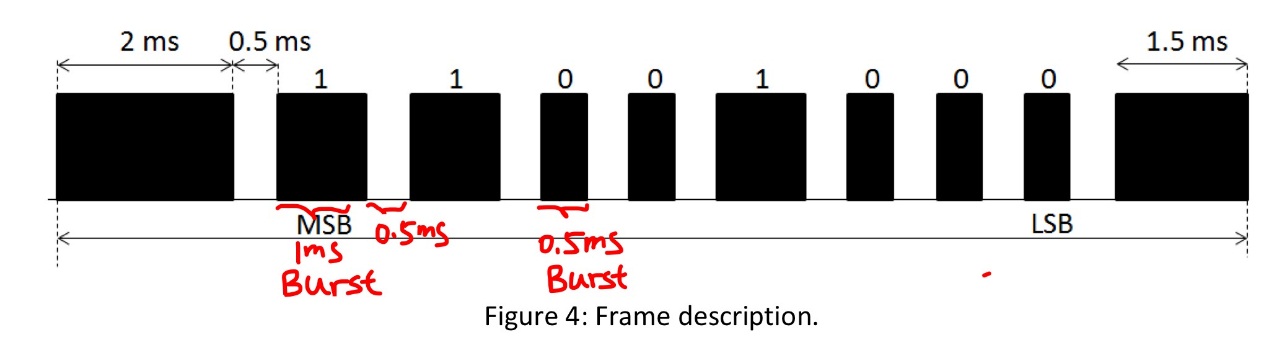
Part I Data Transmitter

This part utilise the IRDA\_TXD module in the DE1 board to transmit an 8 bit binary number to another board. The data is transmitted in format as the flowing diagram shows.



Why are there lead and stop codes?

Lead and ending codes are designed to be different from the data area. This is for the convenience for the receiver to detect when to start reading data and when to finish.

Step 1 Design 38kHz Clock

**What is the duration of 50 MHz and 38 kHz signals?**

The clock cycle duration can be calculated by

50MHz: 1/50\*10^6=20ns

38kHz: 1/38\*10^3=26.3us

**How many cycles of 50 MHz signal do we need in order to achieve one cycle of 38 kHz signal?**

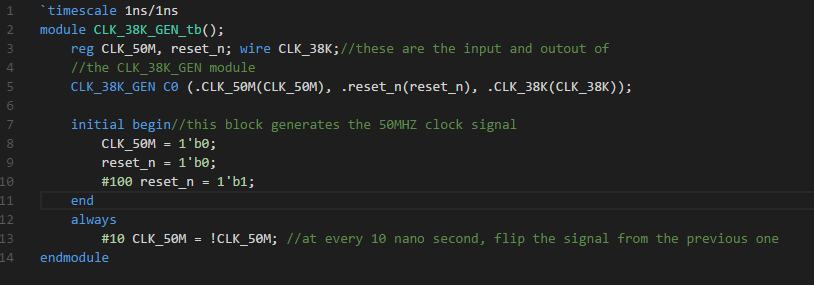
This can be calculated by 26.3us/20ns=1315

HALF\_CYCLE\_DURATION is used since a 1 in the clock only take half of the duration. It’s calculated by 1315/2=657.5

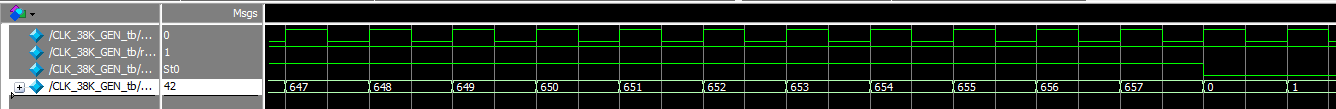
**How many bits do we need for the counter?**

The counter will reach its maximum at 658(base 10) therefore, the binary bit it will required is floor ( ) +1 =10.

The test bench that used was attached below with comments.



This is the simulation of CLK\_38\_GEN module in ModelSim



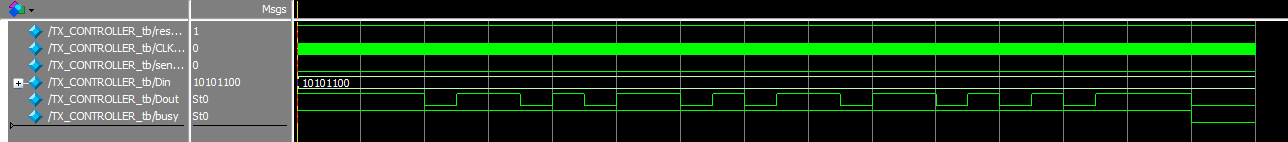
By the ratio between 50M and 38k, around 657 cycles of 50MHz clock is counted as half cycle of the 38kHz clock.

Part I Data Transmitter

Step 2 implementing the finite state machine

The finite state machine is implemented to be: the default state is **STATE\_IDEAL**, if send\_en is HIGH, set Dout=1 to for send the lead high later, jump to **STATE\_LEAD\_HIGH**, which make Dout=1 remain for 2ms, When this is finished, change to **STATE\_LEADER\_LOW**, and set Dout=0. After STATE\_LEADER\_LOW is finished, **STATE\_SENDING\_DATA** will be reached. In this state, if the data haven’t finished transmitting, jump to state **STATE\_SEND\_0** or **STATE\_SEND\_1**, depend on this bit of date is 0 or 1. Shift the DataReg to the left by one bit and fill the LSB with 0 in each sending cycle. When the SentCounter reach 8, jump to the next state **STATE\_STOP** to send the ending burst. (For detail explanation, please refer to the comments of the code).

This is the simulation of the TX\_CONTROLLER module. The output representing data 10101100 is shown in Dout.

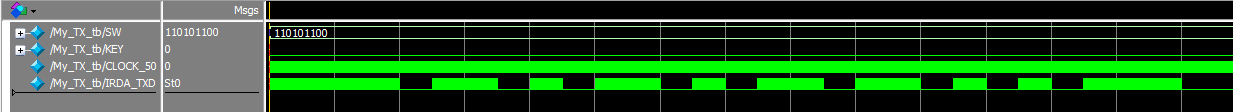


Step 3 Modulate the output signal

The signal is modulated with the 38 kHz clock to create bursts.

The method to obtain that is “assign IRDA\_TXD = Dout & CLK\_38K;”, whenever Dout from TX\_CONTROLLER is 1 and the clock is one, output signal to the IRDA\_TXD module on the board transmit a 1. This is done is the top module MyTX.

A simulation in ModelSim has been performed as below.



Part II Data Receiver

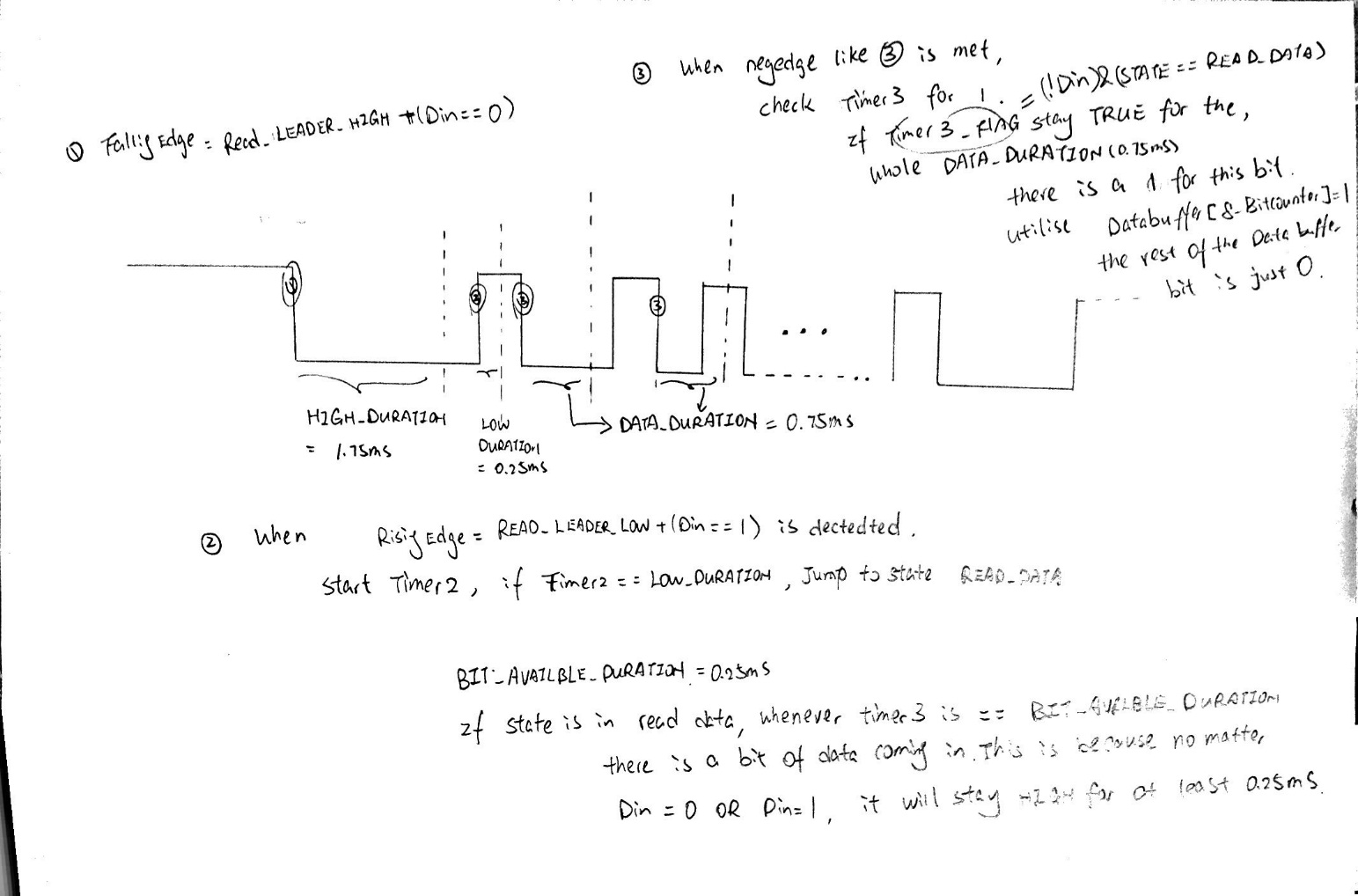
Step 1: Design RX\_CONTROLLER

We modified the original finite state machine by adding one more state to it just for better understanding.

Here’s the state diagram

NB. the input signal is the reverse of the output signal in TX\_CONTROLLER

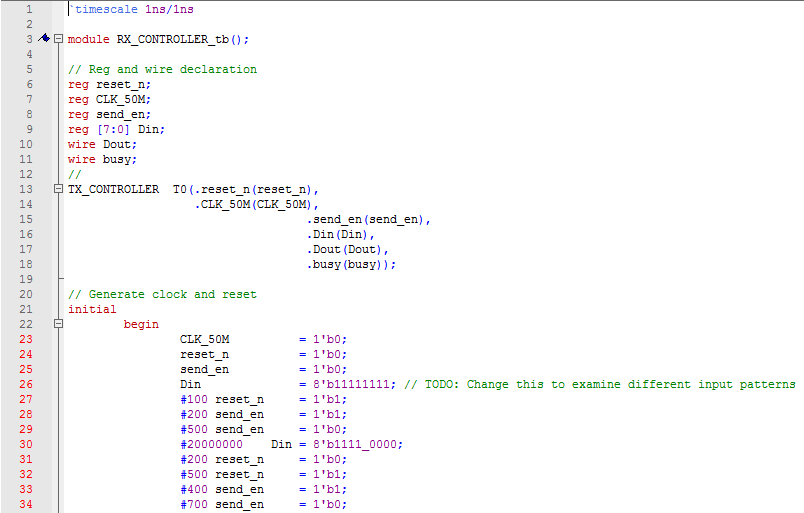
The mechanism of how the RX\_CONTROLLER reads data and is briefly explain in the diagram below.

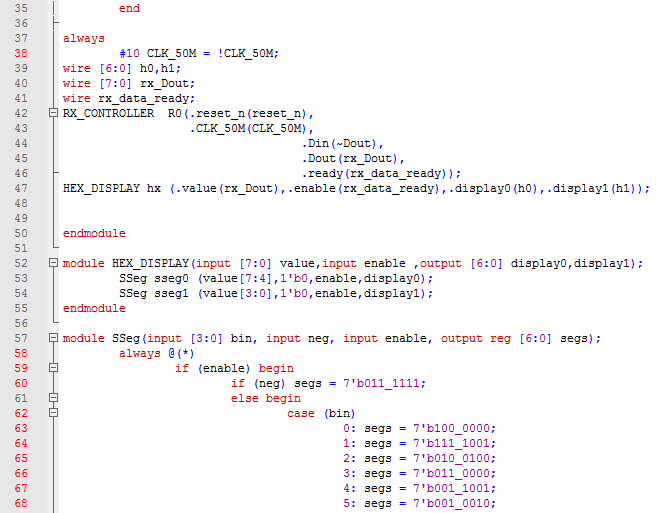


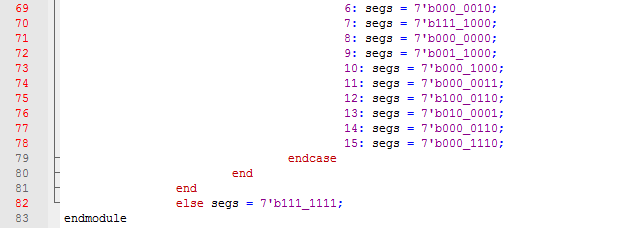
Step2: the reuse of DispHex module from project 1

The module of DispHex is reused from project 1 and it has been included in the test bench for testing purpose.

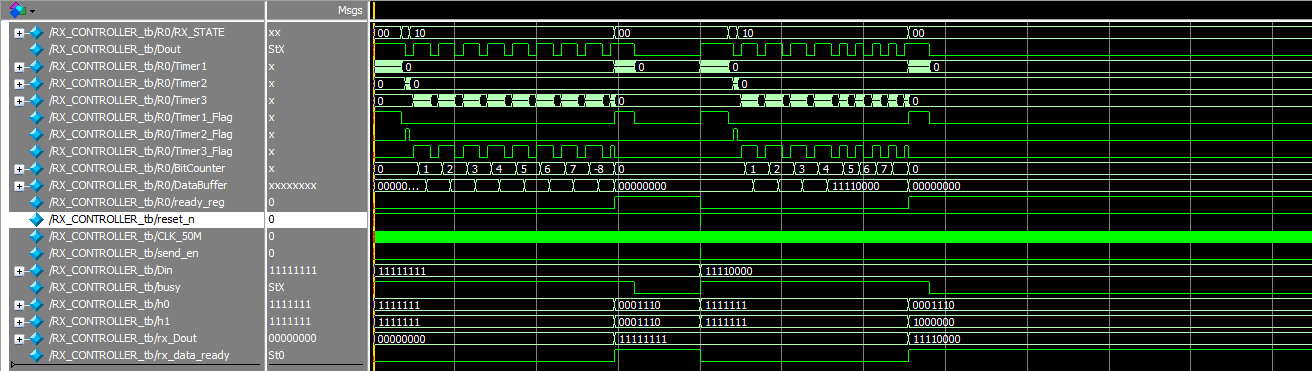
This is the code for test bench.







This is the ModelSim Simulation



Step 3: Top module design

After passing the test in step2, this step is just doing I/O, pin assignment, and instantise a RX\_CONTROLLER and HEX\_DISPLAY by doing

RX\_CONTROLLER rxcontrl (.reset\_n(SW[8]),.CLK\_50M(CLOCK\_50),.Din(IRDA\_RXD),.Dout(Dout),.ready(Ready));

HEX\_DISPLAY display (.value(Dout),.enable(Ready), .display0(HEX0),.display1(HEX1));

**The above design uses 50 MHz clock. What changes need to be made if the system clock of the receiver is 49 MHz or 51 MHz?**

The parameters such as HIGH\_DURATION, LOW\_DURATION need to be changed because of the change in frequency.