Daniel Yeum

Assignment 2

1a. Generate the truth table for the given inputs and outputs.

Truth Table

A	B	C	D	o13	o12	o11	o10	o9	o8	o7	o6	o5	o4	o3	o2	o1	o0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

1b. Drawing the K-Maps for the obtained truth tables.

o13

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	X
10	0	X	0	0

o13 = ABC

o12

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	X
10	0	X	0	0

o12 = AC'D

Daniel Yeum

o11

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	0	0	X
10	0	X	0	0

o11 = ABD’

o10

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	X
10	0	X	1	0

o10 = AB’D

o9

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	X
10	0	X	0	1

o8 = AB’C’

o8

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	X
10	1	X	0	0

o9 = ACD’

o7

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	0	0	X
10	0	X	0	0

o7 = A’BCD

o6

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	0	0	0	X
10	0	X	0	0

o6 = BCD’

o5

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	0	0	X
10	0	X	0	0

o5 = A’BC’D

o4

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	0	0	X
10	0	X	0	0

o4 = A’BC’D’

o3 CD

Daniel Yeum

AB	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	0	0	0	X
10	0	X	0	0

o3 = A’B’CD

o2

AB \ CD	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	0	0	0	X
10	0	X	0	0

o2 = A’B’CD’

o1

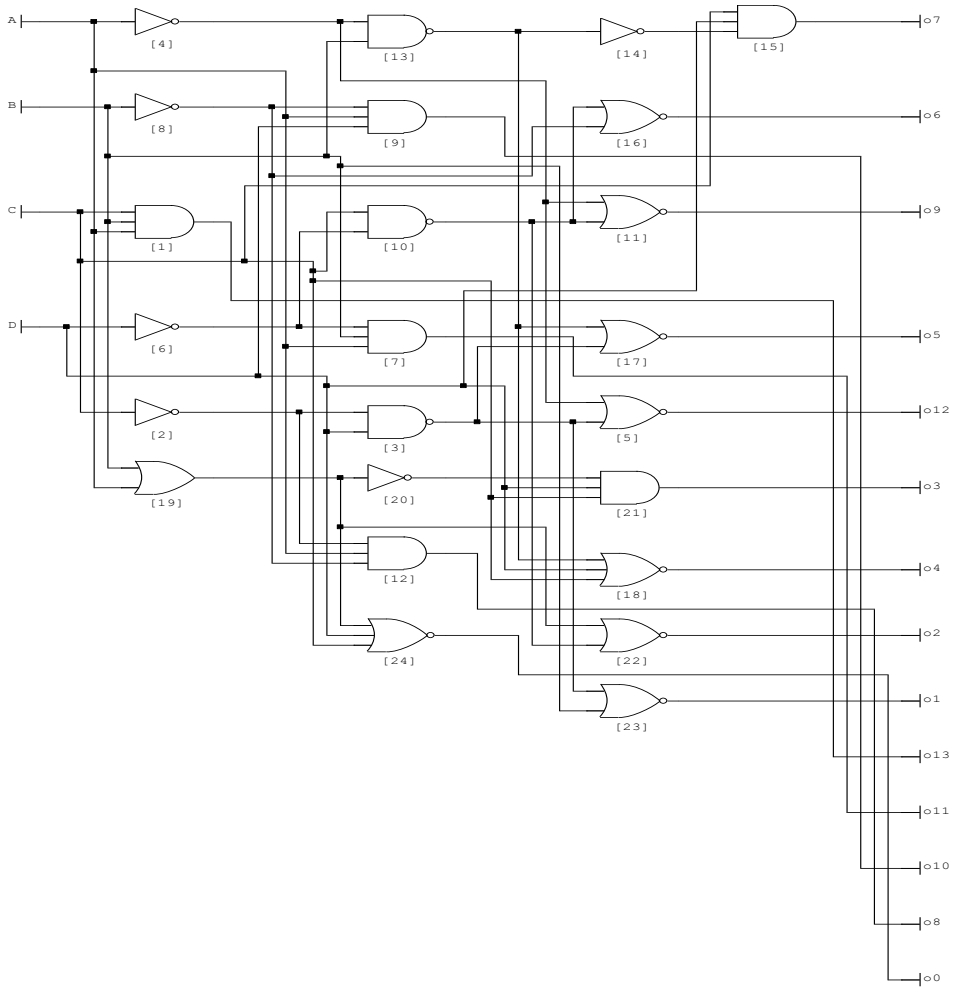
AB \ CD	00	01	11	10
00	0	1	0	0
01	0	0	0	0
11	0	0	0	X
10	0	X	0	0

o1 = B’C’D

o0

AB \ CD	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	0	X
10	0	X	0	0

o0 = A’B’C’D’



1d. Test bench

Daniel Yeum

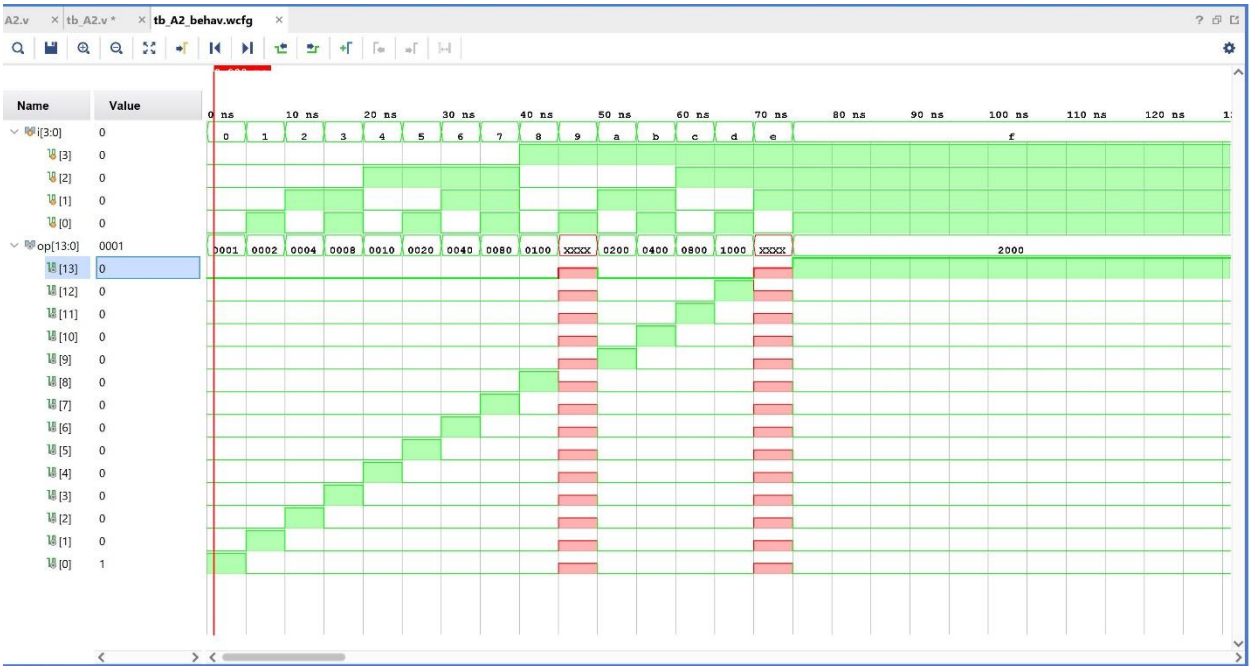
```
module tb_A2();

reg [3:0] i;
wire [13:0] op;

A2 UUT(i, op);

    initial begin
          i = 4'b0000;
       #5 i = 4'b0001;
       #5 i = 4'b0010;
       #5 i = 4'b0011;
       #5 i = 4'b0100;
       #5 i = 4'b0101;
       #5 i = 4'b0110;
       #5 i = 4'b0111;
       #5 i = 4'b1000;
       #5 i = 4'b1001;
       #5 i = 4'b1010;
       #5 i = 4'b1011;
       #5 i = 4'b1100;
       #5 i = 4'b1101;
       #5 i = 4'b1110;
       #5 i = 4'b1111;
    end

endmodule
```

1e. Simulation

```
23  module A2(input [3:0]i, output [13:0] op);
24
25  reg [13:0] op;
26
27      always @(*) begin
28          case(i)
29              4'b0000: op = 14'b00000000000001;
30              4'b0001: op = 14'b00000000000010;
31              4'b0010: op = 14'b00000000000100;
32              4'b0011: op = 14'b00000000001000;
33              4'b0100: op = 14'b00000000010000;
34              4'b0101: op = 14'b00000000100000;
35              4'b0110: op = 14'b00000001000000;
36              4'b0111: op = 14'b00000010000000;
37              4'b1000: op = 14'b00000100000000;
38              4'b1001: op = 14'bXXXXXXXXXXXXXX;
39              4'b1010: op = 14'b00001000000000;
40              4'b1011: op = 14'b00010000000000;
41              4'b1100: op = 14'b00100000000000;
42              4'b1101: op = 14'b01000000000000;
43              4'b1110: op = 14'bXXXXXXXXXXXXXX;
44              4'b1111: op = 14'b10000000000000;
45              default: op = 14'b00000000000000;
46          endcase
47      end
48
49  endmodule
```

1g. Simulate using the test bench.

Daniel Yeum



2. Design an arbiter in Verilog and test using Xilinx simulator.

```
module arbiter(
   req_0,
   req_1,
   req_2,
   clk,
   rst,
   gnt_0,
   gnt_1,
   gnt_2,
   state,
   nstate
   );

   input  req_0;
   input  req_1;
   input  req_2;
```

```
input  clk;
input  rst;
output gnt_0;
output gnt_1;
output gnt_2;
output [1:0] state;
output [1:0] nstate;

reg [1:0] state;    // current state, output is shown at this state
reg [1:0] nstate;   // next state, input is computed at this state
reg gnt_0;         // outputs
reg gnt_1;
reg gnt_2;

wire gnt_0_out;     // To demonstrate the functionality of this circuit
wire gnt_1_out;     // an equation was created from a truth table.
wire gnt_2_out;     // Wires are created for assign statements to be used as net data types
                    // seen in the state machine below.

// Based on the specifications of the design. A truth table was input into
// a program called Logic Friday which can generate a boolean equation
// from the truthtable and minimize the equation. The result is seen below.
// These equations fulfill the described conditions.
// When only req_0 is asserted, gnt_0 is asserted.
// "" req_1 "", gnt_1 "".
// "" req_2 "", gnt_2 "".
// When both req_0 and req_1 are asserted, then gnt_0 is asserted.
// "" req_1 and req_2 "", then gnt_1 "".
// For all other combinations, gnt_2 is asserted.

assign gnt_0_out = ~req_2 & req_0;
```

```
  assign gnt_1_out = req_1 & ~req_0;
  assign gnt_2_out = req_2 & req_0 + ~req_1 & ~req_0;

  // reset and update state
  always @( posedge rst or posedge clk )
   begin
    if ( rst )
     begin
      state  <= 2'b00;      // For the reset, I left state 0 as an empty
      nstate <= 2'b00;      // reset state. This is because if you assign
     end                    // gnt_0 to be associated with state 0, or any output for that matter,
    else                    // then one may run into an issue where gnt_0 is high in instances where it
     state <= nstate;       // is not wanted to be.
   end

  // calculating the next state
  always @( state or req_0 or req_1 or req_2 )
   begin
    case ( state )
     2'b01: if ( gnt_0_out )       // The assign statements created using the
              nstate <= 2'b01;     // net data type outputs are used as the conditionals
            else if ( gnt_1_out )  // in this sequential function.
              nstate <= 2'b10;
            else if ( gnt_2_out )
              nstate <= 2'b11;
            else
              nstate <= 2'b11;

     2'b10: if ( gnt_0_out )
              nstate <= 2'b01;
            else if ( gnt_1_out )
```

```
            nstate <= 2'b10;
          else if ( gnt_2_out )
            nstate <= 2'b11;
          else
            nstate <= 2'b11;

      2'b11: if ( gnt_0_out )
            nstate <= 2'b01;
          else if ( gnt_1_out )
            nstate <= 2'b10;
          else if ( gnt_2_out )
            nstate <= 2'b11;
          else
            nstate <= 2'b11;

       default: nstate <= 2'b11;
     endcase
   end

  // output of the design
  always @( state )          // Because this is a sequential design
    begin                   // the output will not be realized until one
     case ( state )         // clock cycle after the input is realized.
       2'b01:   begin       // Mentioned above gnt_0 is associated
              gnt_0 <= 1;  // with state 1 and so on to allow for a reset state 0...
              gnt_1 <= 0;
              gnt_2 <= 0;
            end
       2'b10:   begin
              gnt_0 <= 0;
              gnt_1 <= 1;
```

```
                gnt_2 <= 0;
              end
        2'b11:   begin
                gnt_0 <= 0;
                gnt_1 <= 0;
                gnt_2 <= 1;
              end
        default: begin
                gnt_0 <= 0;
                gnt_1 <= 0;
                gnt_2 <= 0;
              end
      endcase
    end

endmodule

module tb_arbiter();

   reg  req_0;
   reg  req_1;
   reg  req_2;
   reg  clk;
   reg  rst;
   wire gnt_0;
   wire gnt_1;
   wire gnt_2;
   wire [1:0] state;
   wire [1:0] nstate;
```

```
arbiter UUT
   (
    .req_0  ( req_0  ),
    .req_1  ( req_1  ),
    .req_2  ( req_2  ),
    .clk    ( clk    ),
    .rst    ( rst    ),
    .gnt_0  ( gnt_0  ),
    .gnt_1  ( gnt_1  ),
    .gnt_2  ( gnt_2  ),
    .state  ( state  ),
    .nstate ( nstate )
   );

initial
begin
  rst = 1;
  clk = 0;
    forever #5 clk = ~clk;
end

initial
begin
  #10 rst   = 0;   // state 0
      req_0 = 0;   // next state 0 with no inputs
      req_1 = 0;   // default next state 3 with inputs. next output gnt_2
      req_2 = 0;

  #15 req_0 = 1;   // state 3. output gnt_2
      req_1 = 0;   // next state 1. next output gnt_0
```

```
        req_2 = 0;

    #10 req_0 = 0;   // state 1. output gnt_0
        req_1 = 1;   // next state 2. next output gnt_1
        req_2 = 0;

    #10 req_0 = 0;   // state 2. output gnt_1
        req_1 = 0;   // next state 3. next output gnt_2
        req_2 = 1;

    #10 req_0 = 1;   // state 3. output gnt_2
        req_1 = 1;   // next state 1. next output gnt_0
        req_2 = 0;

    #10 req_0 = 0;   // state 1. output gnt_0
        req_1 = 1;   // next state 2. next output gnt_1
        req_2 = 1;

    #10 req_0 = 1;   // state 2. output gnt_1
        req_1 = 1;   // next state 3. next output gnt_2
        req_2 = 1;

    #10  // buffer   // state 3. output gnt_2
                    // next state 3. next output gnt_2.

    #10 rst   = 1;   // state 0. outputs 0. Reset.
  end

endmodule
```

Daniel Yeum


arbiter.v
tb_arbiter.v
Untitled 12
Name
Value
req_0
req_1
req_2
clk
rst
gnt_0
gnt_1
gnt_2
state[1:0]
nstate[1:0]
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
110.000 ns