Digital Lab 3:

Experiment 6:

Data Communication Interface --- UART

Date: 2023/12/07

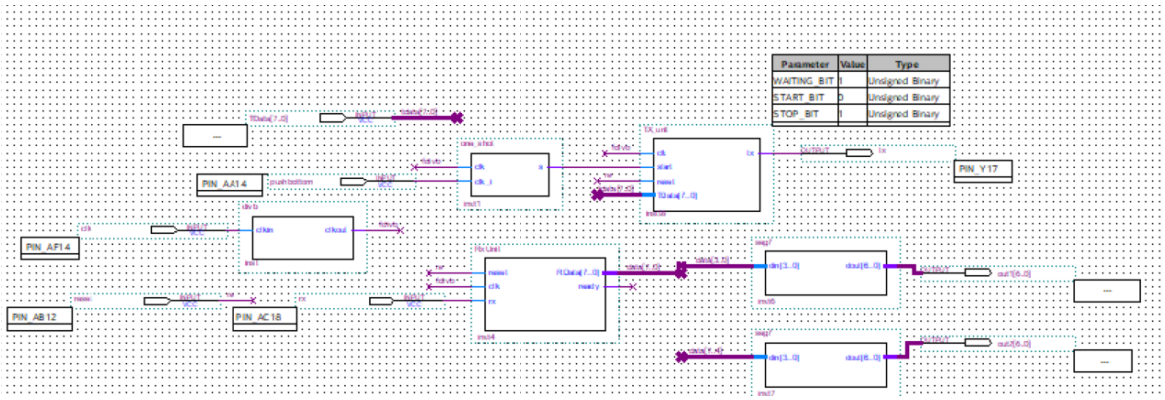
Class: 電機三全英班

Group: Group 11

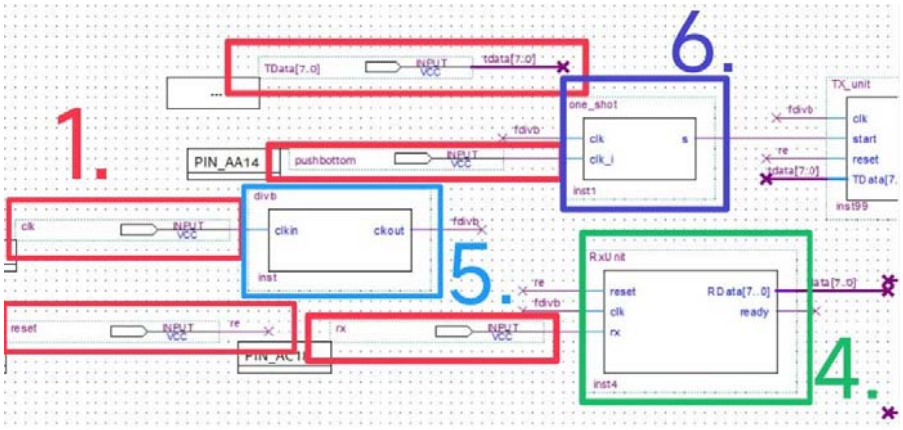
Name: B103105006 胡庭翊

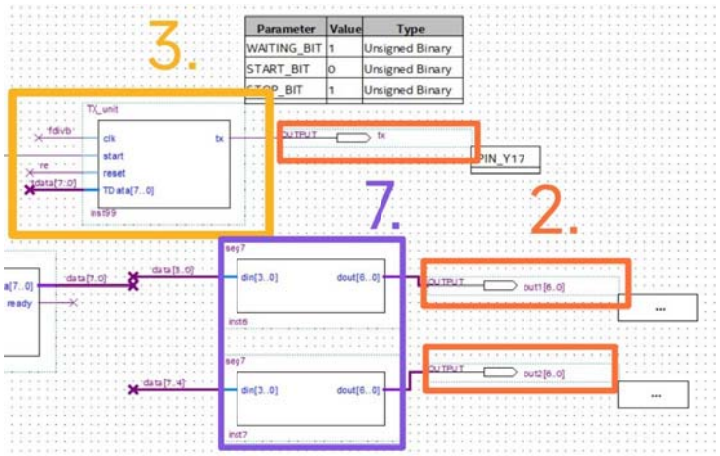
I. Block Diagram

Structure:



Function:



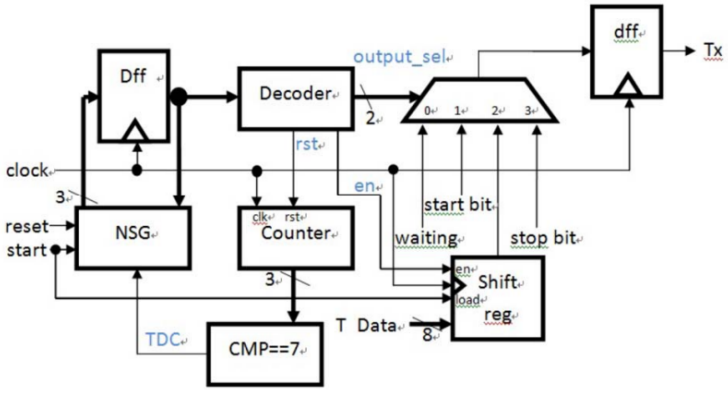


Descriptions:

1. Inputs:
   The inputs are 50MHz clock, 8 DIP switches, 1 button, 1 reset signal, RS232 Rx signal.

2. Outputs:
   The outputs are two sets of seven-segment displays, RS232 Tx signal
3. Transmitter:
   Inputs: Reset signal (reset), 115200Hz clock signal (clock)
   Data to be transmitted, Transmission start signal.
   Output: Tx signal

First, it waits for a transmission signal (triggered by the FPGA's push button). When the push button is pressed, it signifies the start of transmission. At this point, the shift register locks in the ASCII value to be transmitted (corresponding to the value of FPGA's Dip_sw). The state transitions from "waiting to transmit" (State 1) to "transmission start" (State 2). Starting from State 2, the design adheres to the RS232 data transmission format described earlier. It begins by sending the start bit (value 0) (State 2), followed by the sequential transmission of 8 data bits (State 3). After transmitting the 8 data bits, it sends the stop bit (value 1) (State 4). Then, it enters the state of waiting for the next transmission, and during this time, it sends the waiting bit (value 1) (State 1). The output selection is determined based on the current state. When there's no transmission, it outputs 00. When sending the start bit, it outputs 01. During data transmission, it outputs 10. When sending the stop bit, it outputs 11. The counter is a 3-bit counter that outputs values from 0 to 7. It only starts counting when in State 3 (reset is 0), and in all other states, the output is 0 (reset is 1). The backend comparator sets the TDC to 1 when the counter value reaches 7; otherwise, it's 0. The shift register operates as follows: When load is 1, T Data is latched in. The en signal being 1 triggers the internal data shifting action, and the output is the lowest bit.

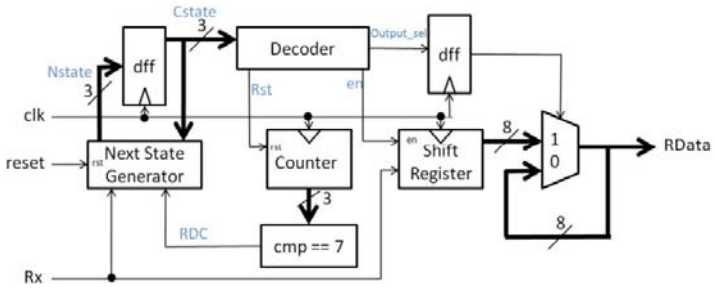


4. Receiver

   Inputs: Reset signal (reset), 115200Hz clock signal (clock), Rx signal.

   Output: Received data.

   The design principle for the receiver is similar to the transmitter. Initially, it waits for the start bit sent from the computer (value should be 0). Upon detecting the start signal, the state transitions from the "detecting" state (State 1) to the "receiving data" state (State 2). In this state, the shift register is enabled, locking in the 8 data bits transmitted by the computer. Simultaneously, the counter starts counting. When the counter reaches 7 (indicating the reception of all 8 bits), the RDC (Receive Data Complete) signal becomes 1; otherwise, it's 0. Upon detecting RDC becoming 1, the state transitions from State 2 to State 3 (detecting stop bit). If Rx data is 1, indicating a correct reception, the state moves to State 4 (output received data). If Rx is not 1, suggesting an error in the 8 bits of data received, the state returns to State 1 without outputting the received data. In State 4, which represents a correct reception, two scenarios can occur: if Rx is 0 (start bit), it means the next data is coming in (continuous transmission), and the state transitions to State 2. If Rx is 1, it returns to State 1 to detect the start bit of the next data. Output selection is determined by the current state. Only in State 4 is it 1, indicating that the received 8-bit value should be outputted. In all other states, the output is 0. The counter is a 3-bit counter, outputting values from 0 to 7. It only starts counting in State 2 (reset is 0), and in all other states, the output is 0 (reset is 1). The backend comparator sets RDC to 1 when the counter value reaches 7; otherwise, it's 0. The shift register only starts shifting data internally when en (enable) is 1. This should happen in State 2, as Rx values are locked in sequentially. After reception is complete, the state outputs data (State 4) and concurrently outputs it.



5. Clock Generator:

   Inputs: 50MHz clock input from DE1 board.

   Output: 115200Hz clock output.

   Utilize a 50MHz clock input connected to a divider of 434 to approximate the frequency of 115200Hz.

6. Debounce Circuit :

   Inputs: Button signal.

   Output: Stable signal output.

   When pressing a button, the signal isn't immediately stable; it experiences a period of bouncing where the voltage fluctuates between high and low states rapidly. This bouncing can lead to receiving multiple signals from a single button press, causing errors. Debouncing is designed to eliminate the instability of the signal. It utilizes a finite state machine to ensure that the input signal is stable before passing it to the next stage of the circuit.

7. Seven-Segment Decoder

   Inputs: Received data

   Output: Seven-segment display.

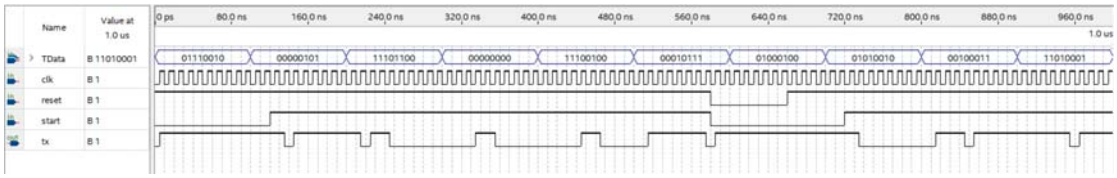
   Decode the value into the corresponding output for the seven-segment display.

II. TX Unit

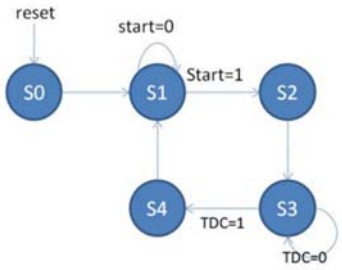
A. Verilog Code and Comment

```
module TX_unit(clk,start,reset,TData,tx);
input         clk,reset,start;
input  [7:0] TData;
output reg    tx;

parameter WAITING_BIT = 1'b1;
parameter START_BIT = 1'b0;
parameter STOP_BIT = 1'b1;

reg    [2:0] cs,ns;
wire         tdc, txd, _tx;
reg    [1:0] OS;
reg          count_rst, shift_en;

//Tx control circuit
//assign ns to cs at every posedge clk
always @(posedge clk)
   cs<=ns;

//state
//trigger at every cs or reset or tdc or start
always@(cs or reset or tdc or start)
if(reset==1'b0) //ns reset to 3'd0 when reset==1'b0
   ns <= 3'd0;
else
   case(cs)
      3'd0:   ns <= 3'd1;                    //reset stage:    when cs=3'd0, ns=3'd1
      3'd1:   ns <= (start)?3'd2:3'd1; //wait push button: when cs=3'd1, ns=3'd2 if start=1, else ns=3'd1
      3'd2:   ns <= 3'd3;                    //tran start bit: when cs=3'd2, ns=3'd3
      3'd3:   ns <= (tdc)   ?3'd4:3'd3;   //shift stage:    when cs=3'd3, ns=3'd4 if tdc=1, else ns=3'd3
      3'd4:   ns <= 3'd1;                    //stop bit:       when cs=3'd4, ns=3'd1
      default:ns <= 3'd0;                    //                else, ns=3'd0
   endcase

//output control signal decoder
//trigger when cs changes
always@(cs)
   case(cs)
   3'd0:begin
      //when cs=3'd0, OS=2'b00, count_rst=1'b1, shift_en=1'b0;
      OS <= 2'b00;
      count_rst <= 1'b1;
      shift_en <= 1'b0;
      end
   3'd1:begin
      //when cs=3'd1, OS=2'b00, count_rst=1'b1, shift_en=1'b0;
      OS <= 2'b00;
      count_rst <= 1'b1;
      shift_en <= 1'b0;
      end
   3'd2:begin
      //when cs=3'd2, OS=2'b01, count_rst=1'b1, shift_en=1'b0;
      OS <= 2'b01;
      count_rst <= 1'b1;
      shift_en <= 1'b0;
      end
   3'd3:begin
      //when cs=3'd3, OS=2'b10, count_rst=1'b0, shift_en=1'b1;
      OS <= 2'b10;
      count_rst <= 1'b0;
      shift_en <= 1'b1;
      end
   3'd4:begin
      //when cs=3'd4, OS=2'b11, count_rst=1'b1, shift_en=1'b0;
      OS <= 2'b11;
      count_rst <= 1'b1;
      shift_en <= 1'b0;
      end
   default:begin
      //else, OS=2'b00, count_rst=1'b1, shift_en=1'b0;
      OS <= 2'b00;
      count_rst <= 1'b1;
      shift_en <= 1'b0;
      end
   endcase

//send value to other modules
count8 U0(.rst(count_rst), .clk(clk), .tdc(tdc));

TXshift U1(.load(start), .clk(clk), .Din(TData), .en(shift_en), .txd(txd));

// mux circuit
//when OS[0]=1, _tx = (OS[1])?(STOP_BIT:START_BIT)
//when OS[0]=0, _tx = (OS[1])?(txd:WAITING_BIT)
assign _tx = (OS[1])?(OS[0]?STOP_BIT:txd):
             ((OS[0])?START_BIT:WAITING_BIT);


always @(posedge clk) //latched tx signal at every posedge clk
   tx <= _tx;

endmodule
```

B. Simulation



At every positive edge of clock, the value of ns will be assigned to cs, while there is a reset when reset is set to 0.
The output of tx is depended by the value of the first and second bit of OS, while the value of OS itself is depended by the current state, which follows the pattern of the finite state machine show below:



state	Description	Decoder Output
S0	Reset State	Rst=1, OS=2'b00,en=0
S1	Waiting for Transmit	Rst=1,OS=2'b00,en=0
S2	Sending Start bit	Rst=1, OS=2'b01,en=0
S3	Sending 8bits Data	Rst=0, OS=2'b10,en=1
s4	Sending Stop bit	Rst=1, OS=2'b11,en=0

TDC (Transmit Data Complete) OS (Output Select)

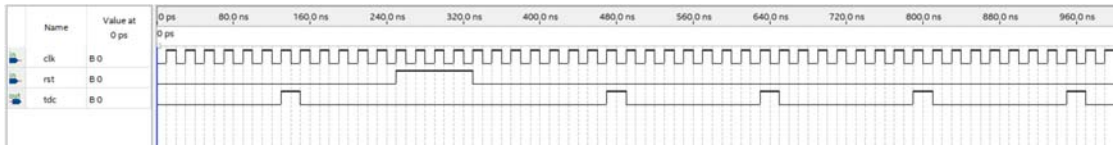
When OS[0]=1, _tx=1'b1 if OS[1]=1, else _tx= 1'b0 .
When OS[0]=0, _tx=txd if OS[1]=1, else _tx= 1'b1 .

III. Count 8

A. Verilog Code and Comment

```
1   module count8(clk,rst,tdc);
2   input     clk,rst;
3   output    tdc;
4
5   reg [2:0] count;
6
7   //triggered at every posedge clk, if rst=1, count =3'd0, else itself+1
8   always@( posedge clk)
9      if(rst)
10     count  <=  3'd0;
11     else
12     count <= count +3'd1;
13
14  assign tdc = &count; //output and single
15
16  endmodule
```

B. Simulation



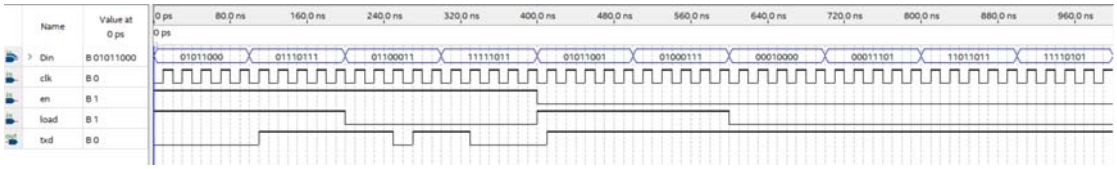
This module is written inside the TX_Unit, when count=3'b111, tdc will output 1, otherwise it remains 0.
That is, it output 1 when clk is counted to 8.

IV. TXshift

A. Verilog Code

```
module TXshift(load, clk, Din, en, txd);
input load, en, clk;
input [7:0] Din;
output txd;

reg [7:0] Data;
wire [7:0] nData;

//triggered at every posedge clk,
//if the input load=1, then the input DIN will be assign to Data
//otherwise, nData will be assign to Data
always @(posedge clk)
if(load)
   Data <= Din;
else
   Data <= nData;

// if en=1, nData=1'b0 plus [7:1] bit of Data, otherwise nData = Data
assign nData = (en)?{1'b0, Data[7:1]} :Data;

assign txd = Data[0]; //the output value would be the [0] bit of Data

endmodule
```

B. Simulation



This module is written inside the TX_Unit.

If en==1, load==1: txd=last bit of Din.

If en==1, load==0: txd=Data[1].

If en==0, load==1: txd=last bit of Din.

If en==0, load==0: txd=last bit of nData.

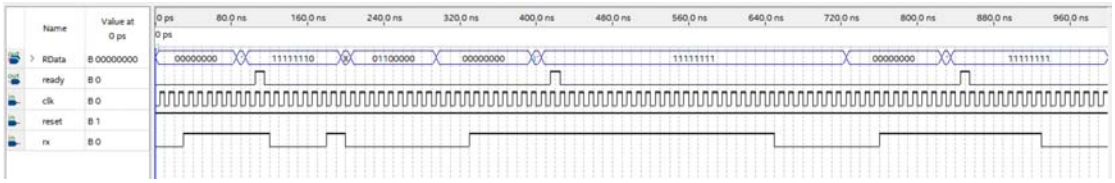
V. RXUnit

A. Verilog Code

```
module RxUnit(reset,clk,rx,RData, ready);

input reset,clk,rx;
output [7:0] RData;
output reg ready;

reg [2:0] cs,ns;
wire rdc;
reg rst,shift_en;

//Rx Control Circuit
//assign ns to cs at every posedge clk
always @(posedge clk)
   cs<=ns;

//state
//trigger at every cs or reset or rx
always @(reset or rx or rdc or cs)
   if(reset==1'b0)
      ns <= 3'd0;
   else
      case(cs)
      3'd0: ns <= 3'd1; //reset stage:    when cs=3'd0, ns=3'd1
      3'd1: ns <= (rx)?3'd1:3'd2;//detect start bit:    when cs=3'd1, ns=3'd1 if rx=1, else ns=3'd2
      3'd2: ns <= (rdc)?3'd3:3'd2;//receive 8 bit data:  when cs=3'd2, ns=3'd3 if rdc=1, else ns=3'd2
      3'd3: ns <= (rx)?3'd4:3'd1;//detect stop bit:     when cs=3'd4, ns=3'd1 if rx=1, else ns=3'd2
      3'd4: ns <= (rx)?3'd1:3'd2;//display the receive data and detect start bit:
                                 //when cs=3'd4, ns=3'd1 if rx=1, else ns=3'd2
      default: ns <= 3'd0;//                 else, ns=3'd0
   endcase

//output control signal decoder
//trigger when cs changes
always @(cs)
   case(cs)
      3'd0:begin
      //when cs=3'd0, rst=1'b1, ready=1'b0, shift_en=1'b0;
      rst <= 1'b1;
      shift_en <= 1'b0;
      ready <= 1'b0;
      end
      3'd1:begin
```

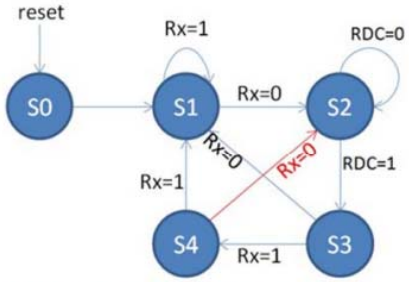
```
43          //when cs=3'd1, rst=1'b1, ready=1'b0, shift_en=1'b0;
44          rst <= 1'b1;
45          shift_en <= 1'b0;
46          ready <= 1'b0;
47          end
48          3'd2:begin
49          //when cs=3'd2, rst=1'b0, ready=1'b0, shift_en=1'b1;
50          rst <= 1'b0;
51          shift_en <= 1'b1;
52          ready <= 1'b0;
53          end
54          3'd3:begin
55          //when cs=3'd3, rst=1'b1, ready=1'b0, shift_en=1'b0;
56          rst <= 1'b1;
57          shift_en <= 1'b0;
58          ready <= 1'b0;
59          end
60          3'd4:begin
61          //when cs=3'd4, rst=1'b1, ready=1'b1, shift_en=1'b0;
62          rst <= 1'b1;
63          shift_en <= 1'b0;
64          ready <= 1'b1;//data received complete
65          end
66          default:begin
67          //else, rst=1'b1, ready=1'b0, shift_en=1'b0;
68          rst <= 1'b1;
69          shift_en <= 1'b0;
70          ready <= 1'b0;
71          end
72          endcase
73
74      //import other modules
75      count8 U0(.rst(rst), .clk(clk), .tdc(rdc));
76      RXshift U1(.rst(rst),.rxd(rx),.clk(clk),
77                 .en(shift_en),.rdc(rdc),.Dout(RData));
78
79      endmodule
80
```

B. Simulation



At every positive edge of clock, the value of ns will be assigned to cs, while there is a reset when reset is set to 0.

The output of RData is depended by the module of count8 and RXshift, where the state of the rx follows the finite state machine as follow:



The red state transition occurs during continuous data

state	Description	Decoder Output
S0	Resrt State	Rst=1,en=0,OS=0
S1	Detecting the start bit	Rst=1,en=0,OS=0
S2	Receiving 8-bit data	Rst=0,en=1,OS=0
S3	Detecting the stop bit	Rst=1,en=0,OS=0
S4	Displaying received data and detecting the start bit of consecutive transmitted data	Rst=1, en=0,OS=1

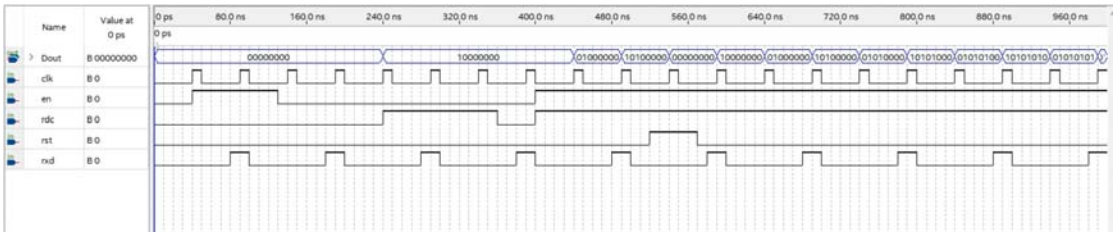
RDC (Receive Data Complete) OS (Output Select)

VI. RXshift

A. Verilog Code

```
module RXshift(rst,rxd,clk,en,rdc,Dout);

input rst,rxd,en,clk,rdc;
output [7:0] Dout;
reg [7:0] Data;
wire [7:0] nData;

//triggered at every posedge clk,
//if the input rst=1, then Data will be reset to 8'd0
//otherwise, nData will be assign to Data
always @(posedge clk)
   if(rst)
      Data <= 8'd0;
   else
      Data <= nData;

// if en=1, nData=rxd plus [7:1] bit of Data, otherwise nData = Data
assign nData = (en)? {rxd,Data[7:1]} :Data;

assign Dout=(rdc)? Data: Dout; //Latch Received Data

endmodule
```

B. Simulation



This module is written inside the RXUnit.
If rdc=1, Data will be assign to the output Dout, while when en=1, the value of Data will be shifted and stored to nData, then it will be reassigned to Data and be the output.

VII. One Shot

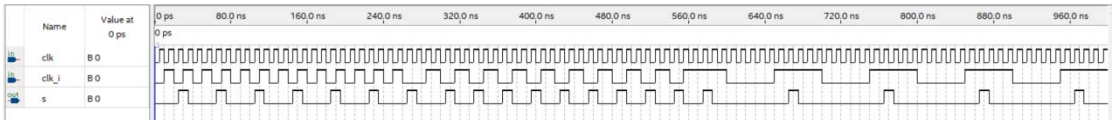
A. Verilog Code

```
module one_shot(clk,clk_i,s);
input clk,clk_i;
output reg s;
reg [1:0] cs,ns;
wire _s;

always @(posedge clk)   //triggered when clk is in posedge
   cs<=ns;  //assign ns to cs

always @(cs) //when the value of cs differs, followed by the value of cs, we have following conditions
   case(cs)
   2'b00: ns <= (clk_i)?2'b01:2'b00; //if clk_i=1, assign 2'b01 to ns, else assign 2'b00 to ns
   2'b01: ns <= 2'b10; //assign 2'b10 to ns
   2'b10: ns <= (clk_i)?2'b10:2'b00; //if clk_i=1, assign 2'b10 to ns, else assign 2'b00 to ns
   default: ns <=2'b00; //otherwise, assign 2'b00 to ns
   endcase

always @(posedge clk)
   s <= _s;
assign _s = (cs==2'b01)?1'b1:1'b0; //if cs equals to 2'b01, s will be 1'b1, else it will be 1'b0

endmodule
```

B. Simulation

The input of clk is the output of frequency divider, while the input of clk_i is the push bottom.
When clk is in posedge, one_shot will judge the value of clk_i, and assign the value of cs corresponding to the states.

After that, the value of s would be assigned corresponding to the status of cs.

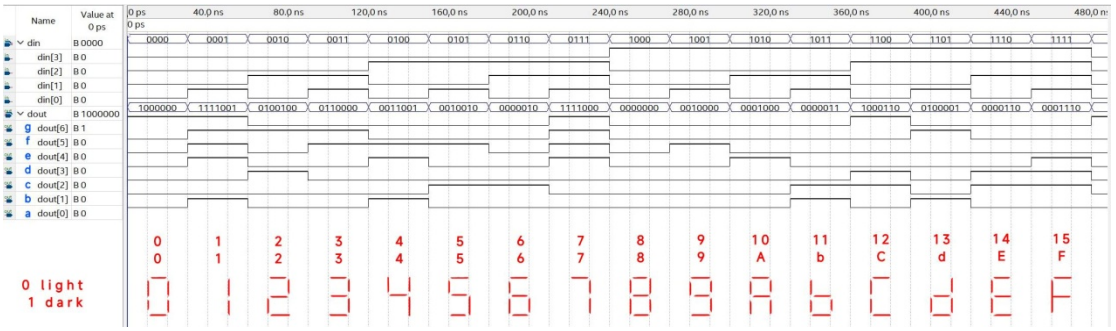


VIII. 7 Segment Decoder

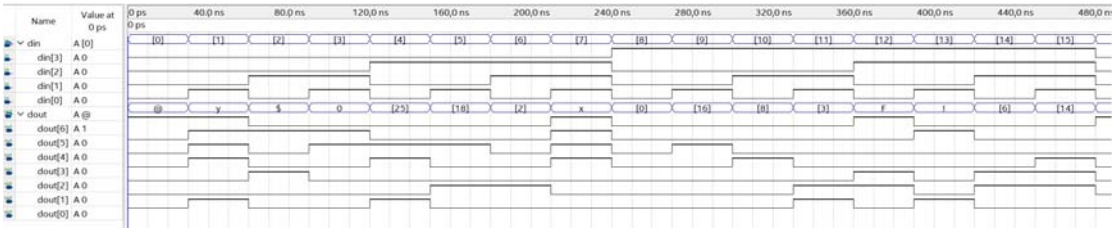
A. Verilog Code and Comment

```
1  module hw01_7seg(din,dout);
2  input    [3:0] din; //4 bits of binary input din
3  output   [6:0] dout; //7 bits of output (7-segments), represented in hexidecimal number)
4  reg      [6:0] dout; //7 bits of registor
5
6  always@(din) //trigger when din changed
7  begin
8     case(din)         //gfedcba, 0:light; 1:dark
9        4'b0000:dout<=7'b1000000; //0
10       4'b0001:dout<=7'b1111001; //1
11       4'b0010:dout<=7'b0100100; //2
12       4'b0011:dout<=7'b0110000; //3
13       4'b0100:dout<=7'b0011001; //4
14       4'b0101:dout<=7'b0010010; //5
15       4'b0110:dout<=7'b0000010; //6
16       4'b0111:dout<=7'b1111000; //7
17       4'b1000:dout<=7'b0000000; //8
18
19       /*complete the remaining part!! */
20       4'b1001:dout<=7'b0011000; //9
21       4'b1010:dout<=7'b0001000; // A
22       4'b1011:dout<=7'b0000011; // b
23       4'b1100:dout<=7'b1000110; // C
24       4'b1101:dout<=7'b0100001; // d
25       4'b1110:dout<=7'b0000110; // E
26       4'b1111:dout<=7'b0001110; // F
27
28    endcase
29 end
30 endmodule
31
```

B. Simulation



Convert the binary value into ASK-II code to check if the decoder is correct:



IX. Frequency Divider

A. Verilog Code and Comment

```
module divb(clkin, clkout);

input clkin;
output reg clkout;
reg [31:0] count;
wire [31:0] divn, divnh;

assign divn= 32'd434;//50MHz/434=115200Hz
assign divnh= divn>>1;//define divnh to be half of divn

always@(posedge clkin) //while clkin is in posedge
begin
    if((count>=divn)) //if counter is larger than the specified period of time
        count<=1; //restart the counter
    else
        count<=count+1; //otherwise keep counting
end

always@(negedge clkin) //wile clkin is in negedge
clkout= (count<= divnh)?1:0;
//output of the frequncy diveder is 1 while counting from 1 to the half period, otherwise is 0

endmodule
```

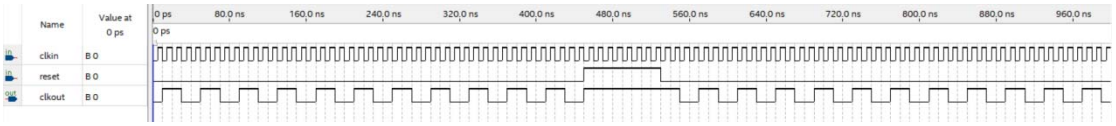
B. Simulation

The code of 10Hz and 50kHz frequency divider are approximately the same, the only difference of the two is its frequency division ratio.

Hence, to simulate, we replace both of the frequency division ratio 32’d5000000 and 32’d1000 into 32’d4 in order to make the simulation better be seen:

```
assign divn= 32'd4;
```

After modifying, we generate a frequency divider that can convert the input frequency into 1/4 times of the frequency.



When clkin is in posedge, the value of count will be varied, while if reset is 1 at this moment, then 1 will be assigned into count.

X. Reflection

The recent electrical engineering experiment delved into the complexities of creating a Data Communication Interface – UART using Verilog and an FPGA board, incorporating the RS232 communication protocol. This undertaking proved to be exceptionally intricate, demanding not only an abundance of code but also intricate procedures, including the additional step of

downloading a program to execute ASCII code input and output.

I am immensely grateful for the teaching assistant's patience during the lab session, especially when helping us troubleshoot issues. If this electrical engineering experiment were not just a one-credit course and didn't coincide with exams in other classes, I might have found it to be an engaging and enjoyable experience. My heartfelt admiration goes out to everyone in the IC team.