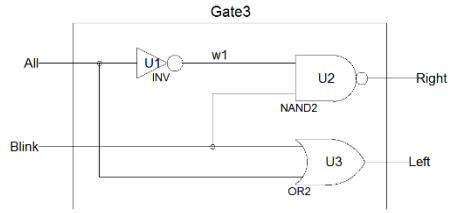
1 Create a relative timing test bench for the following circuit called Gate3.

```
`timescale 1ns / 1ps
//simulation time unit is 1ns and resolution is 1ps
//Author: Jianjian Song
//Date: August 16, 2020
//Purpose: ECE433 example circuit
module Gate3 (
input All, Blink,
output Right, Left);
wire w1;
        not U1 (w1, All);
        nand U2 (Right, w1, Blink);
        or U3 (Left, All, Blink);
endmodule
```

All	Blink	Right	Left
0	0	1	0
0	1	0	1
1	0	1	1
1	1	1	1

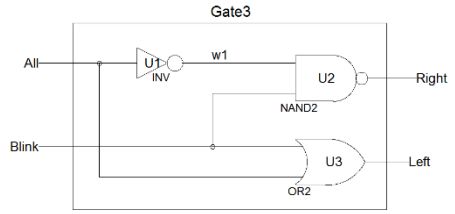


Time	1	2	3	4	5	6	7	8	9	10	11	12	12	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
All																																		
Blink																																		
Right																																		
Left																																		

2 Create an absolute timing test bench for the following circuit called Gate3.

```
`timescale 1ns / 1ps
//simulation time unit is 1ns and resolution is 1ps
//Author: Jianjian Song
//Date: August 16, 2020
//Purpose: ECE433 example circuit
module Gate3 (
input All, Blink,
output Right, Left);
wire w1;
        not U1 (w1, All);
        nand U2 (Right, w1, Blink);
        or U3 (Left, All, Blink);
endmodule
```

All	Blink	Right	Left
0	0	1	0
0	1	0	1
1	0	1	1
1	1	1	1



Time	1	2	3	4	5	6	7	8	9	10	11	12	12	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
All																																		
Blink																																		
Right																																		
Left																																		

3 Gate3 Circuits

3.1 Gate Level

```
module Gate3Gates (
input All, Blink,
output Right, Left);
wire w1;
        not U1 (w1, All);
        nand U2(Right, w1, Blink);
        or U3 (Left, All, Blink);
endmodule
```

3.2 Logic Expressions

```
module Gate3Expression (
input All, Blink,
output reg Right, Left);

always @ (All, Blink) begin
        Right <= !(!All&Blink);
        Left <= All | Blink;
        end
endmodule
```

3.3 Truth Table

```
module Gate3Table (
input All, Blink,
output reg Right, Left);

        always @(All, Blink)
                case({All,Blink})
                0: {Right,Left}=2'b10;
                1: {Right,Left}=2'b01;
                2: {Right,Left}=2'b11;
                3: {Right,Left}=2'b11;
                endcase
endmodule
```

3.4 Assign statement

```
module Gate3Assign (
input All, Blink,
output Right, Left);

        assign Right = !(!All&Blink);
        assign Left = All | Blink;
endmodule
```

4 Gate3 Test Benches

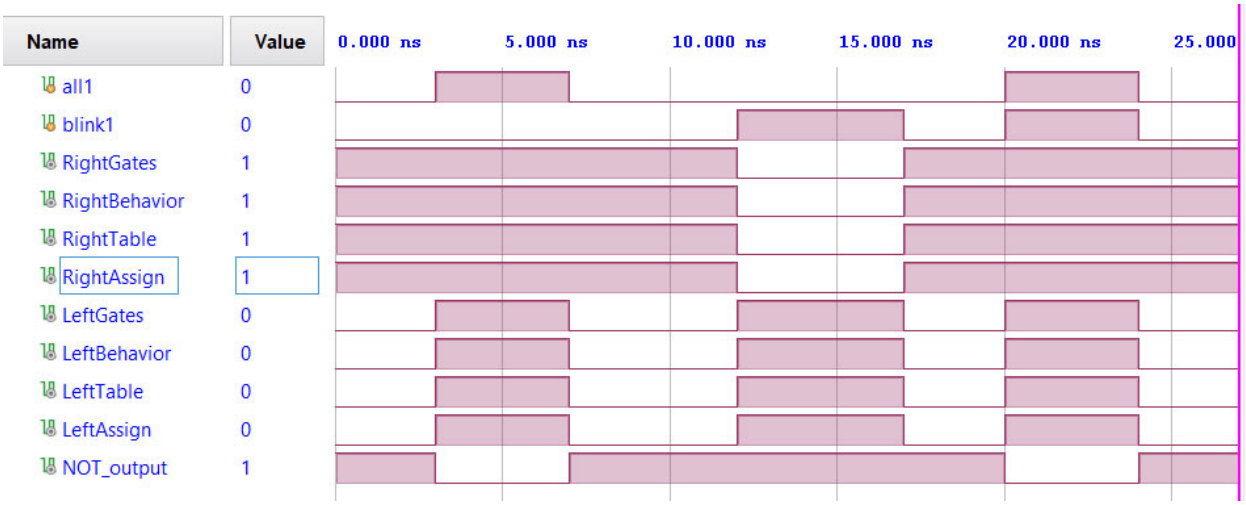
4.1 Relative Time Instants

```
module Gate3TestRelative;
reg all1, blink1;  // Inputs
wire RightGates, LeftGates;          // Outputs
wire RightBehavior, LeftBehavior;
wire RightTable, LeftTable;
wire RightAssign, LeftAssign;
wire NOT_output = Gate3GateChip.w1;     //internal wire of a device under test (DUT)

          initial begin
// Initialize Inputs
          all1 = 0; blink1 = 0;          #3;
          all1 = 1; blink1 = 0;           #4;
          all1 = 0;              blink1 = 0;          #5;
          all1 = 0;              blink1 = 1;           #5;
          all1 = 0;              blink1 = 0;          #3;
          all1 = 1;              blink1 = 1;          #4;
          all1 = 0;              blink1 = 0;          #3;
          $stop;
          end

// Associate the test bench with a device under test (DUT)
Gate3Gates Gate3GateChip ( .Blink(blink1), .Left(LeftGates), .Right(RightGates), .All(all1));
  Gate3Expression Gate3ExpressionChip (all1, blink1, RightBehavior, LeftBehavior);
Gate3Table Gate3TableChip (all1, blink1, RightTable, LeftTable);
Gate3Assign Gate3AssignChip (all1, blink1, RightAssign, LeftAssign);

endmodule
```

4.2 Absolute Time Instants

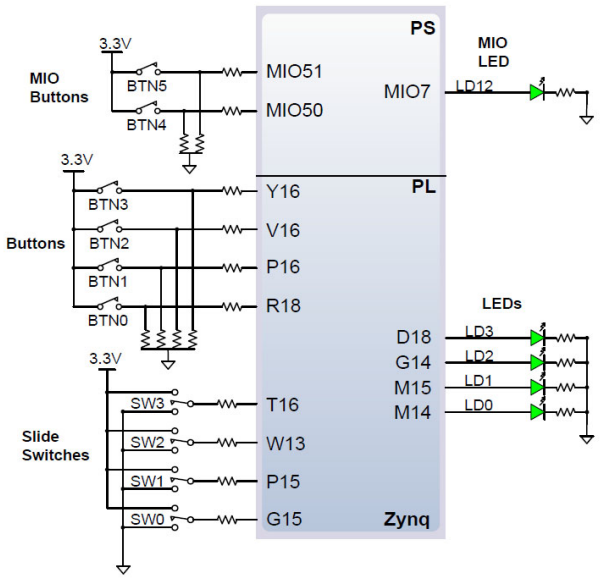
```
module Gate3TestAbsolute;
reg all1, blink1;  // Inputs
wire RightGates, LeftGates;          // Outputs
wire RightBehavior, LeftBehavior;
wire RightTable, LeftTable;
wire RightAssign, LeftAssign;
wire NOT_output = Gate3GateChip.w1;     //internal wire of a device under test (DUT)
initial fork
          #0        all1 = 0; #3        all1 = 1; #7        all1 = 0;
          #20   all1=1;       #25       all1=0;
          #0        blink1 = 0;        #12       blink1 = 1;         #17       blink1 = 0;
          #20       blink1 = 1;        #25       blink1 = 0;
          #28       $stop;    join
// Associate the test bench with a device under test (DUT)
Gate3Gates Gate3GateChip ( .Blink(blink1), .Left(LeftGates), .Right(RightGates), .All(all1));
  Gate3Expression Gate3ExpressionChip (all1, blink1, RightBehavior, LeftBehavior);
Gate3Table Gate3TableChip (all1, blink1, RightTable, LeftTable);
Gate3Assign Gate3AssignChip (all1, blink1, RightAssign, LeftAssign);

endmodule
```

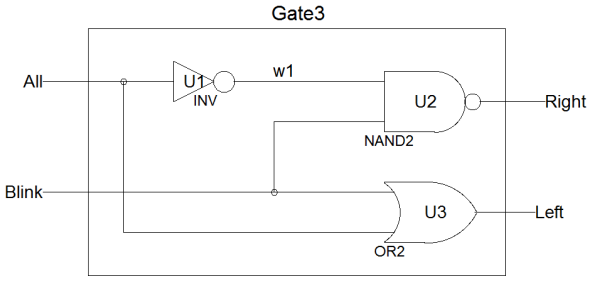
5 Gate3 Simulation Waveforms



6 Gate3 Circuit with Zybo Board: gate3vivado

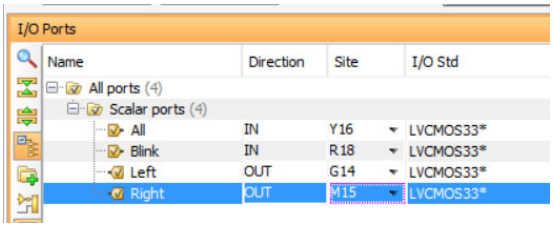
6.1 Zybo peripheral devices





6.2 Pin assignment in Xilinx device constraint file gate3vivado.xdc

```
#//File name: gate3vivado.xdc
set_property PACKAGE_PIN Y16 [get_ports All]
set_property PACKAGE_PIN R18 [get_ports Blink]
set_property PACKAGE_PIN G14 [get_ports Left]
set_property PACKAGE_PIN M15 [get_ports Right]
set_property IOSTANDARD LVCMOS33 [get_ports All]
set_property IOSTANDARD LVCMOS33 [get_ports Blink]
set_property IOSTANDARD LVCMOS33 [get_ports Left]
set_property IOSTANDARD LVCMOS33 [get_ports Right]
```



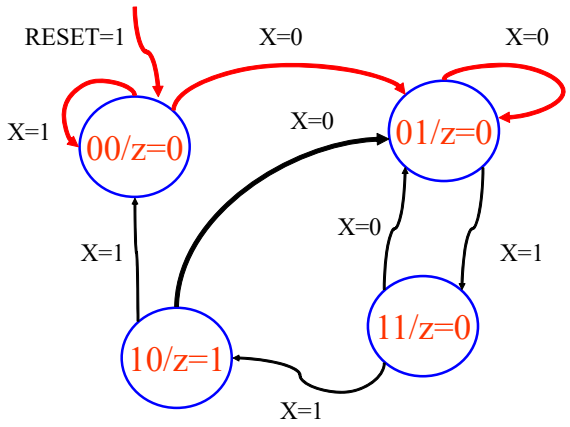
7 Verilog behavior implementation with a state machine.

7.1 Draw a state diagram and check it against the above test sequence

7.2 Create a Verilog Moore state machine for your state diagram

7.3 Create a test bench to test your state machine exhaustively

7.4 Simulate your Verilog circuit to verify its correctness



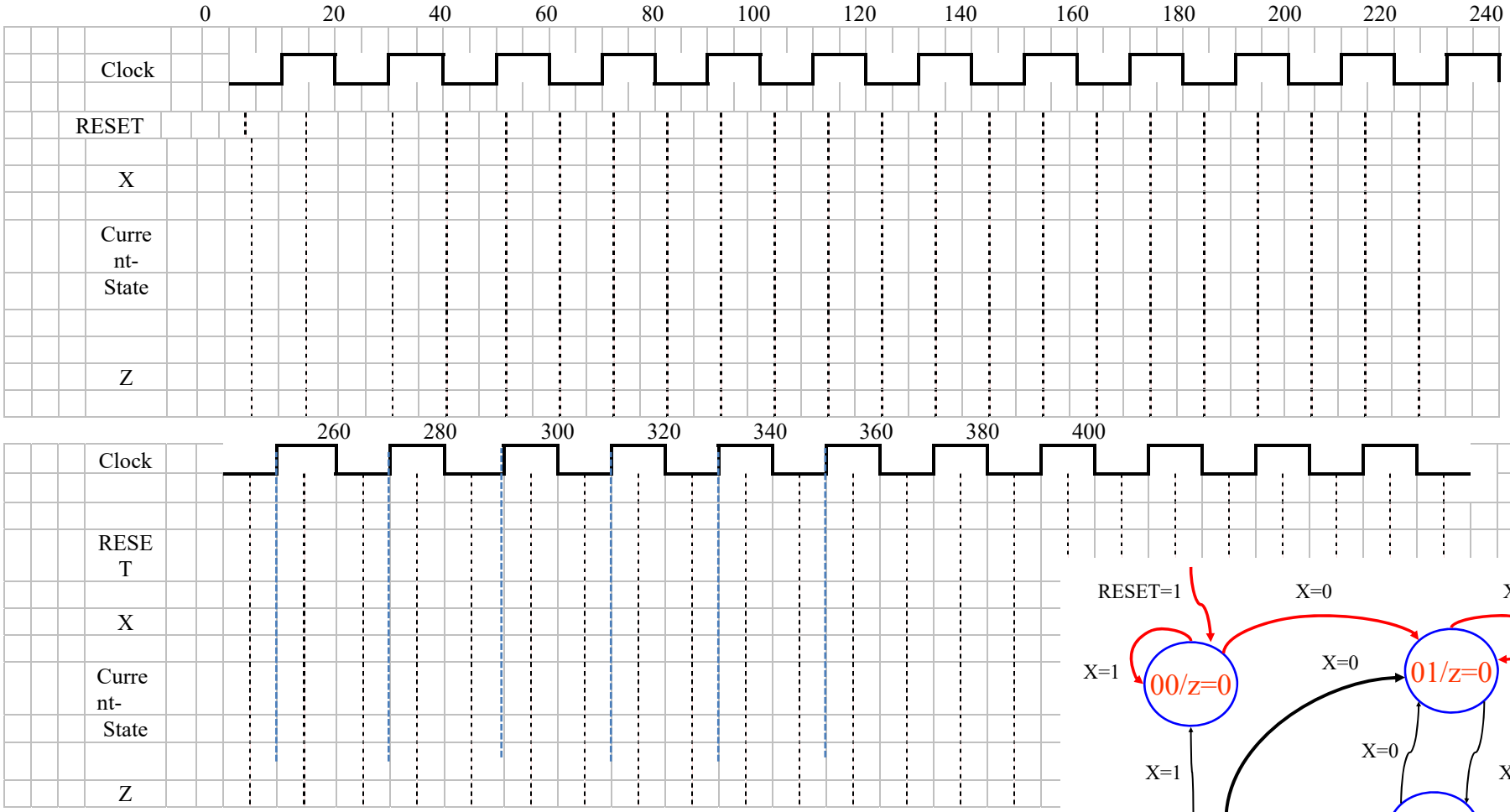
module detect011StateMachineV1(X, Z, reset, CLOCK);

Endmodule

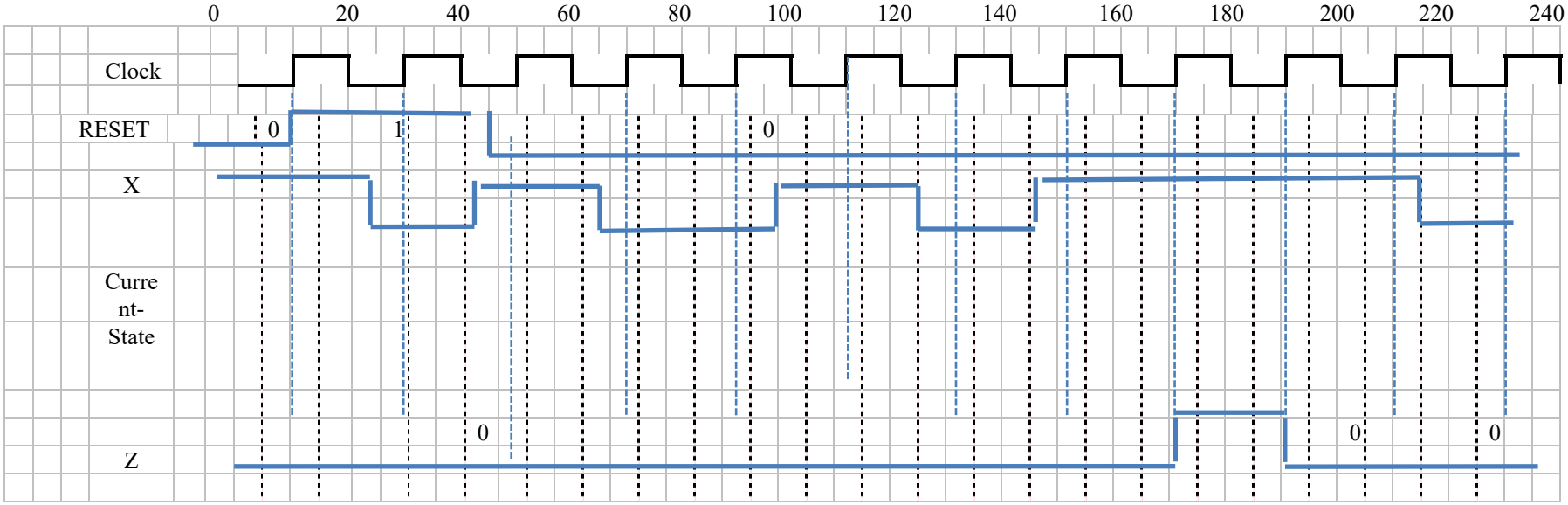
```
`timescale 1ns / 100ps

// File name       : detect011StateMachineV1.v
// Jianjian Song
// Fall 2020 HUST
// If the input X is 011 with 0 being the first bit, the output Z will produce a "1"
//the detection is recursive.

module detect011StateMachineV1(
input    X, RESET, CLOCK,
output reg Z,
// State variables
output reg [1:0] CurrentState);
reg [1:0] NextState;

// State codes
parameter          State1 = 2'b00, State0 = 2'b01, State011 = 2'b10, State01 = 2'b11;

// Output logic
always @ (CurrentState)
          if (CurrentState == State011) Z <= 1;
          else Z <= 0;

// State registers
always @ (posedge CLOCK)
          begin
          if (RESET==1)
                    CurrentState <= State1;
          else
                    CurrentState <= NextState;
          end

//next state logic
always@(CurrentState or X)
          case (CurrentState)
          State1:   begin
                              if (X==0) NextState <= State0;
                              else NextState <= State1;
                    end
          State0:   begin
                              if (X==0) NextState <= State0;
                              else NextState <= State01;
                    end
          State011:           begin
                              if (X==0) NextState <= State0;
                              else NextState <= State1;
                    end
          State01: begin
                              if (X==0) NextState <= State0;
                              else NextState <= State011;
                    end
          default: NextState<=State1;
          endcase

endmodule
```

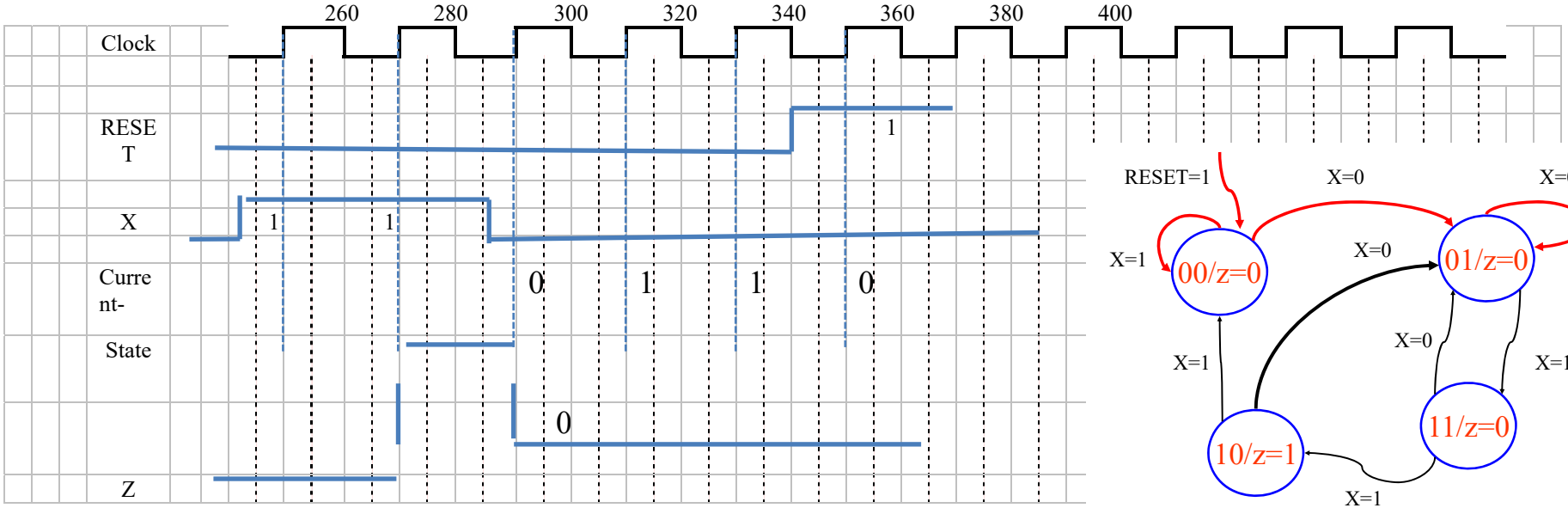
```
`timescale 1ns / 100ps

// File name       : detect011StateMachineV2.v
// Jianjian Song
// Fall 2020 HUST
// If the input X is 011 with 0 being the first bit, the output Z will produce a "1"
// The detection is recursive.

module detect011StateMachineV2(input X, RESET, CLOCK, output reg Z,
// State variables
output reg [1:0] CurrentState);

// State codes
parameter          State1 = 2'b00, State0 = 2'b01, State011 = 2'b10, State01 = 2'b11;

// Output logic
always @ (CurrentState)
          if (CurrentState == State011) Z <= 1;
          else Z <= 0;

// State registers
always @ (posedge CLOCK)
          if (RESET==1) CurrentState <= State1;
          else
            case (CurrentState)
            State1:if (X==0) CurrentState <= State0; else CurrentState <= State1;
            State0:if (X==0) CurrentState <= State0; else CurrentState <= State01;
            State011:  if (X==0) CurrentState <= State0;          else CurrentState <= State1;
            State01: if (X==0) CurrentState <= State0; else CurrentState <= State011;
            default: CurrentState<=State0;
          endcase
endmodule
```

8 Draw a timing diagram in the grid paper to perform a manual timing simulation of the state diagram below.



9 Detect 011 Test Pattern Generation;

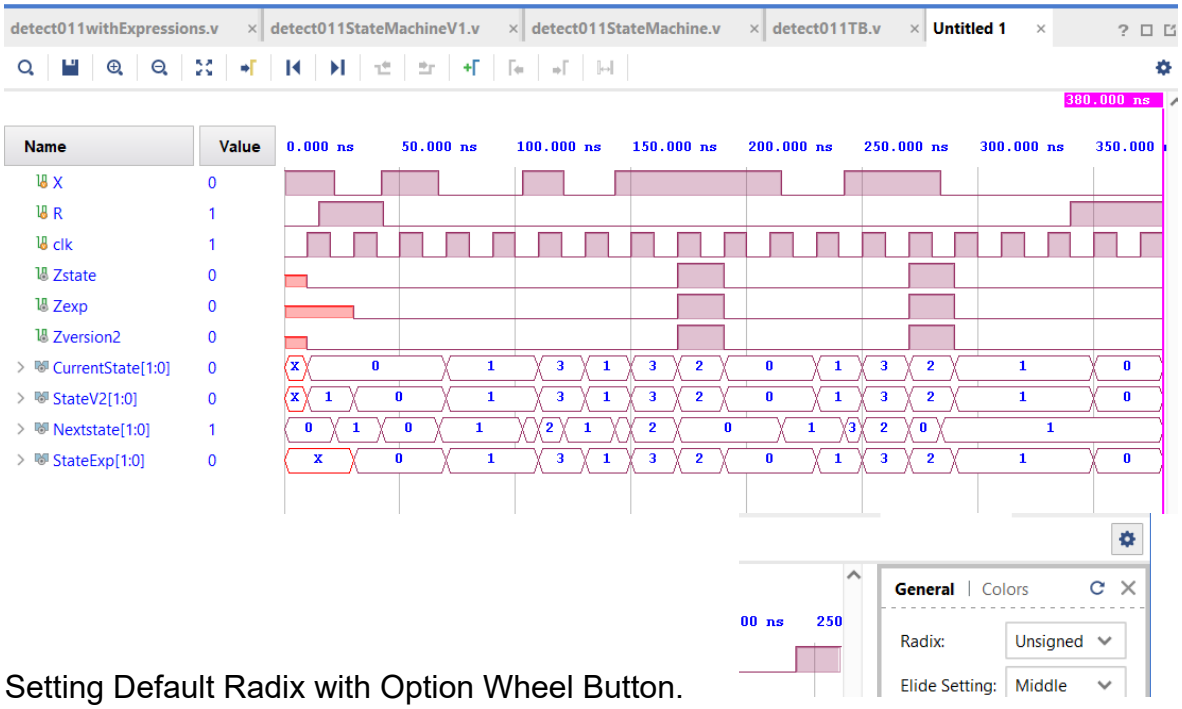




Notice how a periodic clock signal is generated.

```
`timescale 1ns / 100ps

// File name       : detect011TB.v
// Jianjian Song
//Fall 2020 HUST
// If the input X is 011 with 0 being the first bit,
//the output Z will produce a "1". The detection is recursive.

module detect011_test_circuit;
reg X, R, clk;
wire Zstate, Zexp, Zversion2;
wire [1:0] CurrentState,StateV2;
wire [1:0] Nextstate= UnitV1.NextState;
wire [1:0] StateExp= {UnitExpressions.QA,UnitExpressions.QB};

initial begin #0 clk=0; end            //initial value for input clock
always #10 clk=~clk;       //generate a periodic signal as clock

        initial fork          //absolute time instances
                #0  R=0; #15 R = 1;  #43 R=0; #340 R=1;
                #0 X=1;   #22 X=0; #42 X=1;   #67 X=0;
                #103 X=1;  #121 X=0;  #143 X=1;
                #215 X=0; #242 X=1; #284 X=0;
                #380 $stop;
        join

detect011StateMachineV1 UnitV1(X, R, clk, Zstate, CurrentState);
detect011Expressions UnitExpressions(X, R, clk, Zexp);
detect011StateMachineV2 UnitV2(X, R, clk, Zversion2, StateV2);

endmodule
```

9.1 Vivado Simulation Results



Setting Default Radix with Option Wheel Button.