk'event and rst ='0') then
          case current is
               WHEN reset =>
               if(x='1') then current <= got1;
               else
               current <= reset;
               end if;

               WHEN got1 =>
               if(x='1') then current <= got11;
               else
               current <= reset;
               end if;

               WHEN got11 =>
               if(x ='0') then current <= got110;
               end if;

               WHEN got110 =>
               if(x='1') then current <= got1101;
               else
               current <= reset;
               end if;

               WHEN got1101 =>
               if(x='1') then current <= got11011;
               else
               current <= reset;
               end if;

               WHEN got11011 =>
               if(x='1') then current <= got11;
               else
               current <= got110;
               end if;

          end case;
          end if;

     end process;

     z<='1' when current = got11011 else '0';

end behave;
```

(iii)

The code for the test bench and a snapshot of the simulation output are shown below:

```
Entity detector_test is
End detector_test;



----------------------



Architecture test of detector_test is

     Component detector_11011 is
          port(x, clk, rst: in bit;
          z: out bit);
     End Component;

     signal xin, clock, rstin, zout: bit;

begin
     a1: detector_11011 port map(xin,clock,rstin,zout);
     clock <= not clock after 50 ns;

     xin<='0',
     '1' after 200 ns, -- x will remain at 0 for 2 clock
                       --cycles (200 ns) then it gets '1'
     '0' after 400 ns,
     '1' after 500 ns,
     '0' after 600 ns,
     '1' after 700 ns,
     '0' after 1000 ns,
     '1' after 1100 ns,
     '0' after 1300 ns,
     '1' after 1400 ns,
     '0' after 1600 ns,
     '1' after 1800 ns,
     '0' after 1900 ns,
     '1' after 2000 ns,
     '0' after 2300 ns,
     '1' after 2400 ns,
     '0' after 2700 ns;


end test;
```


wave - default
File Edit View Insert Format Tools Window
/detector_test/xin
/detector_test/clock
/detector_test/rstin
/detector_test/zout
1
1
0
1
Now
3000 ns
Cursor 1
1250 ns
1 us
2 us
3 us
1250 ns
0 ns to 3472 ns