Digital Lab 3:

Experiment 6:

Data Communication Interface --- UART

Date: 2023/12/07

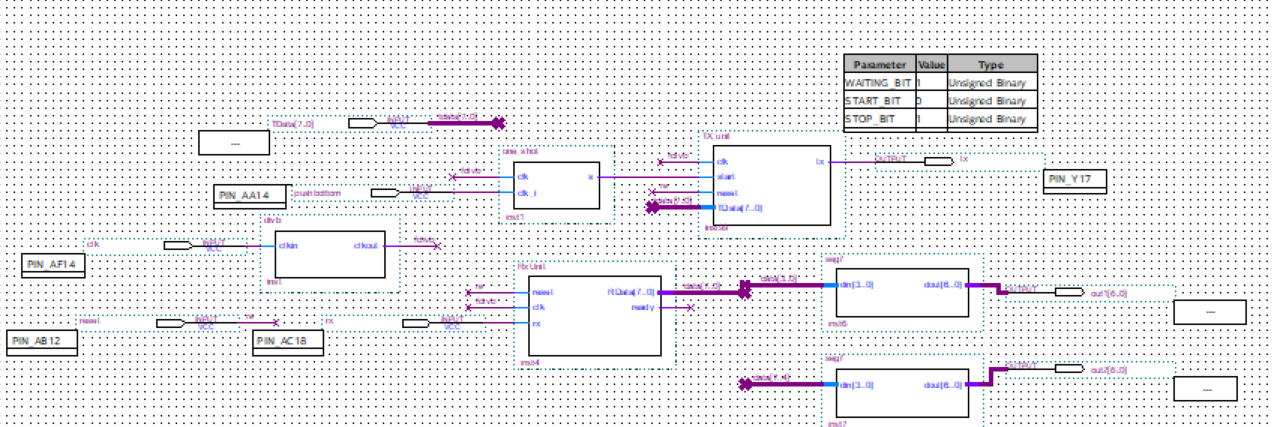
Class: 電機三全英班

Group: Group 11

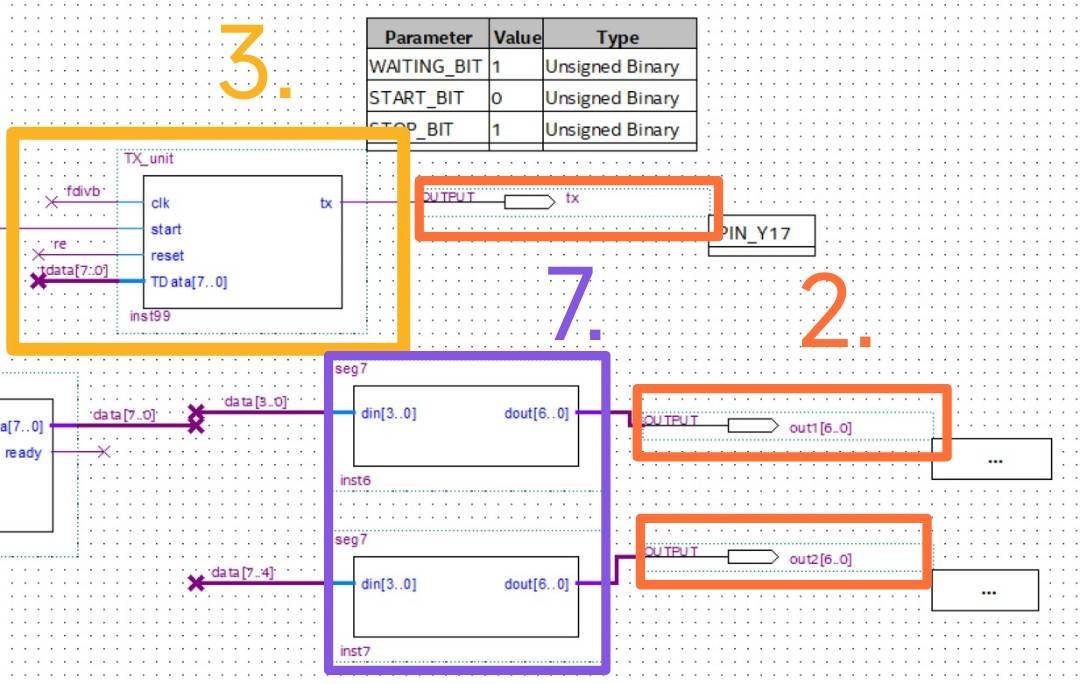
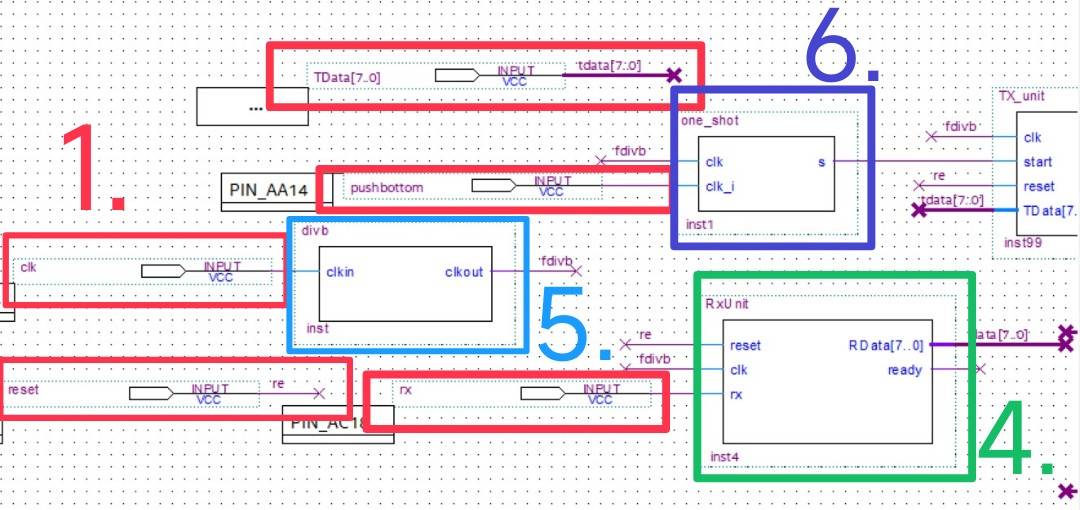
Name: B103105006 胡庭翊

1. Block Diagram

Structure:



Function:



Descriptions:

1. Inputs:

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

1. Outputs:

The outputs are two sets of seven-segment displays, RS232 Tx signal

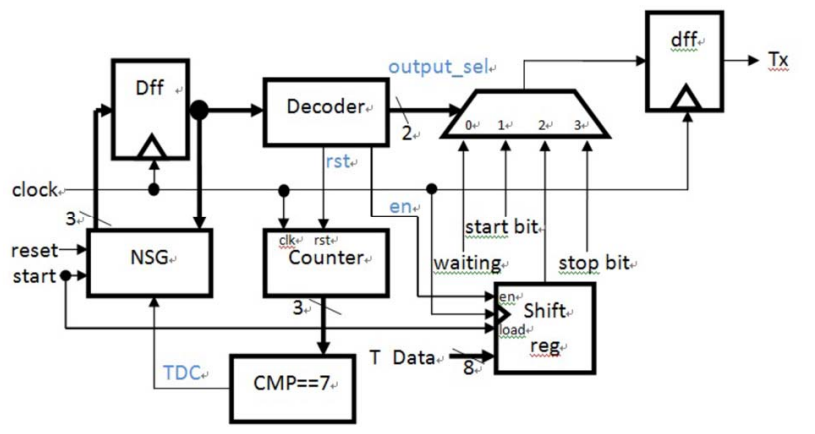
1. Transmitter: 

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.

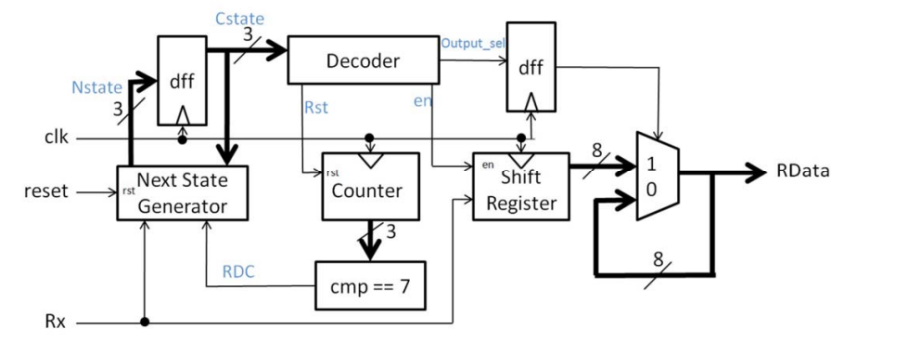


1. Receiver 

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.



1. 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.

1. 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.

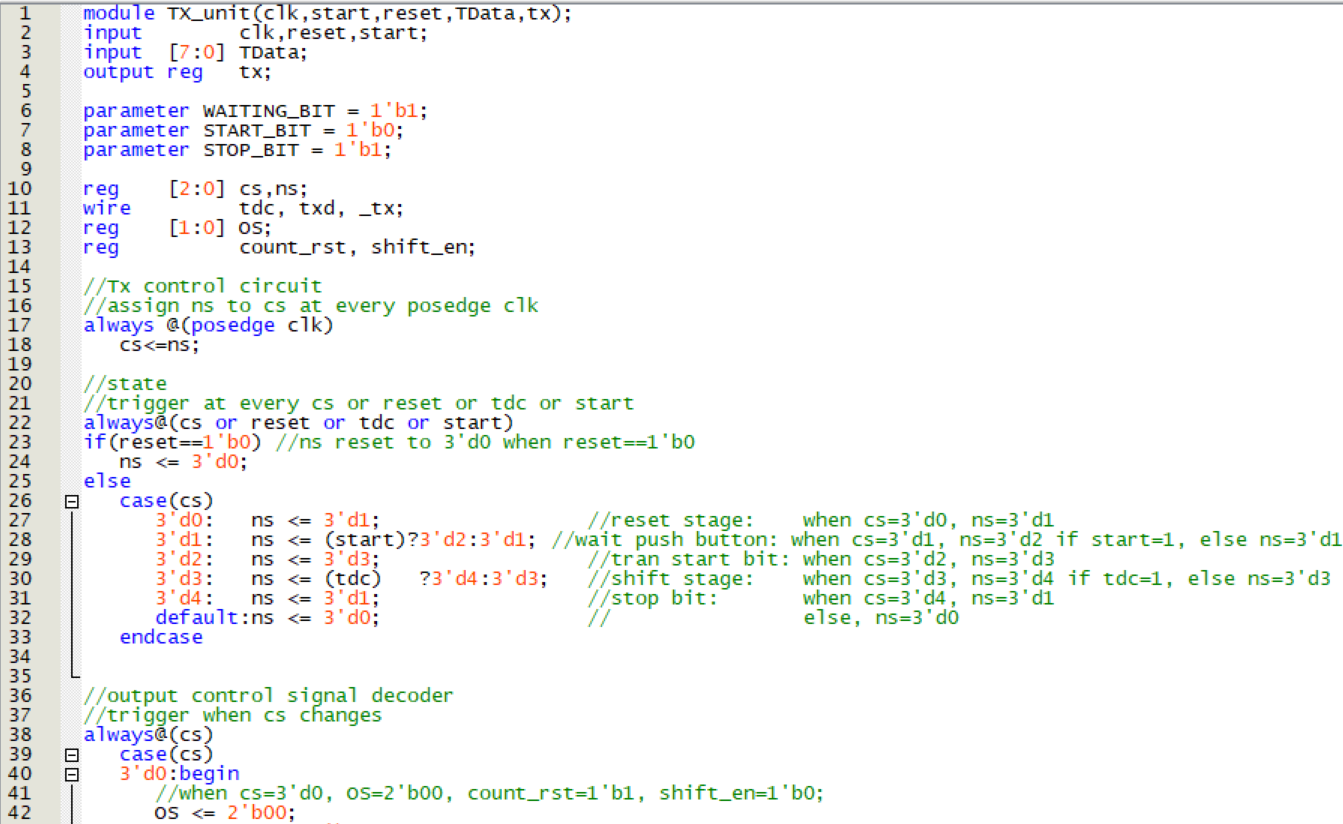
1. Seven-Segment Decoder 

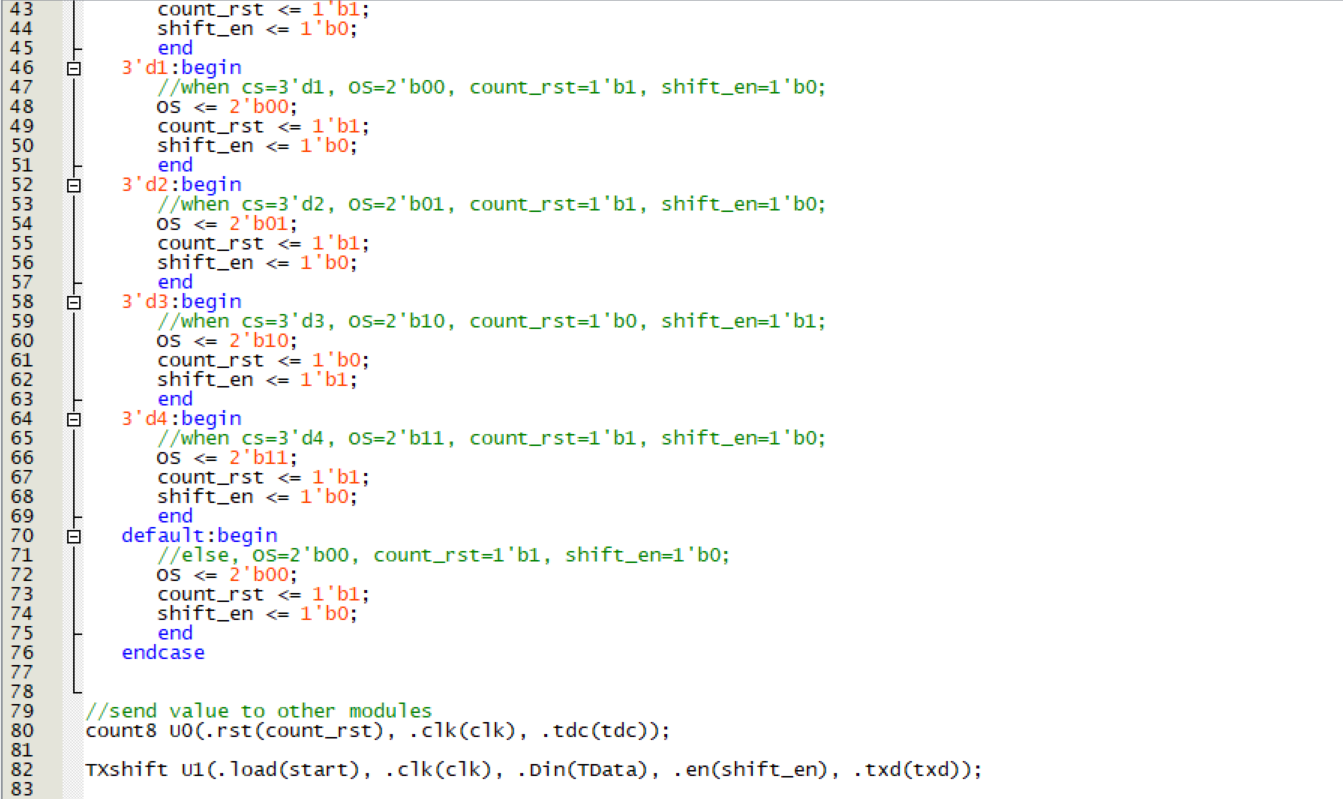
Inputs: Received data

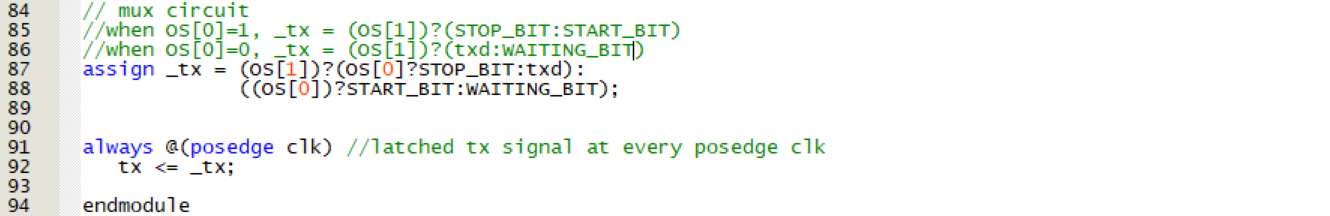
Output: Seven-segment display.

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

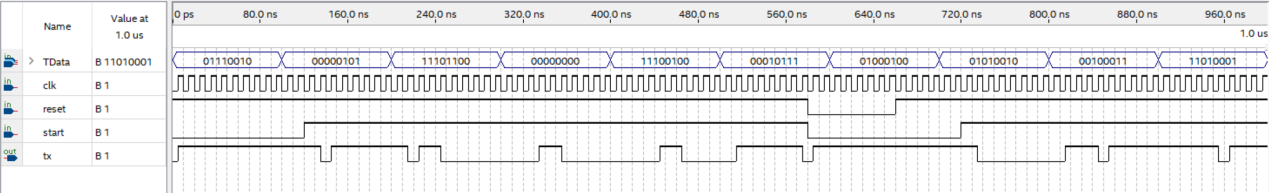
1. TX Unit
2. Verilog Code and Comment





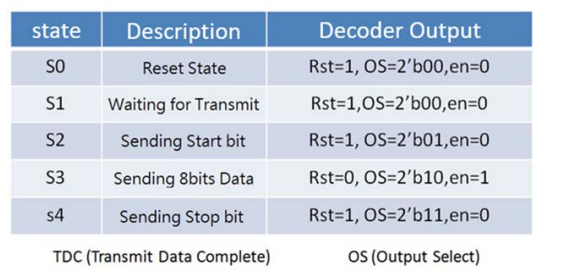
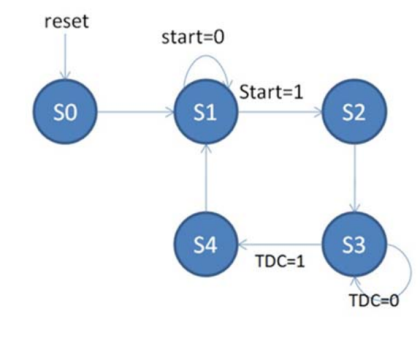


1. Simulation



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.

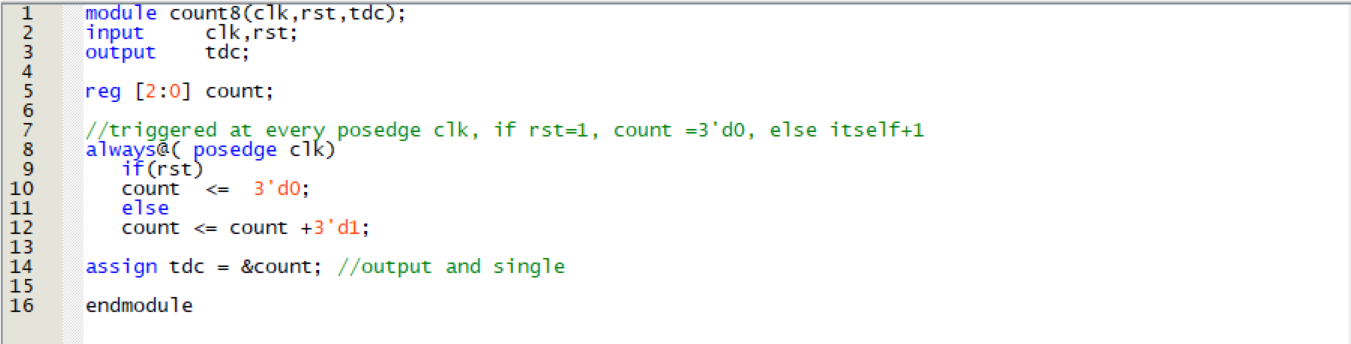
The output of tx is depended by the value of the first and second bit of OS, while the value of OS itself is depended by the current state, which follows the pattern of the finite state machine show below:



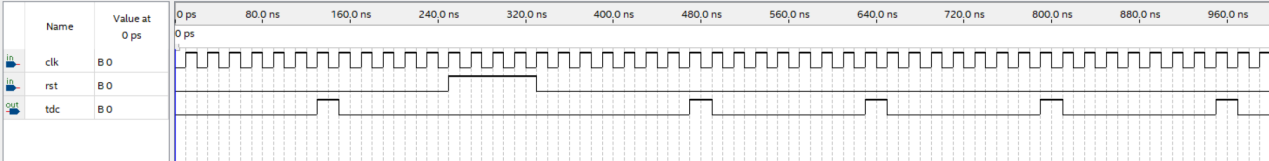
When OS[0]=1, \_tx=1’b1 if OS[1]=1, else \_tx= 1’b0 .

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

1. Count 8
2. Verilog Code and Comment



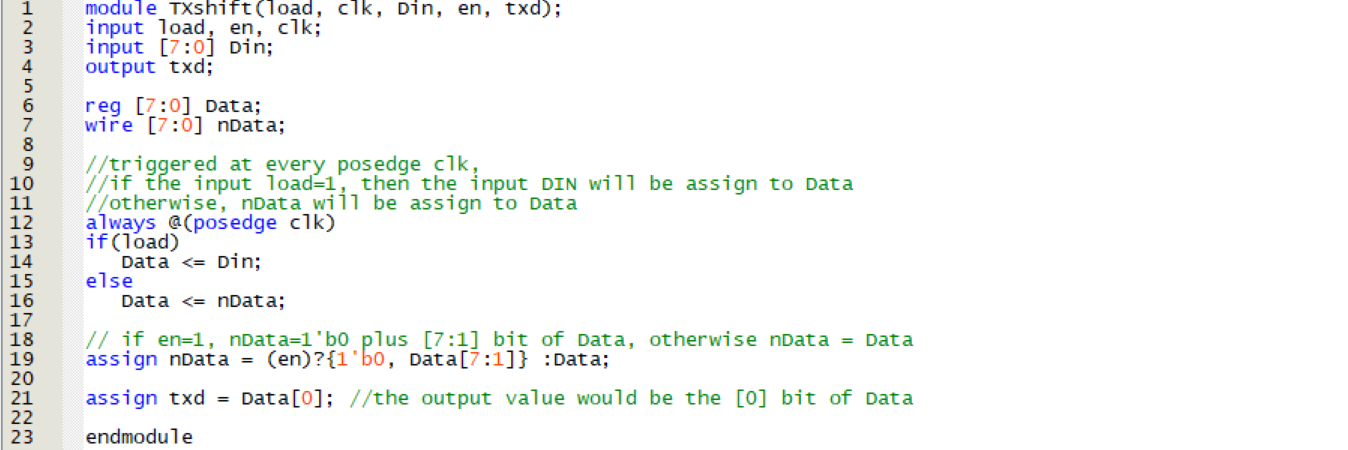
1. Simulation



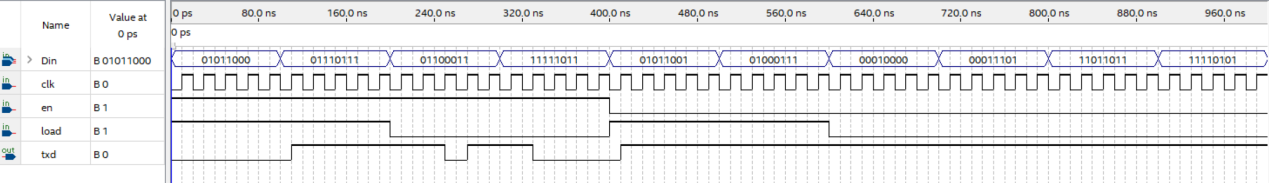
This module is written inside the TX\_Unit, when count=3’b111, tdc will output 1, otherwise it remains 0.

That is, it output 1 when clk is counted to 8.

1. TXshift
2. Verilog Code



1. Simulation



This module is written inside the TX\_Unit.

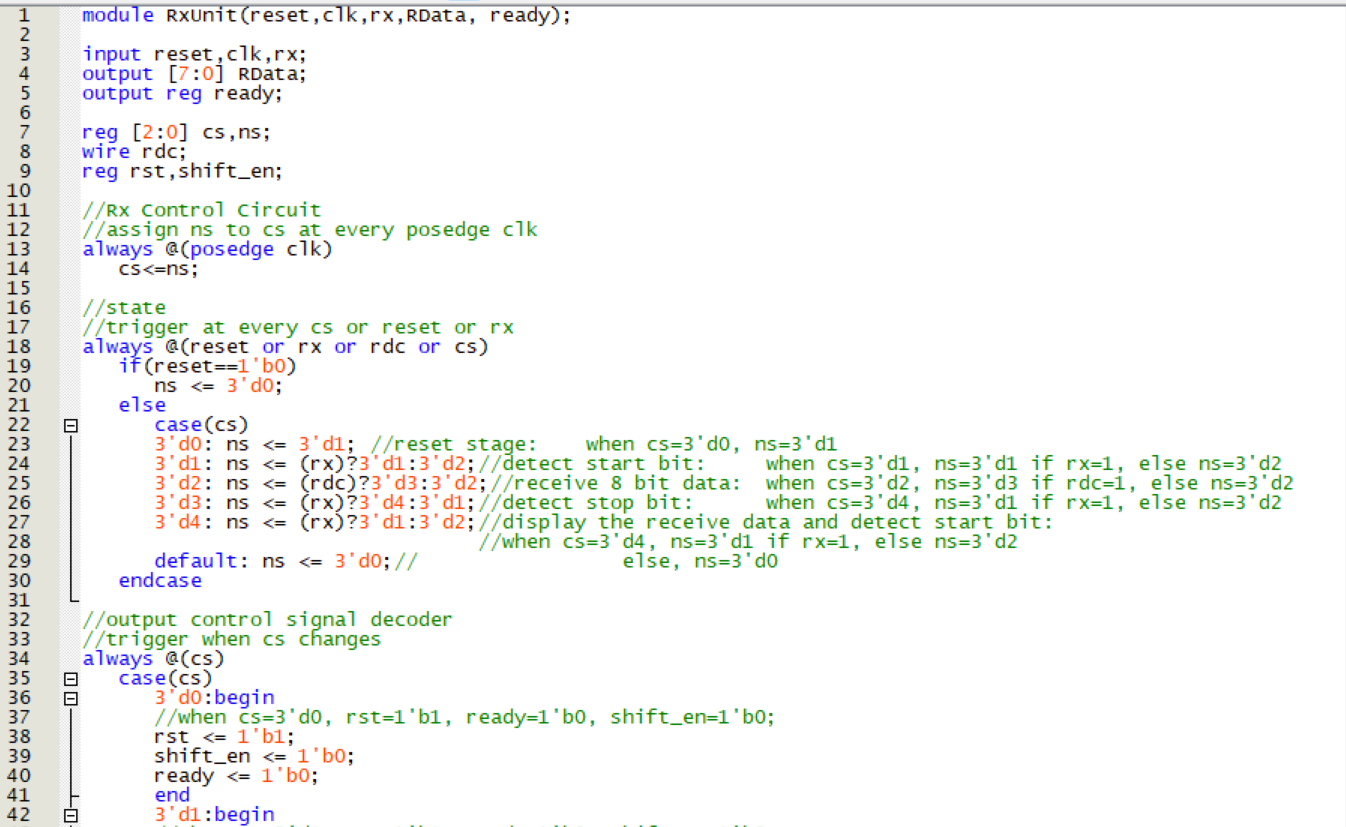
If en==1, load==1: txd=last bit of Din.

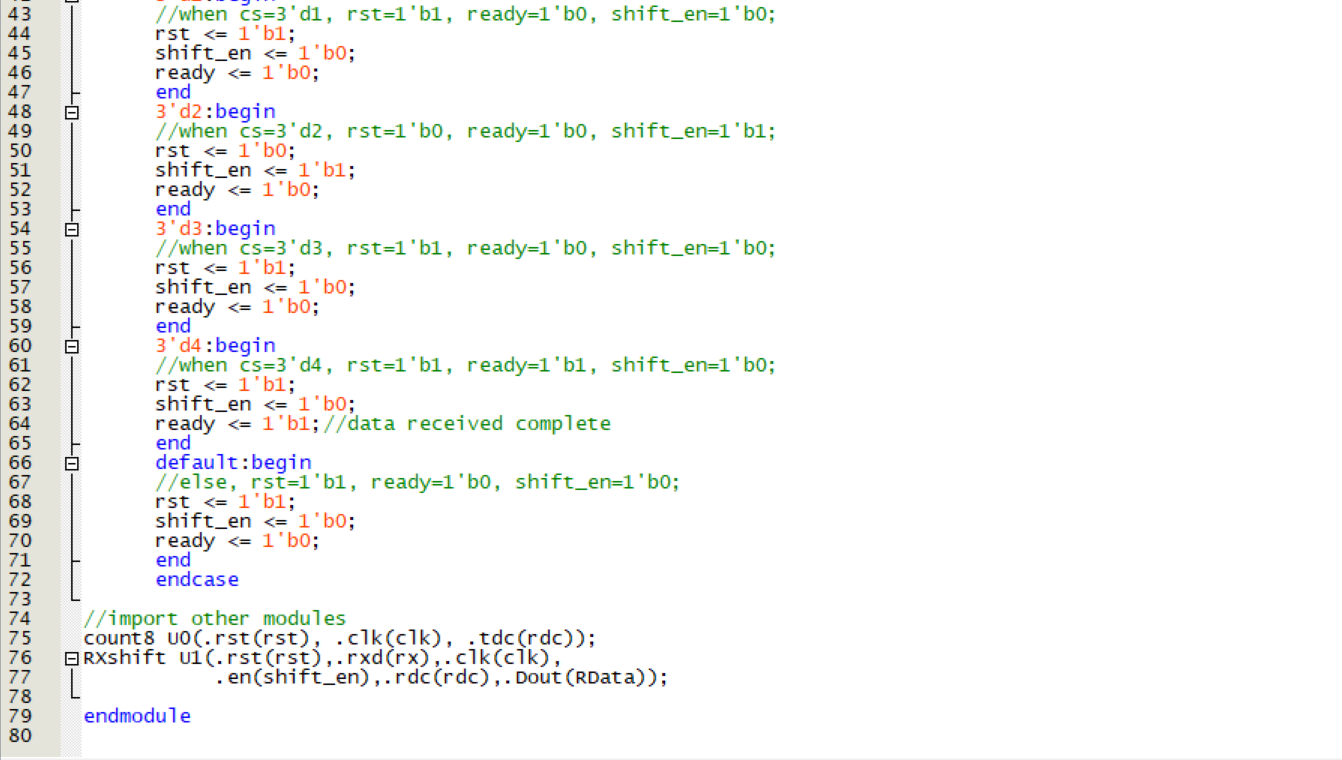
If en==1, load==0: txd=Data[1].

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

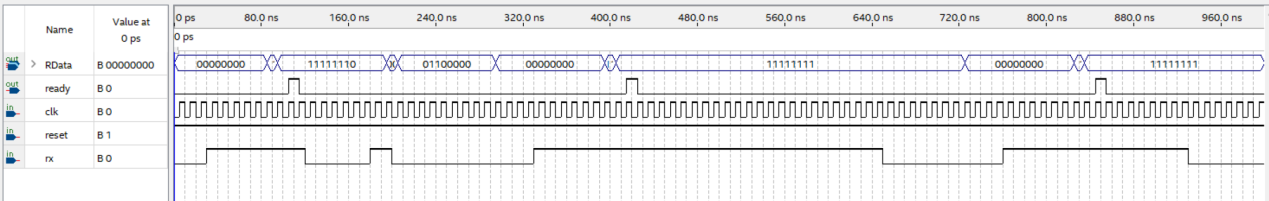
If en==0, load==0: txd=last bit of nData.

1. RXUnit
2. Verilog Code



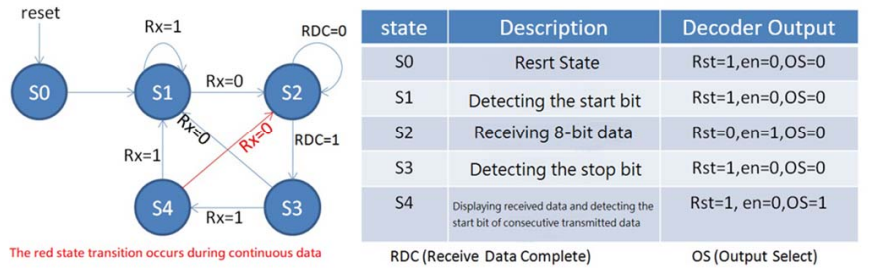


1. Simulation

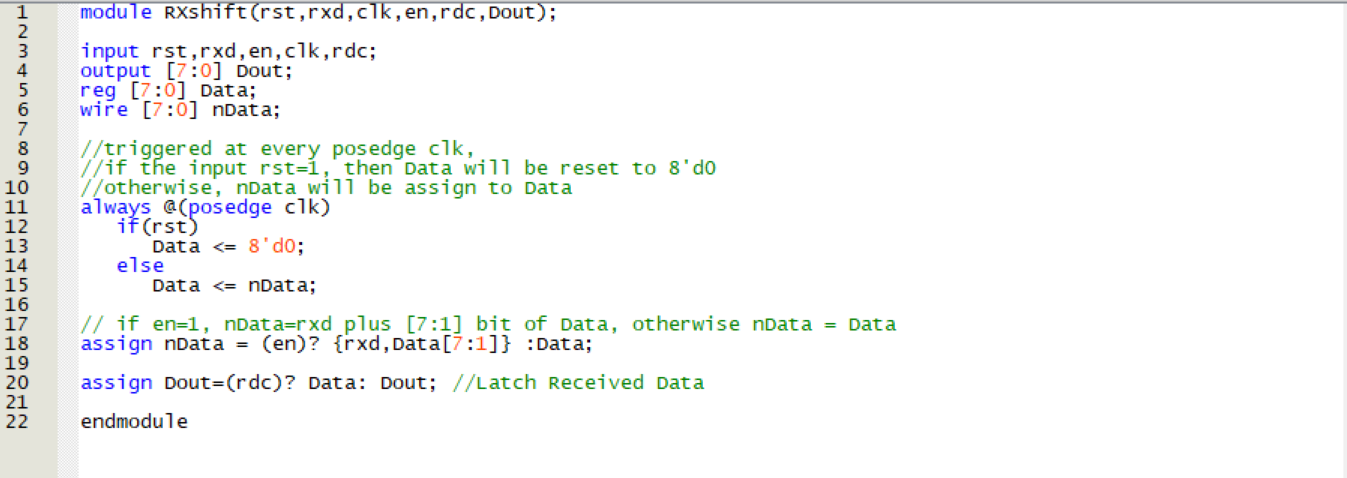


At every positive edge of clock, the value of ns will be assigned to cs, while there is a reset when reset is set to 0.

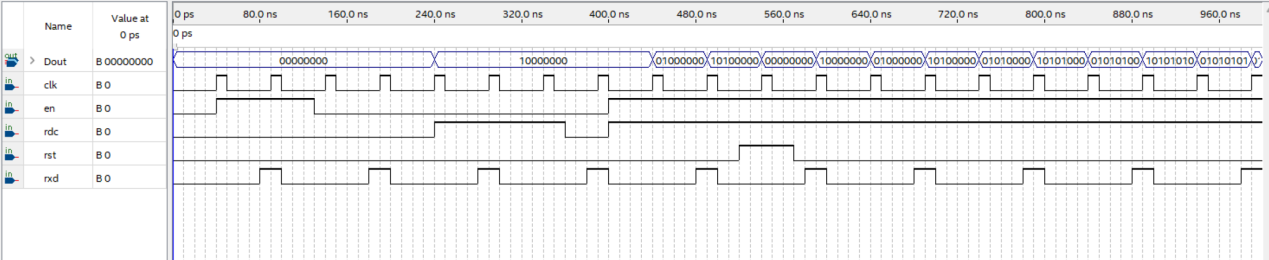
The output of RData is depended by the module of count8 and RXshift, where the state of the rx follows the finite state machine as follow:



1. RXshift
2. Verilog Code



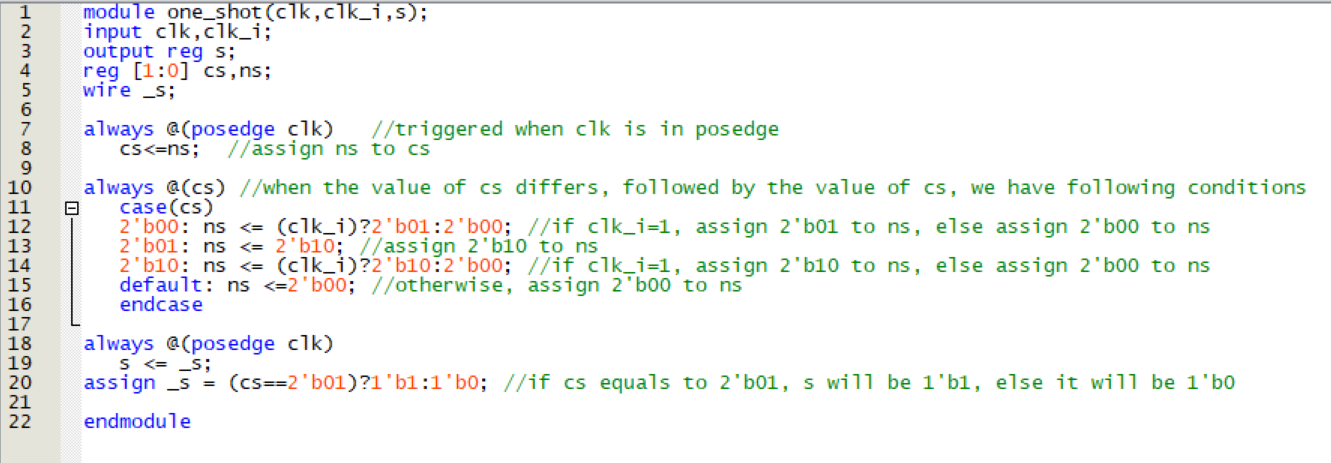
1. Simulation



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.

1. One Shot
2. Verilog Code

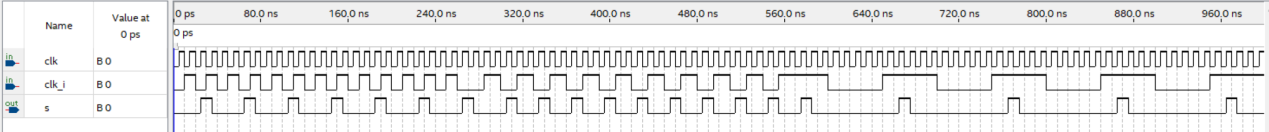


1. Simulation

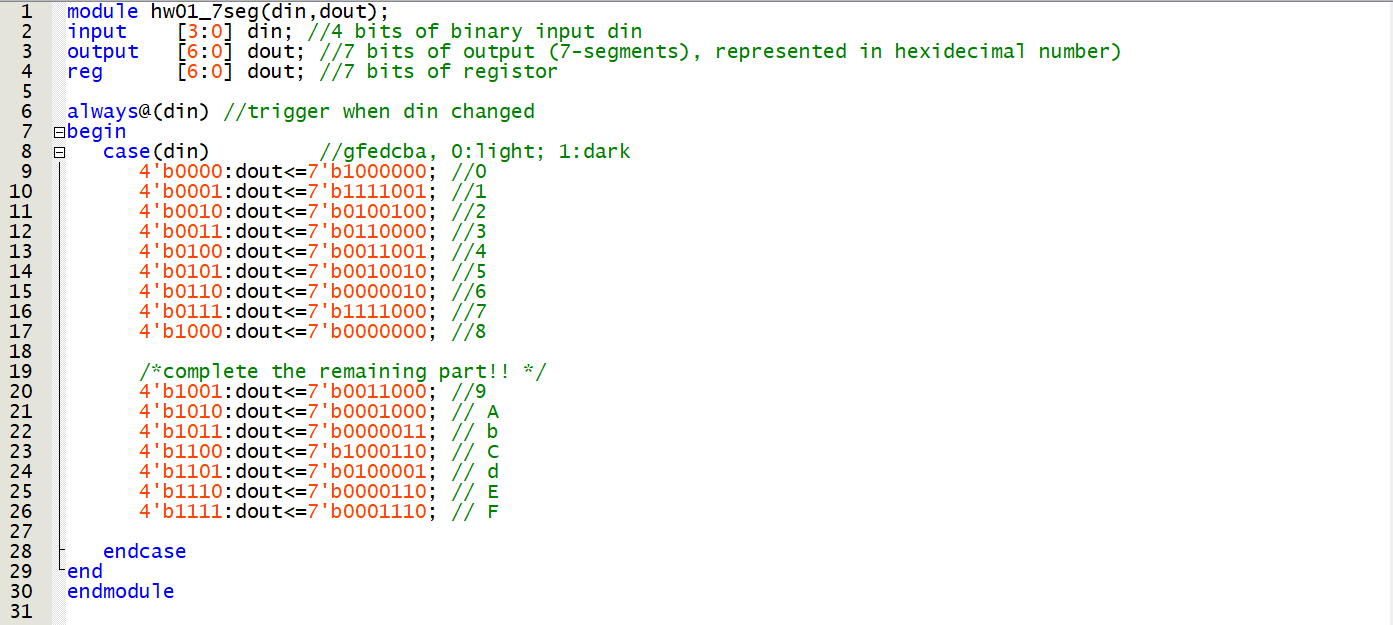
The input of clk is the output of frequency divider, while the input of clk\_i is the push bottom.

When clk is in posedge, one\_shot will judge the value of clk\_i, and assign the value of cs corresponding to the states.

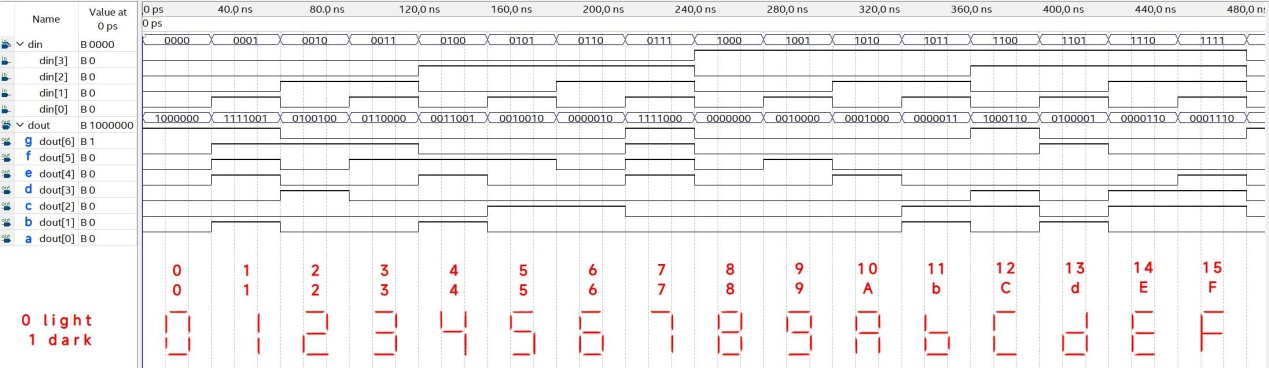
After that, the value of s would be assigned corresponding to the status of cs.



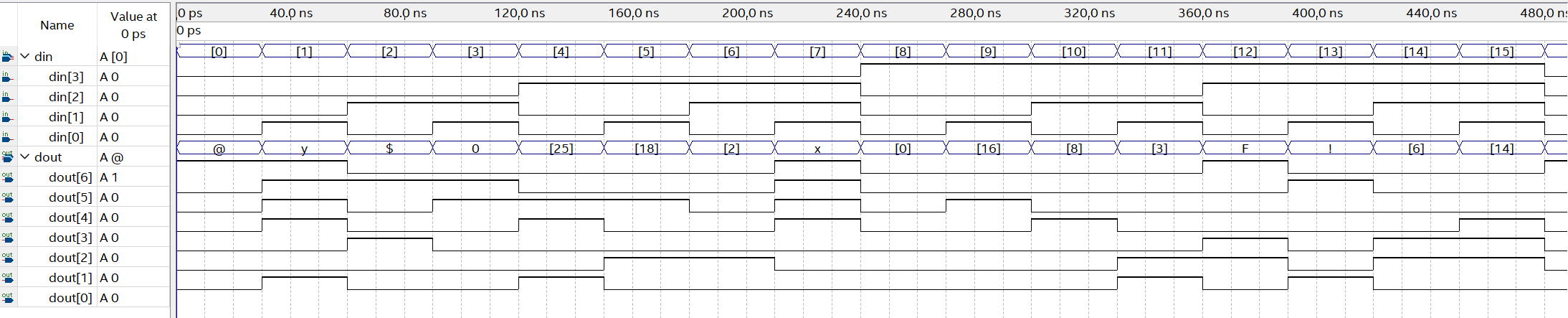
1. 7 Segment Decoder
2. Verilog Code and Comment



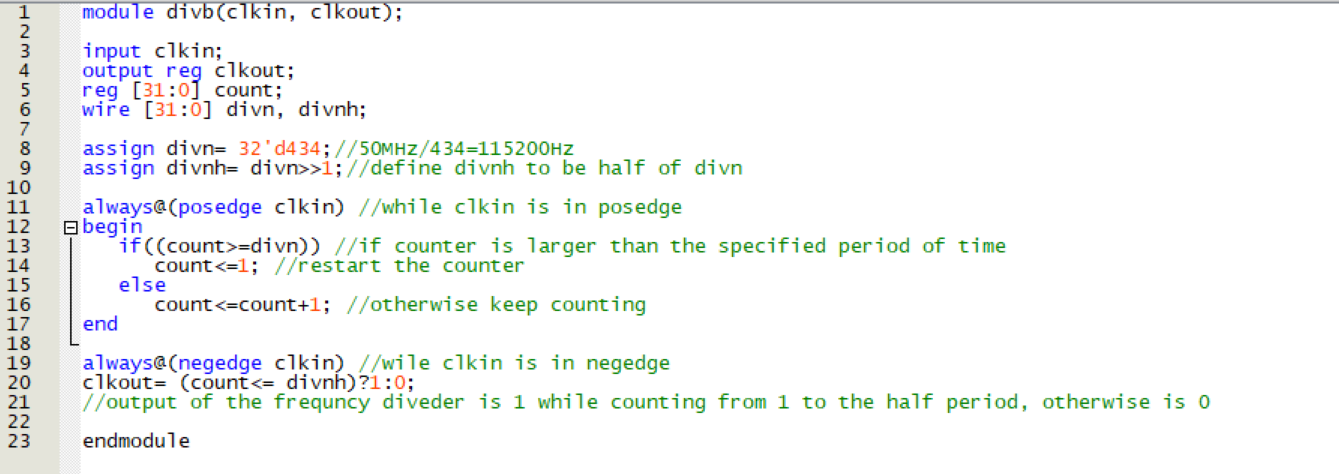
1. Simulation



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



1. Frequency Divider
2. Verilog Code and Comment



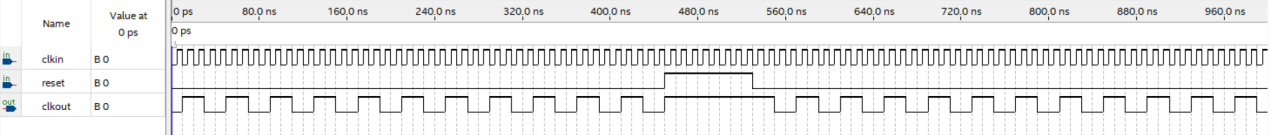
1. Simulation

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:



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.

1. 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.