COL 215-Digital Logic and System Design

UART Protocol Establishment for Serial Communication(Report-III)

Udit Jain , Shashwat Shivam

Specifications

We want to design a module on FPGA which will transmit an ASCII characters from board to PC at an agreed frequency say 9600 Baud rate.

This will happen by an agreed upon protocol between them.

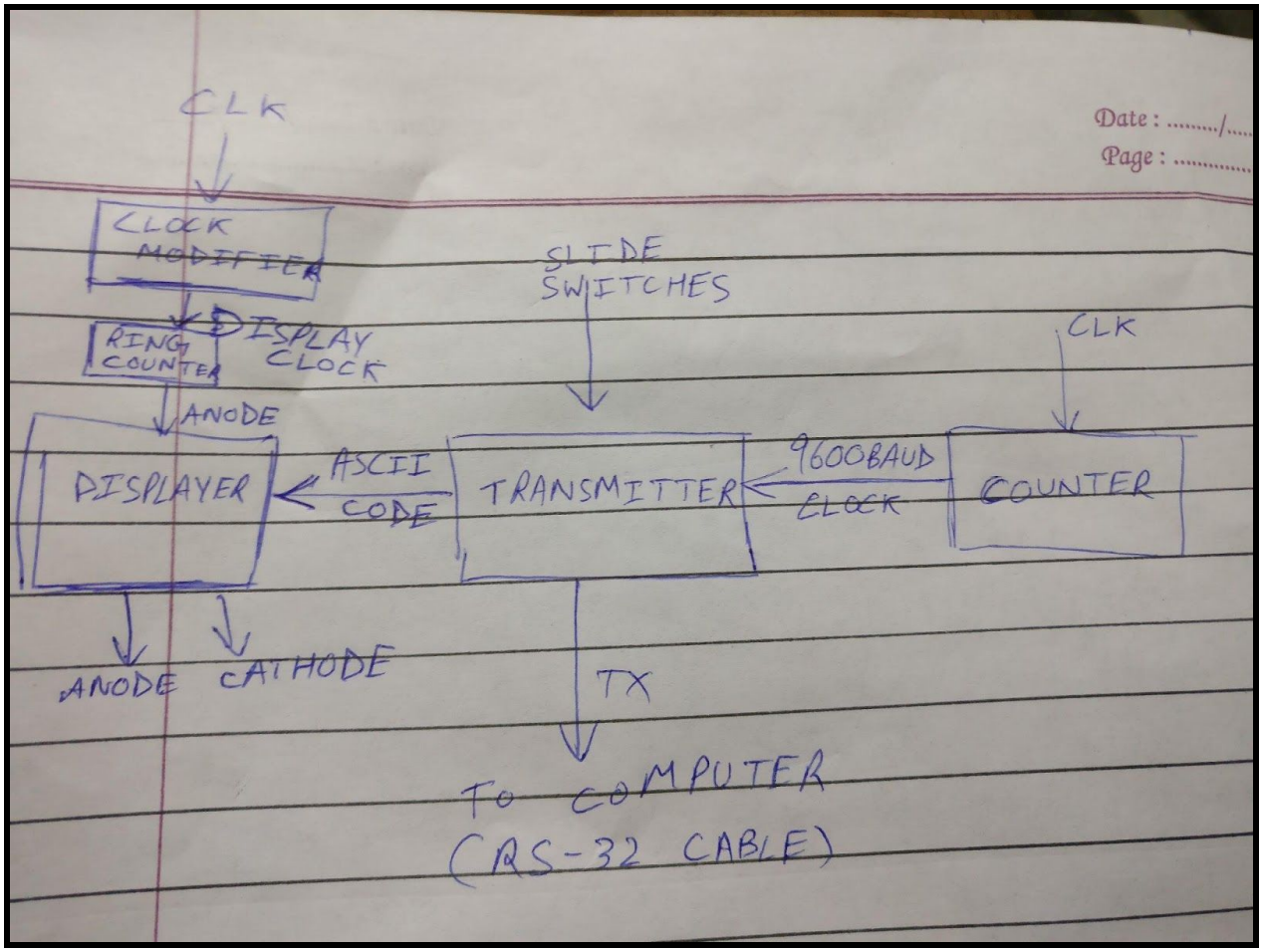
Overall Approach

To transmit an ASCII character we would need 8 bits at a time. To establish a protocol we would need some bit to know that the incoming of the character bits has started so this bit will be called a START BIT . Similarly we would need a STOP BIT to indicate an end of transmission and that Bit will be also be sent when there is no signal to be transmitted . Choices of start and stop can be 0 1 or 1 0 but , we'll go with the industry protocol standard of Start<=0 and Stop<=1 and there will be a stream of 1's when nothing is being sent, upon encountering the first 0 PC will start listening . The choice of whether to send start as 0 or 1 and to send MSB or LSB first will be dictated also by the protocol to receive programmed in the receiving end of PC.

Block Diagram

Transmitter has a counter of 10 bits which will select the bit that will be sent from the sequence (start + 8 bits + stop) and transmits the configuration of switches to cable. The display unit will display the

current value for the configuration of switches. To transmit the input, we'll hit push button and send .



Test and Demonstration

We will take input from the switches on FPGA , display the corresponding ASCII value on the Seven Segment Display and send the same value to PC's receiving terminal.

Input / Output

Input from switches (16 bits) will be encoded into start bit '0' then two packets of 8 bits each followed by stop bit '1'

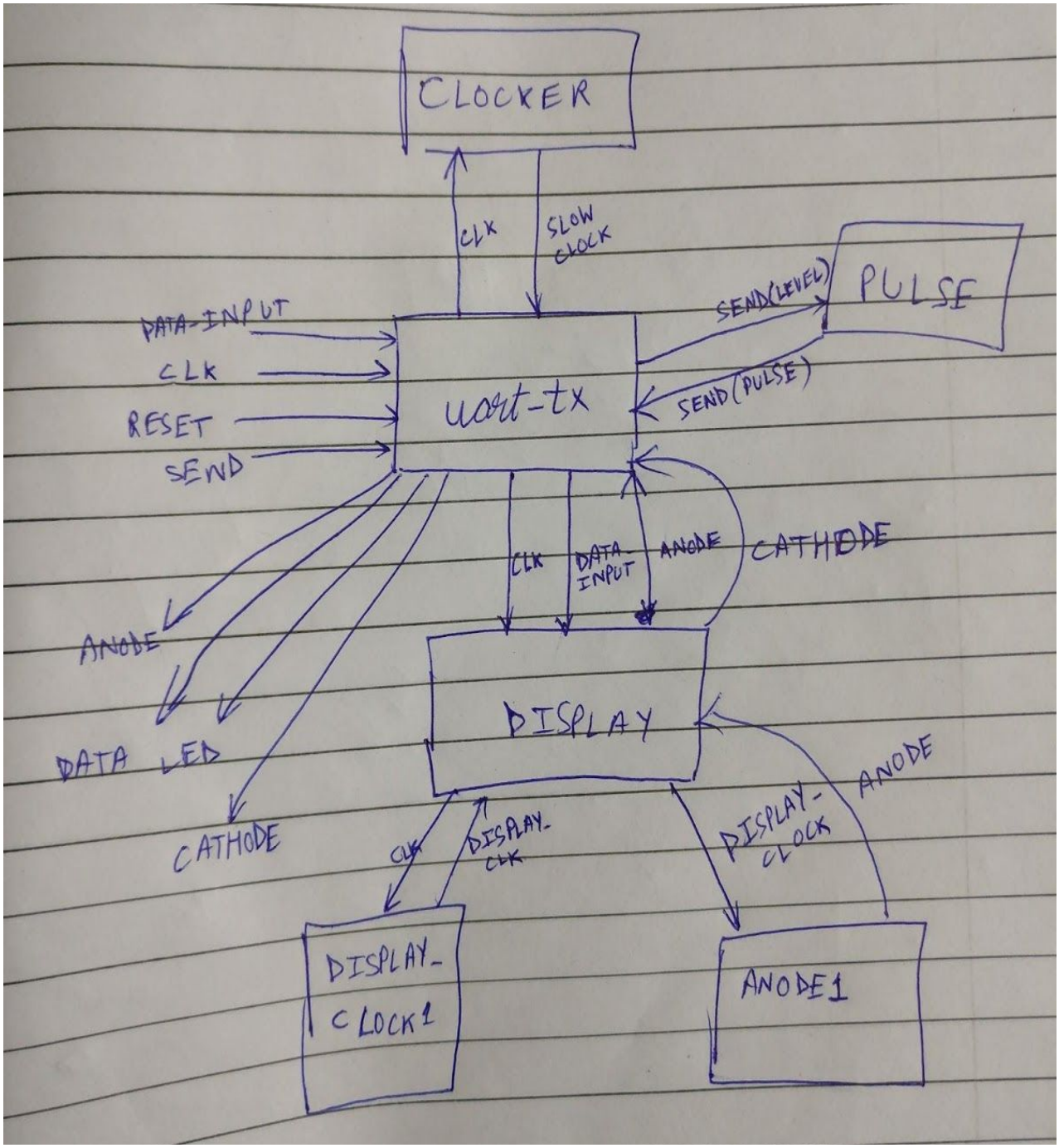
A push button will function as send button (which will send the configuration of switches on being pressed)

There will be a push button functioning as reset button

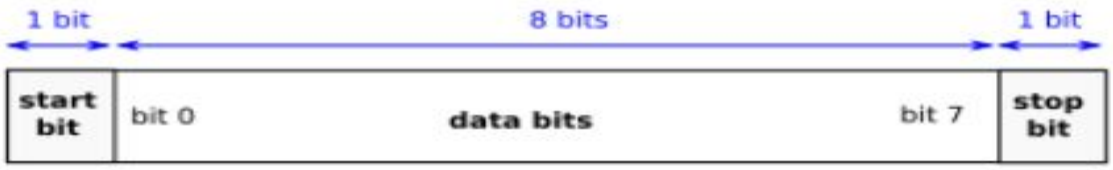
Switch configuration will be shown on LED's and optionally ASCII code will be shown on the SSD.

Output will be on terminal of PC , the two corresponding characters.

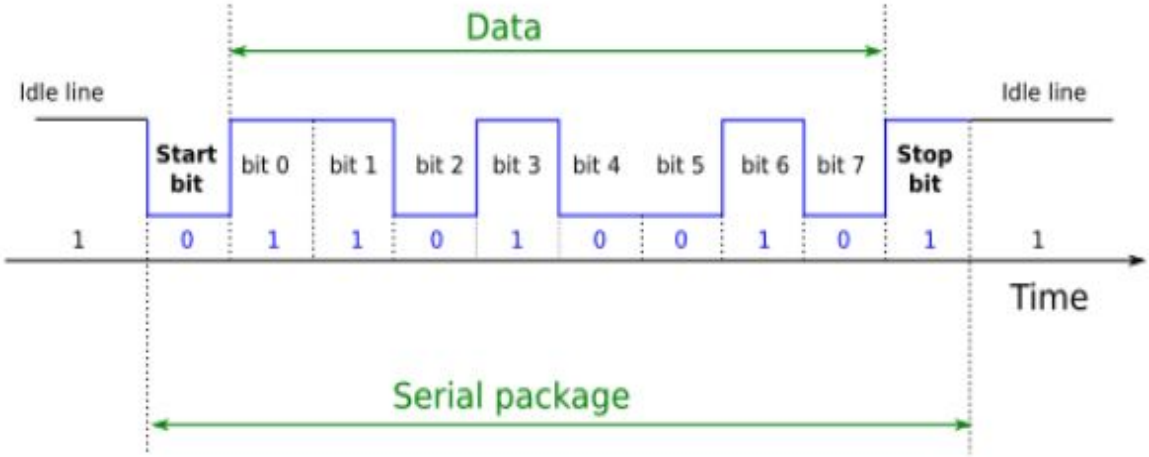
BLOCK DIAGRAM



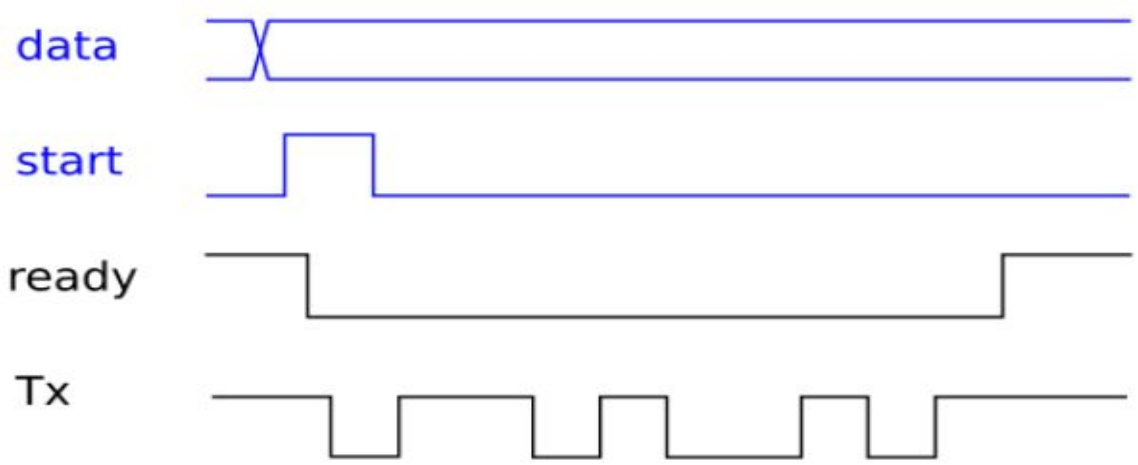
SERIAL PACKET



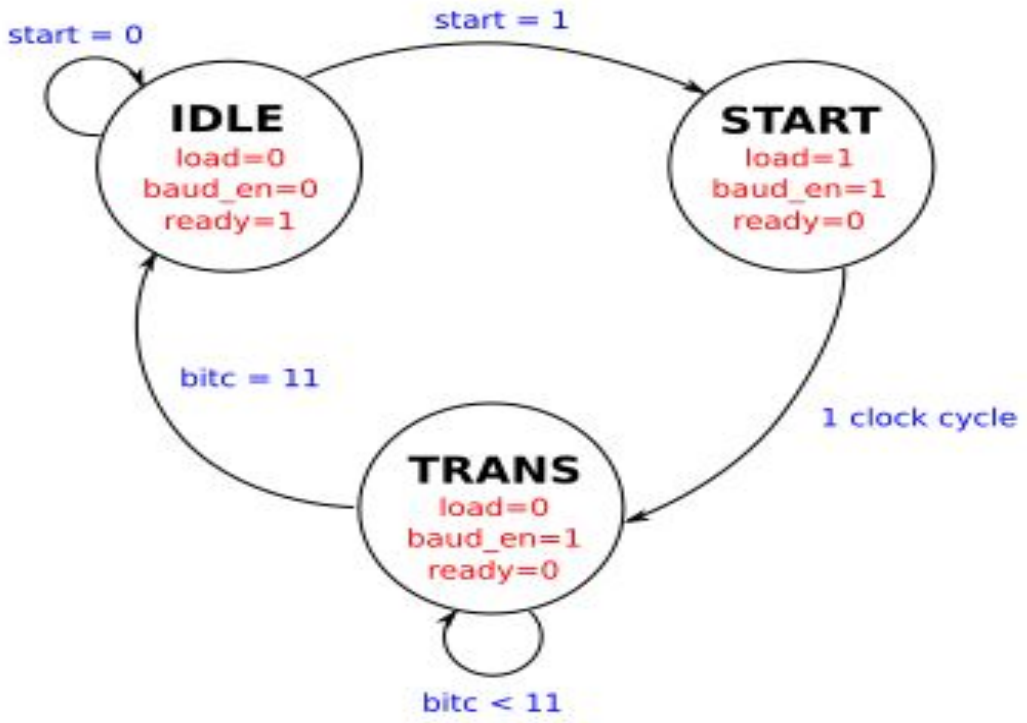
SERIAL TRANSMISSION



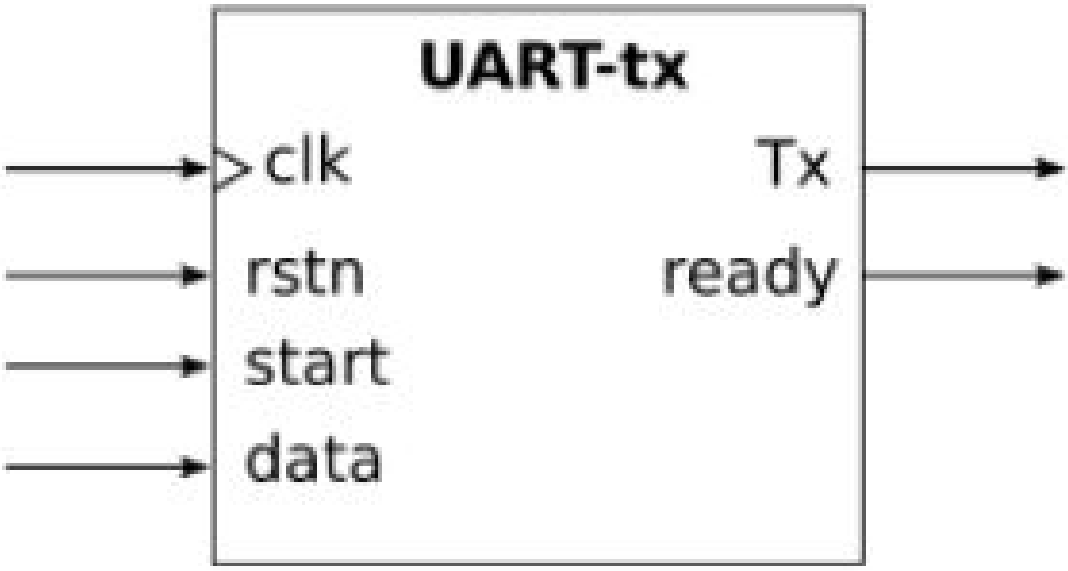
CHRONOGRAM



CONTROLLER FSM

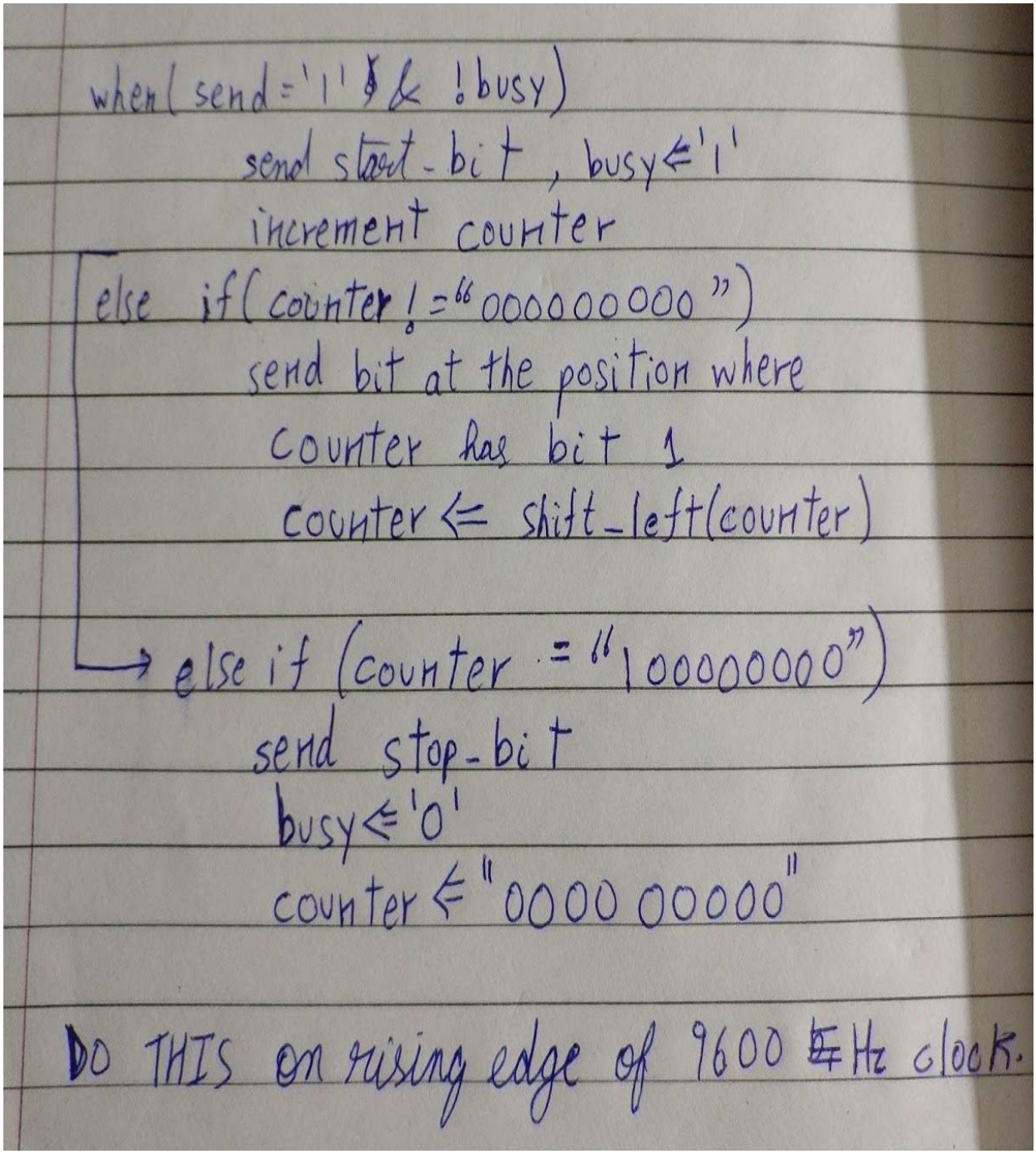


UART MODULE



PSEUDO-CODE

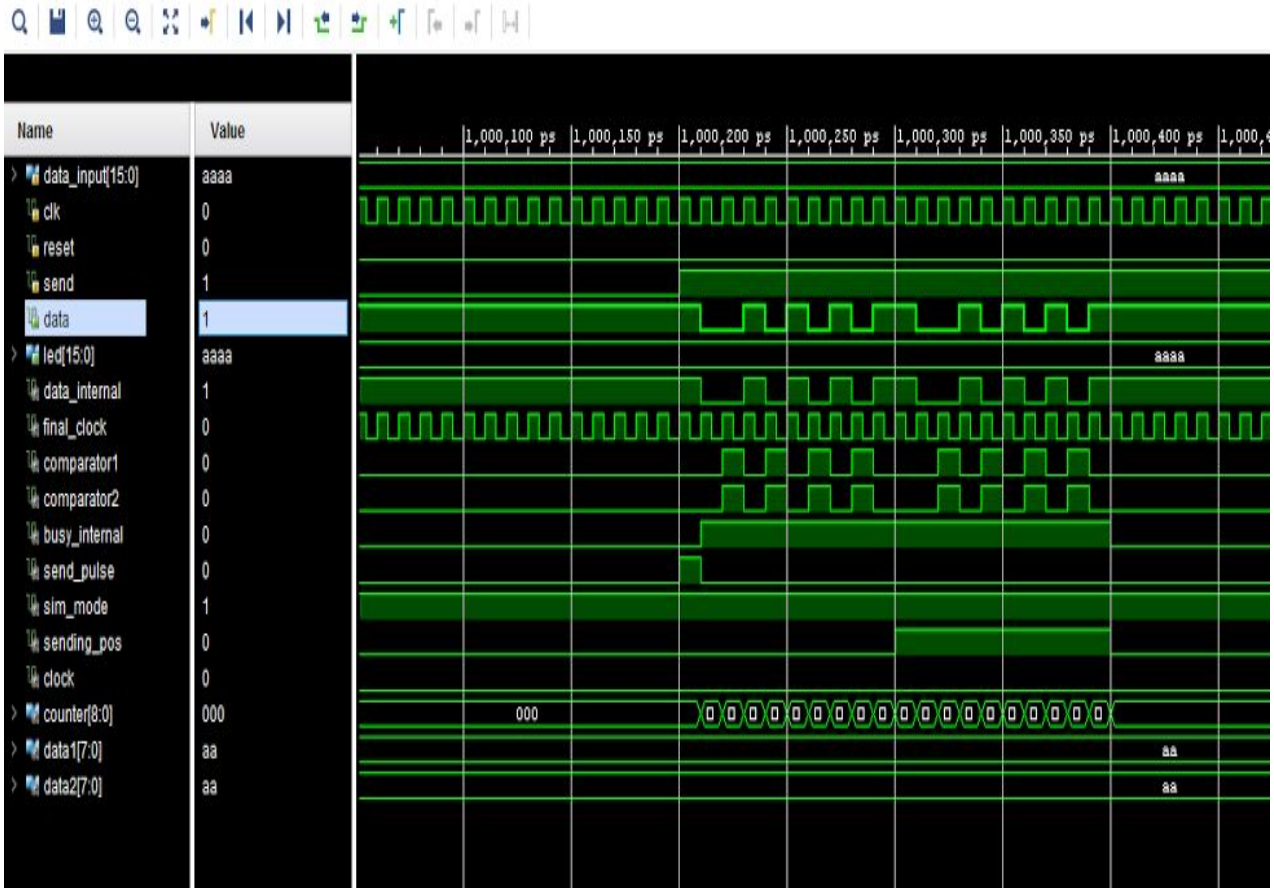
Our code is working excellently in the simulation mode , but we couldn't test it on the board due to some permissions restrictions on listening on the ports on the DHD lab computers. Otherwise the bitstream generated just fine.

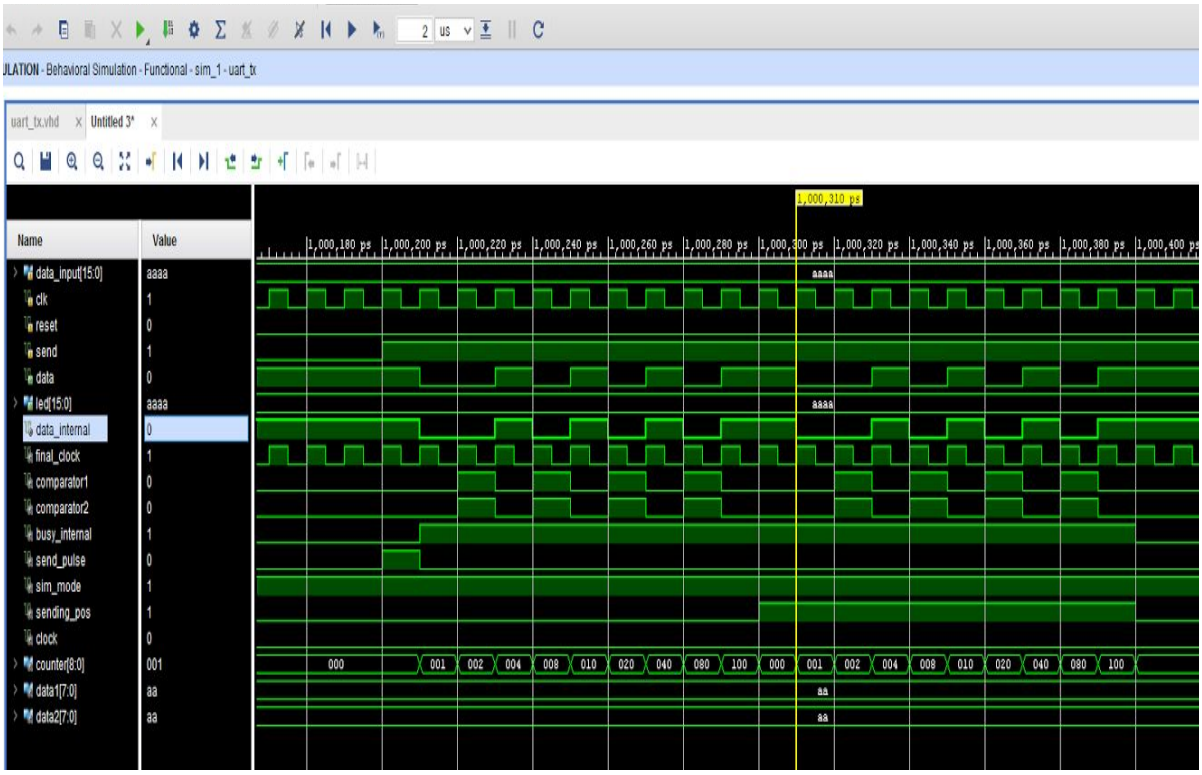


TESTING & SIMULATION

```
cs1160327@beas2:~$ cat /dev/tty
tty         tty21       tty35       tty49       tty62       ttyS17      ttyS30
tty0        tty22       tty36       tty5        tty63       ttyS18      ttyS31
tty1        tty23       tty37       tty50       tty7        ttyS19      ttyS4
tty10       tty24       tty38       tty51       tty8        ttyS2       ttyS5
tty11       tty25       tty39       tty52       tty9        ttyS20      ttyS6
tty12       tty26       tty4        tty53       ttyprintk   ttyS21      ttyS7
tty13       tty27       tty40       tty54       ttyS0       ttyS22      ttyS8
tty14       tty28       tty41       tty55       ttyS1       ttyS23      ttyS9
tty15       tty29       tty42       tty56       ttyS10      ttyS24      ttyUSB1
tty16       tty3        tty43       tty57       ttyS11      ttyS25
tty17       tty30       tty44       tty58       ttyS12      ttyS26
tty18       tty31       tty45       tty59       ttyS13      ttyS27
tty19       tty32       tty46       tty6        ttyS14      ttyS28
tty2        tty33       tty47       tty60       ttyS15      ttyS29
tty20       tty34       tty48       tty61       ttyS16      ttyS3
cs1160327@beas2:~$ cat /dev/ttyUSB1
```

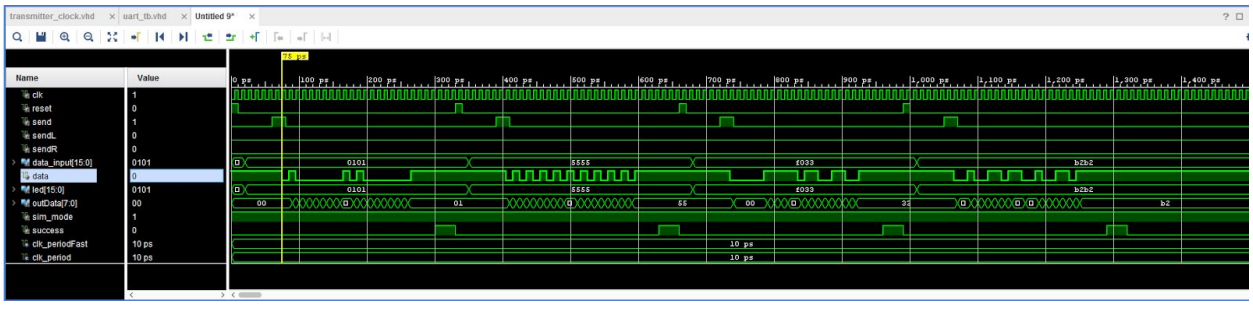
This cat/dev/ttyUSB1(from FPGA) will enable us to listen on that port , given if it's permissible. Other tty's are the other ports we can listen on.

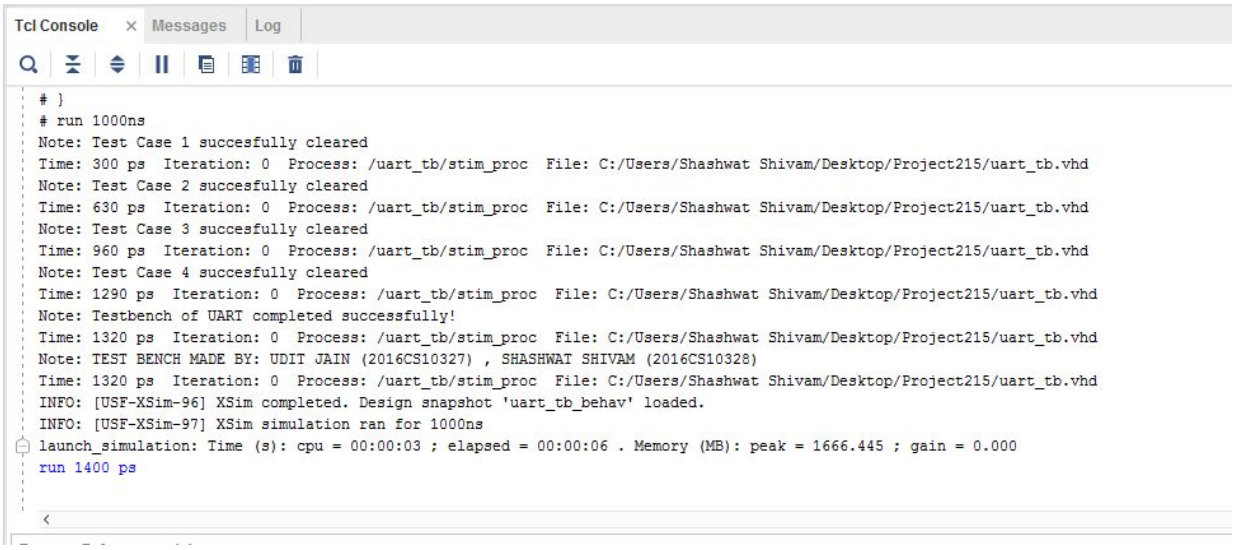




TESTBENCH TESTING OF MAIN MODULE

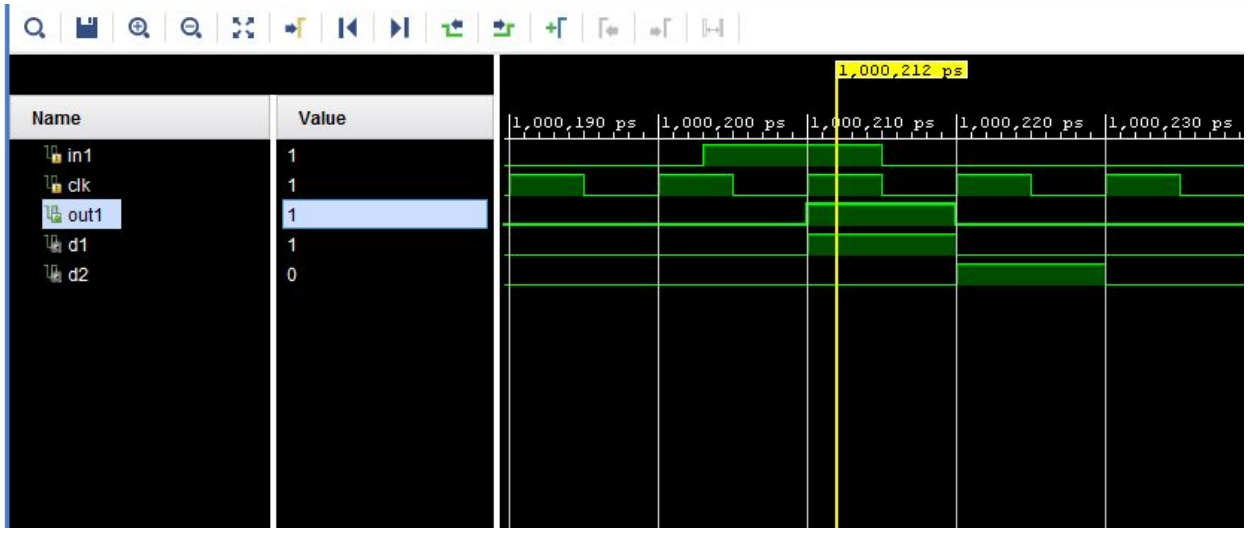
The main module was tested using a testbench run in simulation. The simulation was designed to send a clock pulse for 3-4 clock cycles. After sending a pulse start bit (=’0’) was tested and consequently the bits of the sent data (data signal was set prior to the test case and then wait was applied for a few clock cycles) was checked in data signal in each cycle and also in the reverse order i.e. the LSB (Least Significant Bit) first to the MSB (Most Significant Bit). After this the data was checked for stop bit (=’1’). After this the signal was tested to give ‘1’ signal if sent is not 1 and no data is being sent. Many random combinations of bits was checked for consistency. The test bench file has been attached along with the report. A few images of the testing are as shown below :-





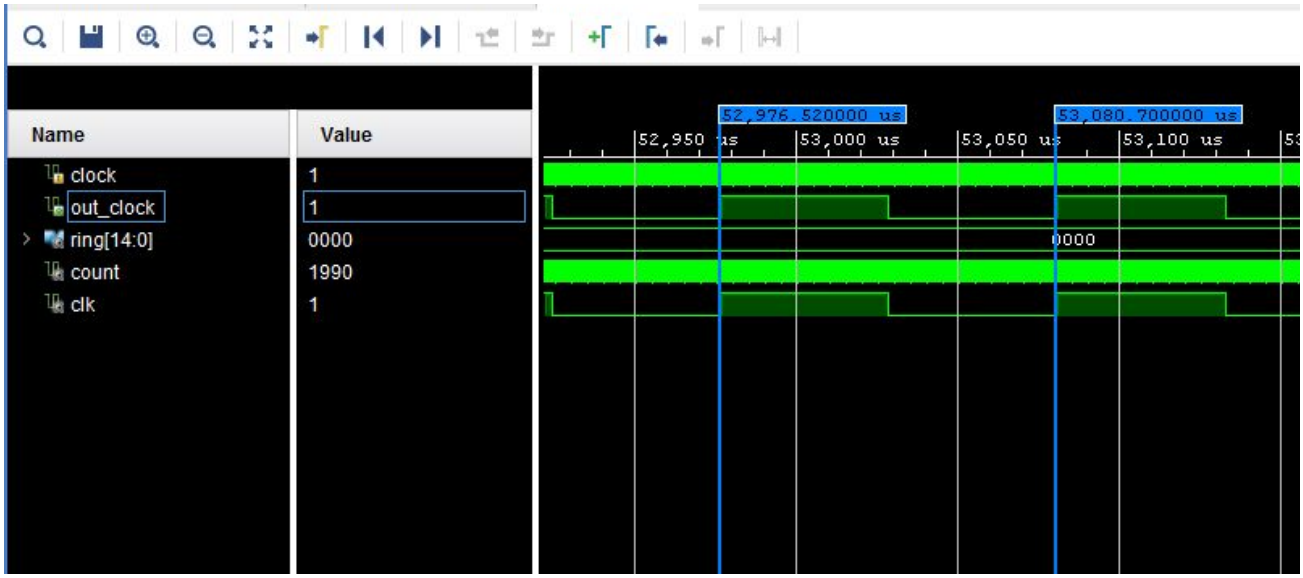
UNIT TESTING USING SIMULATION

The subunits were tested separately using simulation. The images are as follows :-

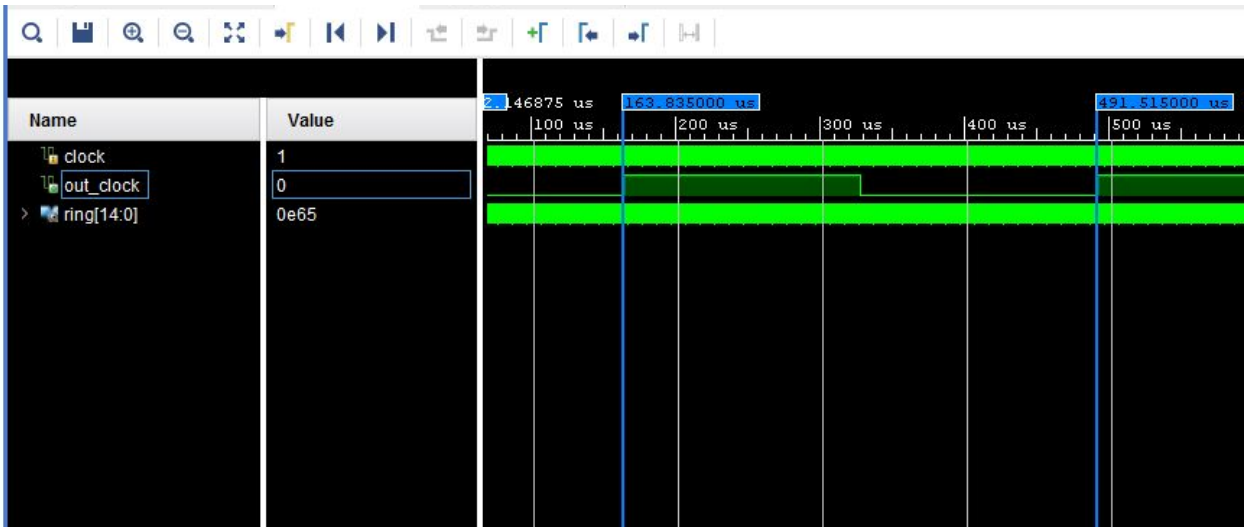


The above waveform is for the **level to pulse converter**. 'In1' is the level input signal which is converted to 'out1' pulse.

The above waveform is for the **ring counter**. The display clock (clock modified to use on seven segment display) is the input. The output are the corresponding anode values which shift the 0 to left circularly on every clock cycle. The outputs are of the form :- ‘1110’ -> ‘1101’ -> ‘1011’ and so on.



The above waveform is of the **transmission clock modifier**. On running the simulation clock at 100MHz (time period 10ns) we get a modified clock with time period (53080.70 us - 52976.52 us =) 104.18 us which is almost equal 9600 Hz (9598.77 Hz). This modified clock will thus help us to send a data signal at 9600 bits per second (9600 bauds). It works using a counter to count circularly upto 5208 on every clock edge.



The above waveform is of the **display clock modifier** (for SSD clock). Its design is similar to the transmission clock modifier and suitable output is created similarly.

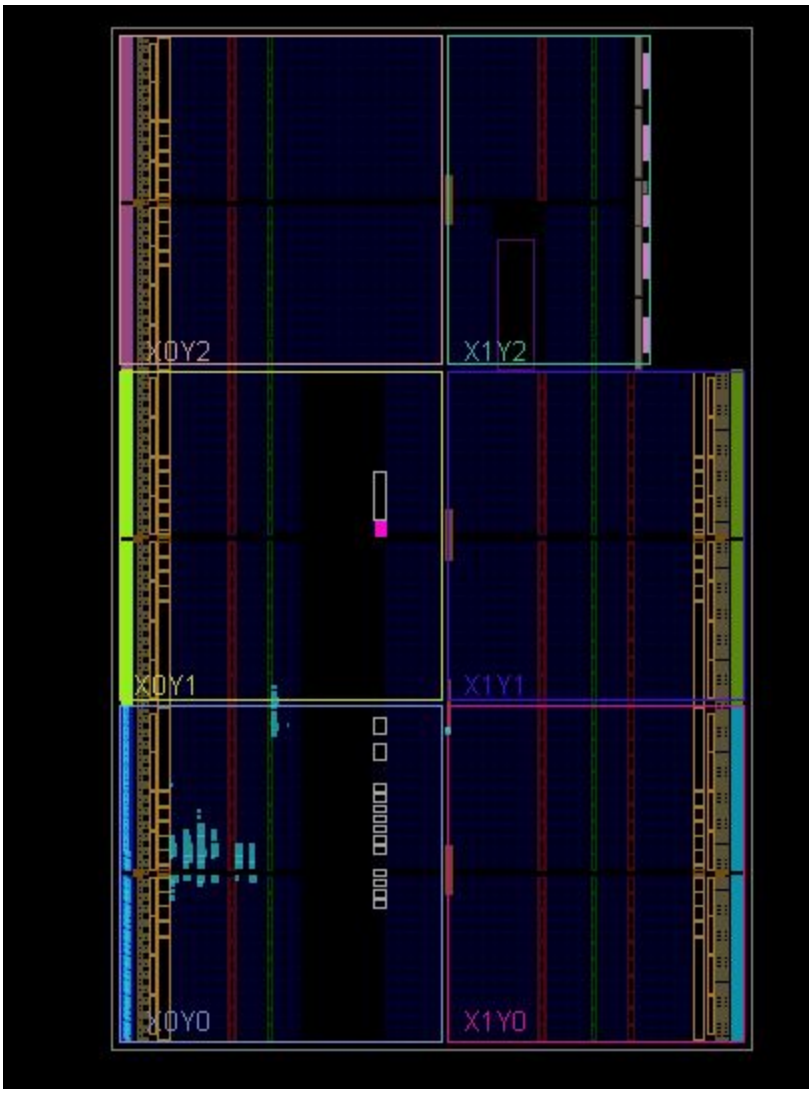
TESTING ON BOARD

The code was uploaded onto an FPGA and tested using Putty and debugged. The errors encountered on doing so were:-

1. Initially no output was visible. No transmission light was also blinking on the FPGA. The reason of this error was found that the pulse of the send button signal was too short to be caught within a clock cycle. This was then corrected.
2. Next we encountered error relating to garbage output on the terminal. On observing the ascii values of the characters output on the terminal it was found that the baud rate was not in sync. The baud rate being half of 9600 each bit was being received twice by the computer terminal. This was then corrected by changing the counter value in the transmission clock modifier entity. After this the output was as expected.

The images of the output on Putty will be put up in next report.

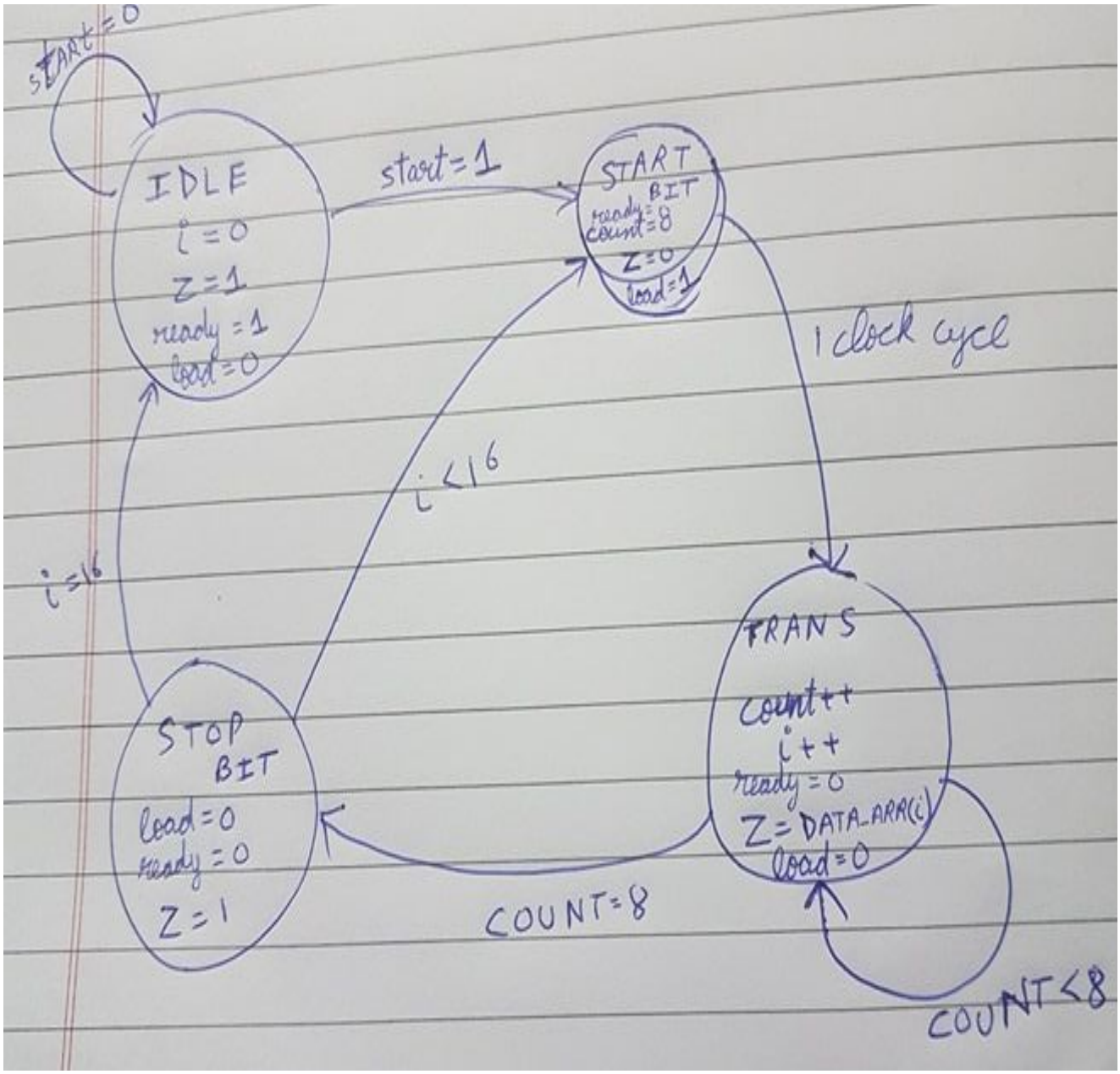
The image of the implemented design is as follows:-



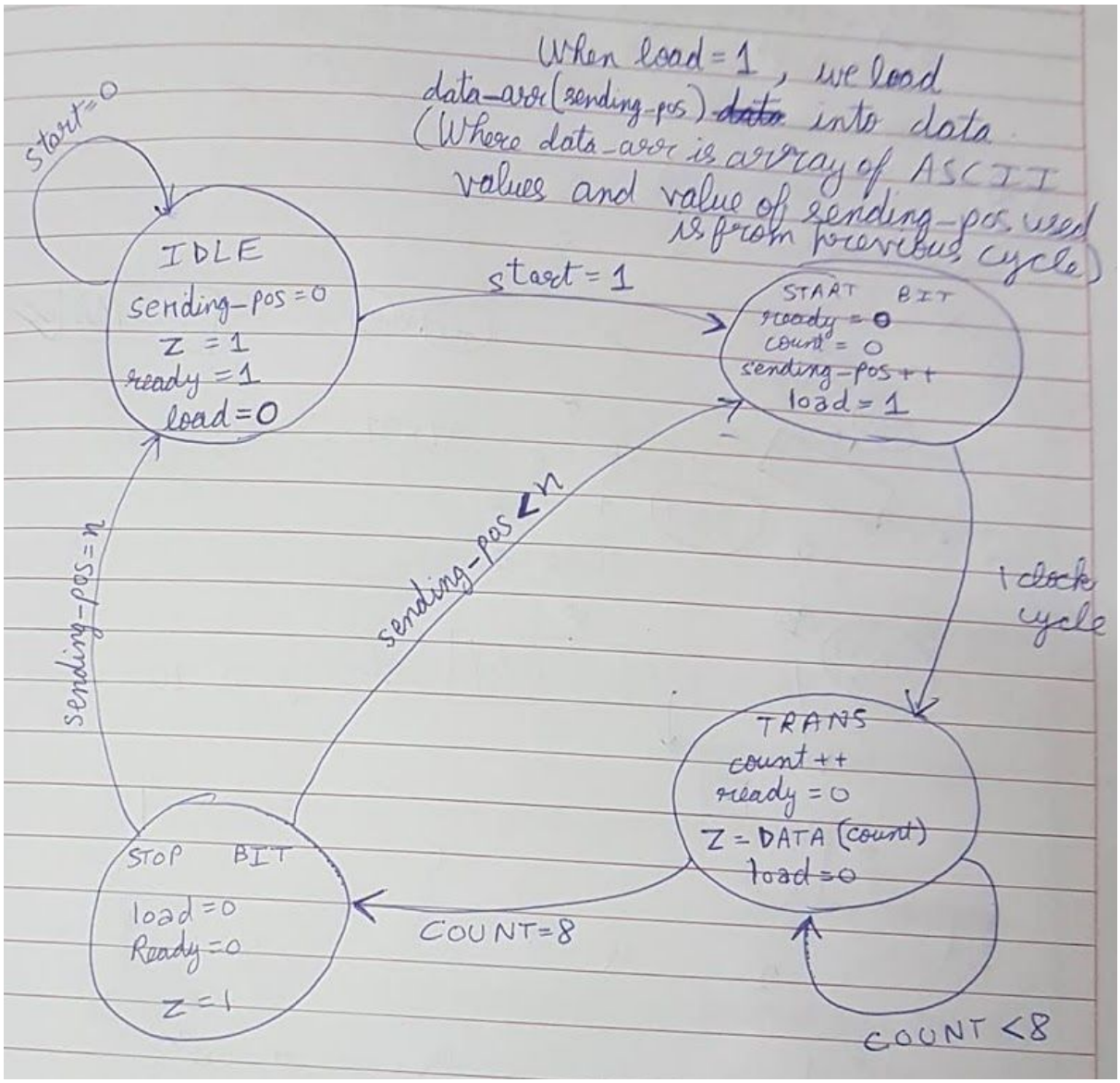
MODIFICATIONS AND EXTRA FEATURES

1. The 16 switches were used to send bits as pairs (two 8 bit ASCII codes.)
2. The extra buttons on the FPGA were used to program saved messages or images to be displayed on pressing them.
3. The SSD will be used to display the ASCII code that has been set on the switches to aid the user in setting ASCII codes faster.

The modified state diagram for sending 2 ASCII codes back to back is as follows:-



This state diagram is then generalized to send ‘n’ ASCII values to send pre- saved long messages. The image is as follows:-



CONCLUSION

Overall, we have successfully tested on simulation, with and without testbench for timing and correctness of the model, we have also done extensive testing on board. Additionally we are showing custom text and graphics on push of buttons and the relevant pictures of the output terminal and board will be uploaded in the final report.

Udit Jain - 2016CS10327 *Shashwat Shivam - 2016CS10328*