FPGA – based Embedded System Design

LAB #1

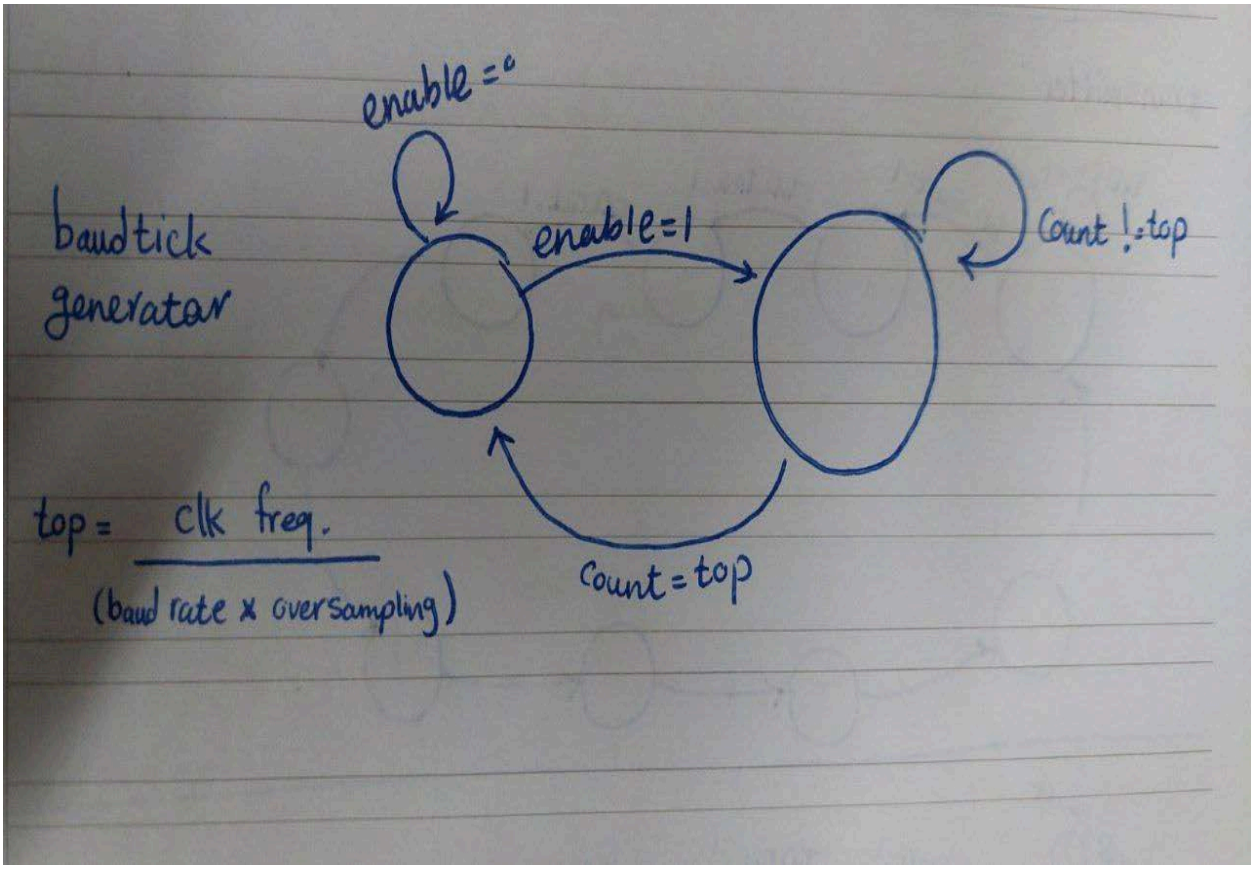
Group Members

Mohammad Taghizadeh Givari	810198373
Zeinab Saeedi	810198411
Amin Aroufzad	810198538

Contents

UART Implementation

Baud Tick Generator Implementation



Whenever Baud_Tick_enable is activated, Baud Tick Generator simply counts up to {clock frequency / (Baud Rate × over sampling rate)}, to make the Baud Tick frequency {Baud Rate × over sampling rate}, then it asserts the output just for 1 clock. At the end it repeats this countlessly until Baud_Tick_enable is deactivated.

D:/modelsim_ase/Projects/BaudTickGen.v

File Edit View Tools Bookmarks Window Help

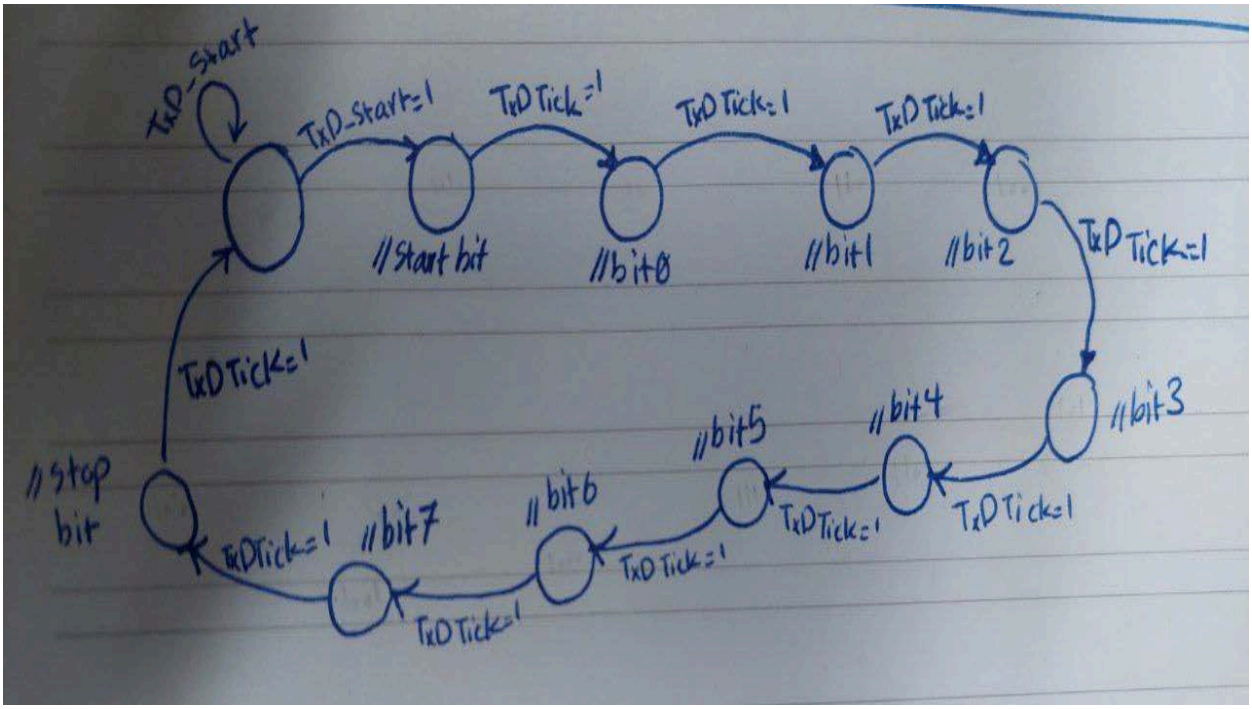
D:/modelsim_ase/Projects/BaudTickGen.v - Default

```
1  module BaudTickGen(
2      input clk, enable,
3      output reg tick  // generate a tick at the specified baud rate * oversampling
4  );
5  parameter ClkFrequency = 50000000;
6  parameter Baud = 115200;
7  parameter Oversampling = 1;
8  reg [24:0] count = 0;
9  reg [24:0] top;
10
11 reg BaudTickGen_state = 0;
12
13 always @ (posedge clk) begin
14         top =( (ClkFrequency / Baud) /Oversampling);
15         tick = 1'b0;
16         case(BaudTickGen_state)
17             1'b0: if(enable) BaudTickGen_state <= 1'b1;
18             1'b1:
19                 if(count == top)
20                     {tick, count, BaudTickGen_state} <= {1'b1, 26'b0};
21                 else count = count + 1'b1;
22
23             default: BaudTickGen_state <= 1'b0;
24         endcase
25
26 end
27
28 endmodule
29
30
```

Ln: 4 Col: 0

UART Implementation

Asynchronous Transmitter Implementation



First we use the Baud Tick Generator to create Transmitter Tick(TxD_Tick) signal. Whenever TxD_Start is activated, transmission starts:

1. start bit, which is 0, is sent.

2. In the next 8 Transmitter's clocks, data is sent respectively. To specify the nth bit we just shift the data n times, The LSB of the data is the nth bit of our data which is about to be sent.

3. At the end stop bit, which is 1, is sent.

D:/modelsim_ase/Projects/async_transmitter.v

File Edit View Tools Bookmarks Window Help

D:/modelsim_ase/Projects/async_transmitter.v - Default

```
module async_transmitter(
    input clk,
    input TxD_start,
    input [7:0] TxD_data,
    output TxD,
    output TxD_busy
);

// Assert TxD_start for (at least) one clock cycle to start transmission of TxD_data
// TxD_data is latched so that it doesn't have to stay valid while it is being sent
parameter ClkFrequency = 50000000;
parameter Baud = 115200;
parameter Oversampling = 1;

wire TxDTick;
BaudTickGen #(ClkFrequency, Baud, Oversampling) tickgen(.clk(clk), .enable(TxD_busy), .tick(TxDTick));

reg [3:0] TxD_state = 0;
wire TxD_ready = (TxD_state==0);
assign TxD_busy = ~TxD_ready;

reg [7:0] TxD_shift = 0;
always @(posedge clk)
begin
    if(TxD_ready & TxD_start)
        TxD_shift <= TxD_data;
    else
    if((TxD_state>4'b0001) & TxDTick)
        TxD_shift <= (TxD_shift >> 1);

         case(TxD_state)
         4'b0000: if(TxD_start) TxD_state <= 4'b0001;
         4'b0001: if(TxDTick) TxD_state <= 4'b0010;  // start bit
         4'b0010: if(TxDTick) TxD_state <= 4'b0011;  // bit 0
         4'b0011: if(TxDTick) TxD_state <= 4'b0100;  // bit 1
         4'b0100: if(TxDTick) TxD_state <= 4'b0101;  // bit 2
         4'b0101: if(TxDTick) TxD_state <= 4'b0110;  // bit 3
         4'b0110: if(TxDTick) TxD_state <= 4'b0111;  // bit 4
```

Ln: 3 Col: 0

D:/modelsim_ase/Projects/async_transmitter.v

File Edit View Tools Bookmarks Window Help

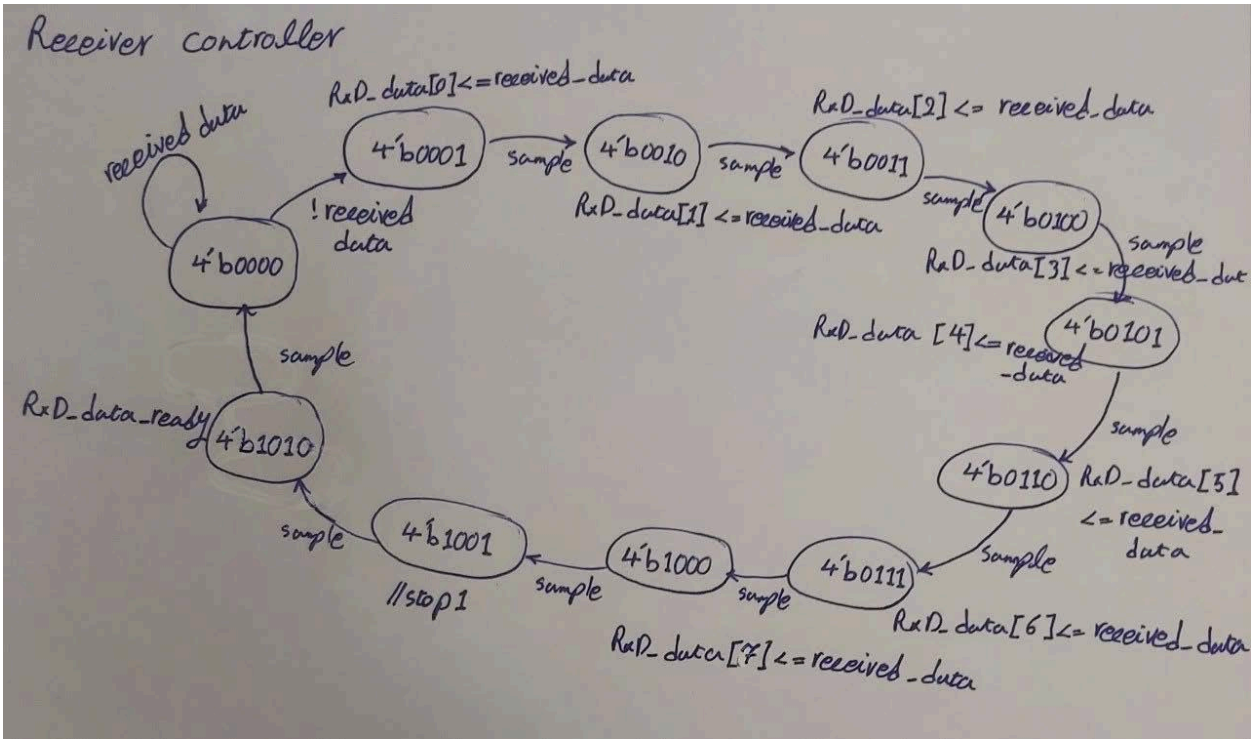
D:/modelsim_ase/Projects/async_transmitter.v - Default

```
parameter Oversampling = 1;

wire TxDTick;
BaudTickGen #(ClkFrequency, Baud, Oversampling) tickgen(.clk(clk), .enable(TxD_busy), .tick(TxDTick));

reg [3:0] TxD_state = 0;
wire TxD_ready = (TxD_state==0);
assign TxD_busy = ~TxD_ready;

reg [7:0] TxD_shift = 0;
always @(posedge clk)
begin
    if(TxD_ready & TxD_start)
        TxD_shift <= TxD_data;
    else
    if((TxD_state>4'b0001) & TxDTick)
        TxD_shift <= (TxD_shift >> 1);

         case(TxD_state)
         4'b0000: if(TxD_start) TxD_state <= 4'b0001;
         4'b0001: if(TxDTick) TxD_state <= 4'b0010;  // start bit
         4'b0010: if(TxDTick) TxD_state <= 4'b0011;  // bit 0
         4'b0011: if(TxDTick) TxD_state <= 4'b0100;  // bit 1
         4'b0100: if(TxDTick) TxD_state <= 4'b0101;  // bit 2
         4'b0101: if(TxDTick) TxD_state <= 4'b0110;  // bit 3
         4'b0110: if(TxDTick) TxD_state <= 4'b0111;  // bit 4
         4'b0111: if(TxDTick) TxD_state <= 4'b1000;  // bit 5
         4'b1000: if(TxDTick) TxD_state <= 4'b1001;  // bit 6
         4'b1001: if(TxDTick) TxD_state <= 4'b1010;  // bit 7
         4'b1010: if(TxDTick) TxD_state <= 4'b0000;  // stop1
         default: if(TxDTick) TxD_state <= 4'b0000;
     endcase
end

assign TxD = (TxD_state == 4'b0000) | ((TxD_state > 4'b0001) & TxD_shift[0]) | (TxD_state == 4'b1010);
endmodule
```

Ln: 3 Col: 0

UART Implementation

Asynchronous Receiver Implementation



First we use the Baud Tick Generator to create Receiver Tick(RxD_Tick) signal. To oversample, oversampling parameter is set to 4 so that data is sampled 4 times between each 2 Transmitter's clocks. The third sample is considered as the received data.

1. start bit, which is 0, is received.

2. In the next 8 Transmitter's clock, data is received respectively. In fact the third sampled data between each two Transmitter's clocks is considered as the received data.

3. At the end stop bit, which is 1, is received and RxD_data_ready is activated.

D:/modelsim_ase/Projects/async_receiver.v (/FIR_Module_Test_Bench/FIR/DataPath/uart/Receiver)

File Edit View Tools Bookmarks Window Help

D:/modelsim_ase/Projects/async_receiver.v (/FIR_Module_Test_Bench/FIR/DataPath/uart/Receiver) - Default

Ln# 1208697 ns

```
1  module async_receiver(
2      input clk,
3      input RxD,
4      output reg RxD_data_ready = 0,
5      output reg [7:0] RxD_data = 0  // data received, valid only (for one clock cycle) when RxD_data_ready is asserted
6  );
7
8  parameter ClkFrequency = 50000000;
9  parameter Baud = 115200;
10 parameter Oversampling = 4;     // needs to be a power of 2
11
12 wire RxDTick;
13 BaudTickGen #(ClkFrequency, Baud, Oversampling) tickgen(.clk(clk), .enable(1), .tick(RxDTick));
14
15 reg TxDTick, received_data;
16 reg [3:0] RxD_state = 0;
17 reg [1:0] over_sampling_state, sampled_data = 0;
18
19 always @(posedge TxDTick)
20 begin
21         RxD_data_ready = 0;
22         case(RxD_state)
23         4'b0000: if(~received_data) RxD_state <= 4'b0001;  // start bit
24         4'b0001: begin RxD_state <= 4'b0010; RxD_data[0] <= received_data; end // bit 0
25         4'b0010: begin RxD_state <= 4'b0011; RxD_data[1] <= received_data; end // bit 1
26         4'b0011: begin RxD_state <= 4'b0100; RxD_data[2] <= received_data; end // bit 2
27         4'b0100: begin RxD_state <= 4'b0101; RxD_data[3] <= received_data; end // bit 3
28         4'b0101: begin RxD_state <= 4'b0110; RxD_data[4] <= received_data; end // bit 4
29         4'b0110: begin RxD_state <= 4'b0111; RxD_data[5] <= received_data; end // bit 5
30         4'b0111: begin RxD_state <= 4'b1000; RxD_data[6] <= received_data; end // bit 6
31         4'b1000: begin RxD_state <= 4'b1001; RxD_data[7] <= received_data; end // bit 7
32         4'b1001: RxD_state <= 4'b1010;  // stop1
33         4'b1010: begin RxD_state <= 4'b0000;  RxD_data_ready <= 1; end // ready
34         default: RxD_state <= 4'b0000;
35     endcase
36 end
37
38 always @(posedge clk)
```

Ln: 4 Col: 0

D:/modelsim_ase/Projects/async_receiver.v (/FIR_Module_Test_Bench/FIR/DataPath/uart/Receiver)

File Edit View Tools Bookmarks Window Help

D:/modelsim_ase/Projects/async_receiver.v (/FIR_Module_Test_Bench/FIR/DataPath/uart/Receiver) - Default

Ln# 1208697 ns

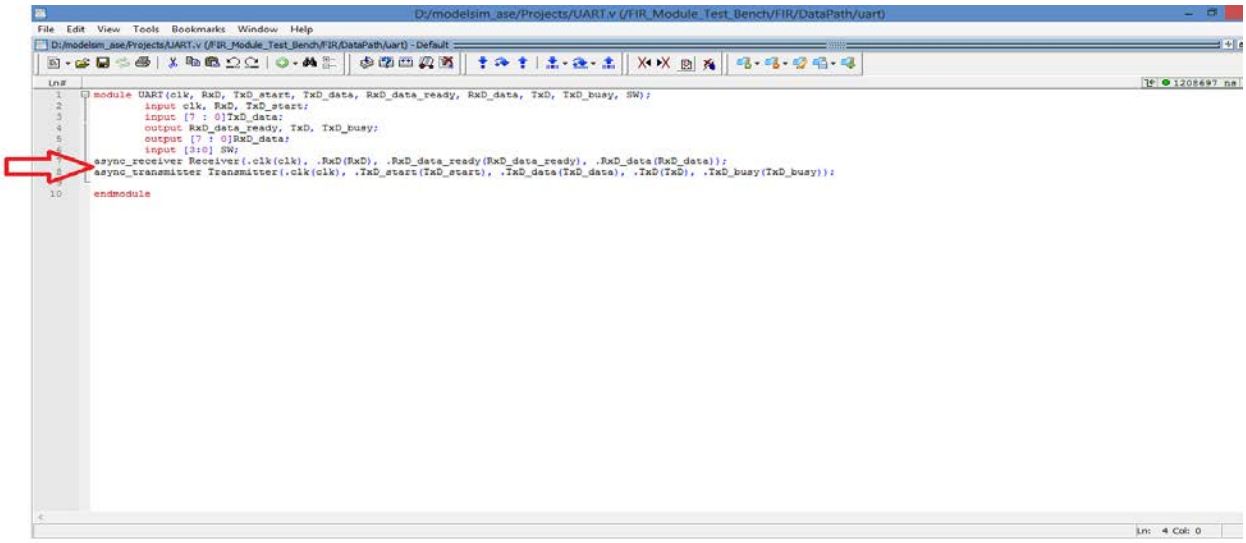
```
14
15 reg TxDTick, received_data;
16 reg [3:0] RxD_state = 0;
17 reg [1:0] over_sampling_state, sampled_data = 0;
18
19 always @(posedge TxDTick)
20 begin
21         RxD_data_ready = 0;
22         case(RxD_state)
23         4'b0000: if(~received_data) RxD_state <= 4'b0001;  // start bit
24         4'b0001: begin RxD_state <= 4'b0010; RxD_data[0] <= received_data; end // bit 0
25         4'b0010: begin RxD_state <= 4'b0011; RxD_data[1] <= received_data; end // bit 1
26         4'b0011: begin RxD_state <= 4'b0100; RxD_data[2] <= received_data; end // bit 2
27         4'b0100: begin RxD_state <= 4'b0101; RxD_data[3] <= received_data; end // bit 3
28         4'b0101: begin RxD_state <= 4'b0110; RxD_data[4] <= received_data; end // bit 4
29         4'b0110: begin RxD_state <= 4'b0111; RxD_data[5] <= received_data; end // bit 5
30         4'b0111: begin RxD_state <= 4'b1000; RxD_data[6] <= received_data; end // bit 6
31         4'b1000: begin RxD_state <= 4'b1001; RxD_data[7] <= received_data; end // bit 7
32         4'b1001: RxD_state <= 4'b1010;  // stop1
33         4'b1010: begin RxD_state <= 4'b0000;  RxD_data_ready <= 1; end // ready
34         default: RxD_state <= 4'b0000;
35     endcase
36 end
37
38 always @(posedge clk)
39 begin
40         TxDTick = 0;
41         case(over_sampling_state)
42         2'b00: if(RxDTick) over_sampling_state <= 2'b01;
43         2'b01: if(RxDTick) begin over_sampling_state <= 2'b10; end
44         2'b10: if(RxDTick) begin over_sampling_state <= 2'b11; sampled_data[1] <= RxD; end
45         2'b11: if(RxDTick) begin over_sampling_state <= 2'b00; TxDTick = 1; received_data = sampled_data[1]; end
46         default: if(RxDTick) over_sampling_state <= 2'b00;
47     endcase
48 end
49
50
51 endmodule
```

Ln: 4 Col: 0

UART Implementation

UART (Top Level) Implementation

We simply put Asynchronous Transmitter and Asynchronous Receiver together to make UART module:

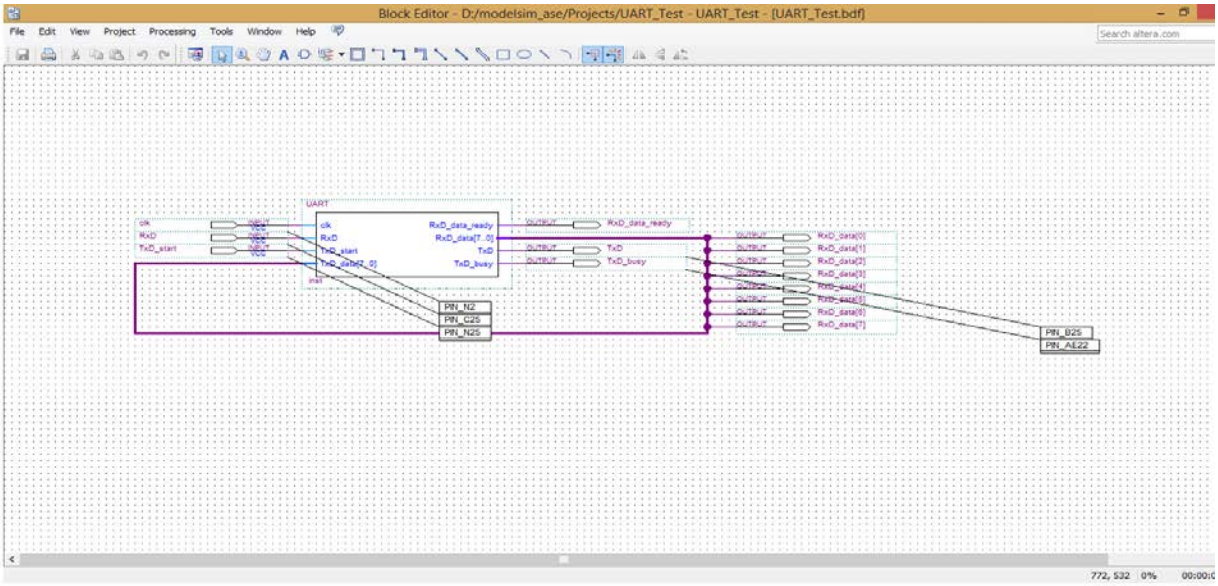


UART Implementation

UART (Top Level) Verification

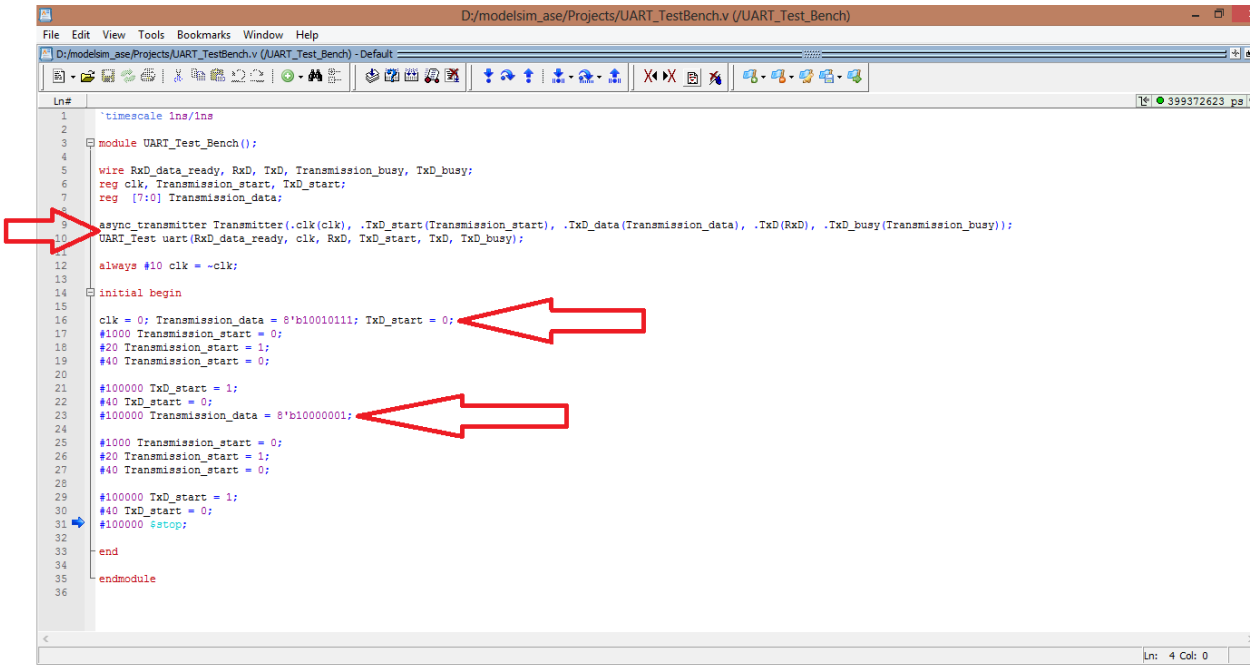
To test the UART module, the RxD_data port of UART, which is the received data, is connected to the TxD_data port, which specifies the data that is going to be transmitted by UART. So UART module sends all its received data thus it is expected that UART module echoes whatever it receives.

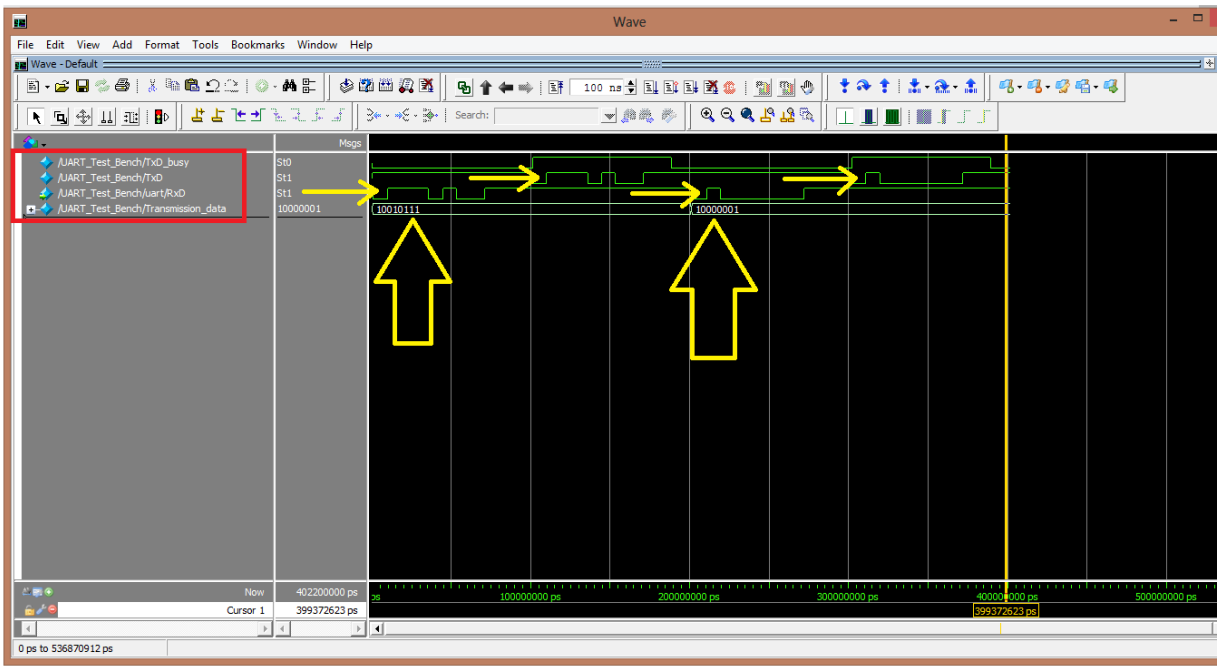
Quartus Synthesized UART Test



UART Verilog Test Bench

To test UART module, data is transferred to UART by the Asynchronous Transmitter. We send 2 different data and expect the UART module to echoes the same data through its TxD port.

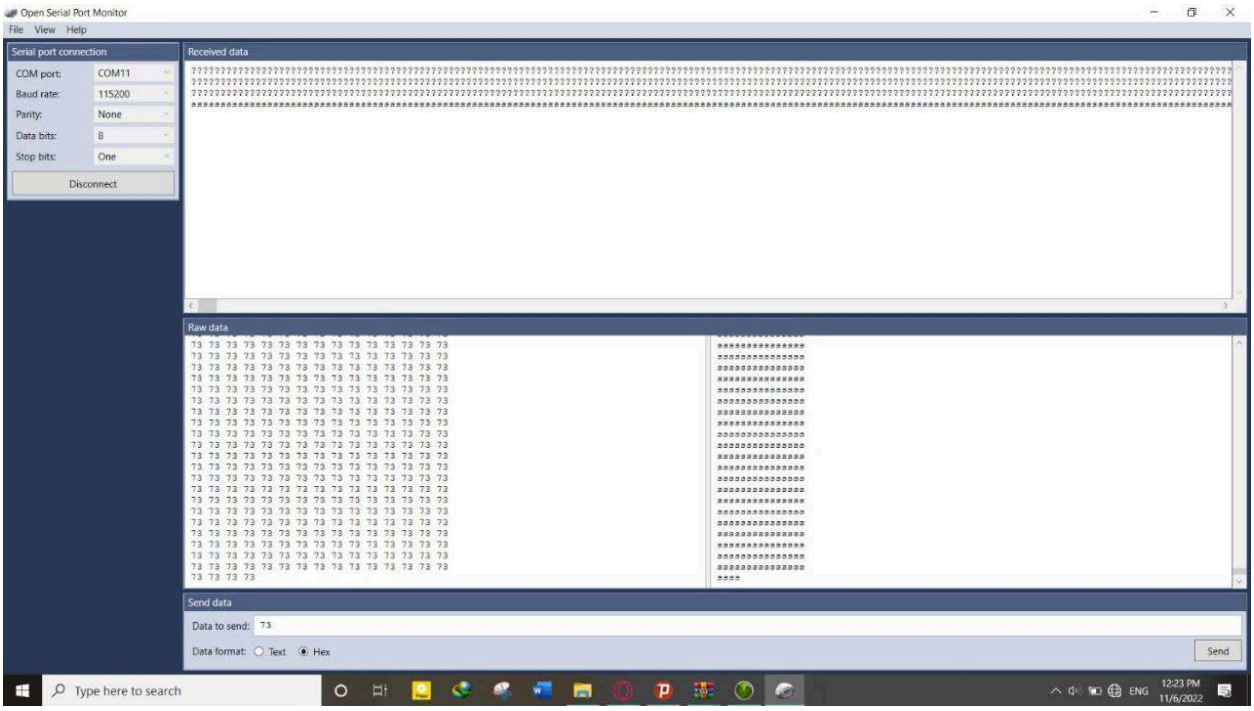


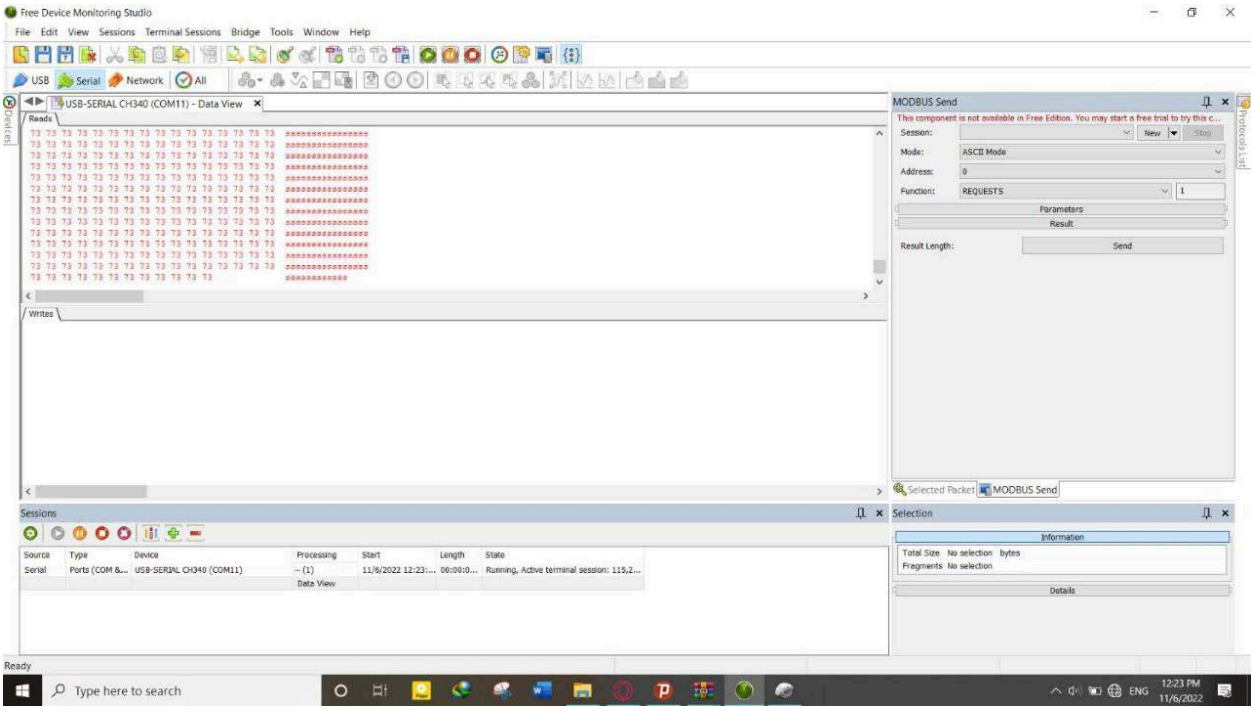


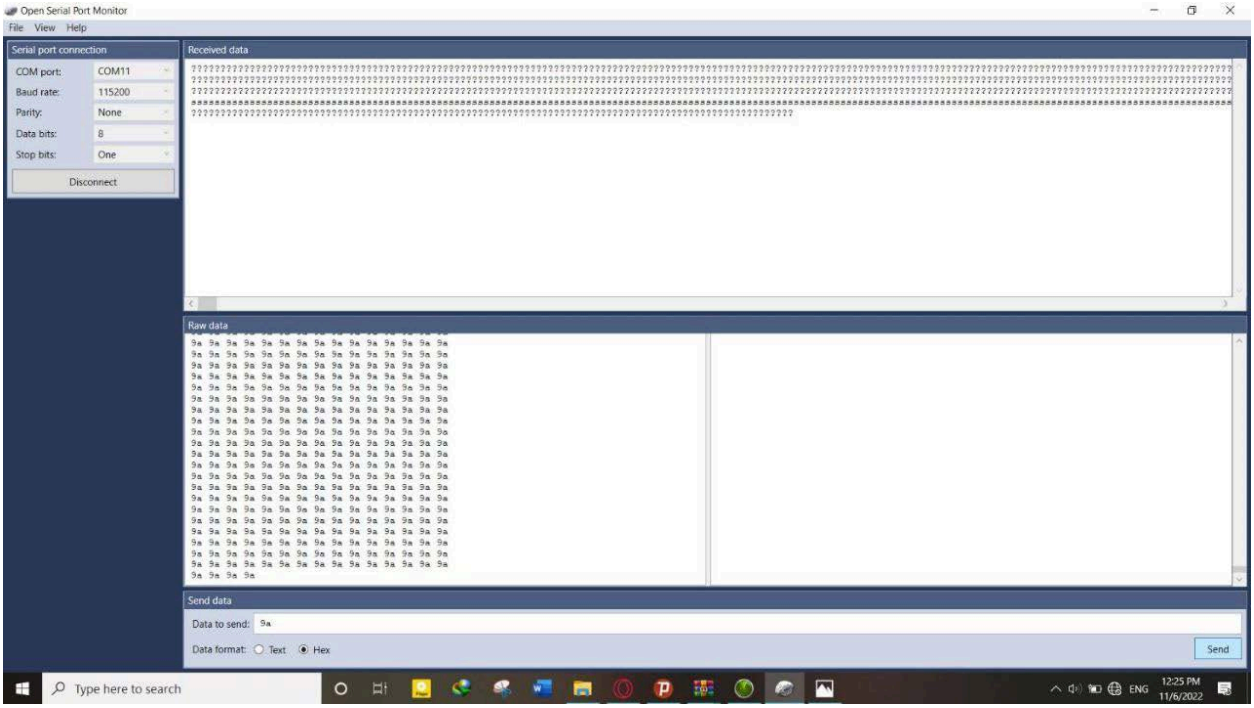
As you can see, Transmission_data is received in serially on RxD port. Then UART sends the same data serially through TxD port. This is true for both 2 different data. This indicates that UART module (including transmitter and receiver) works properly.

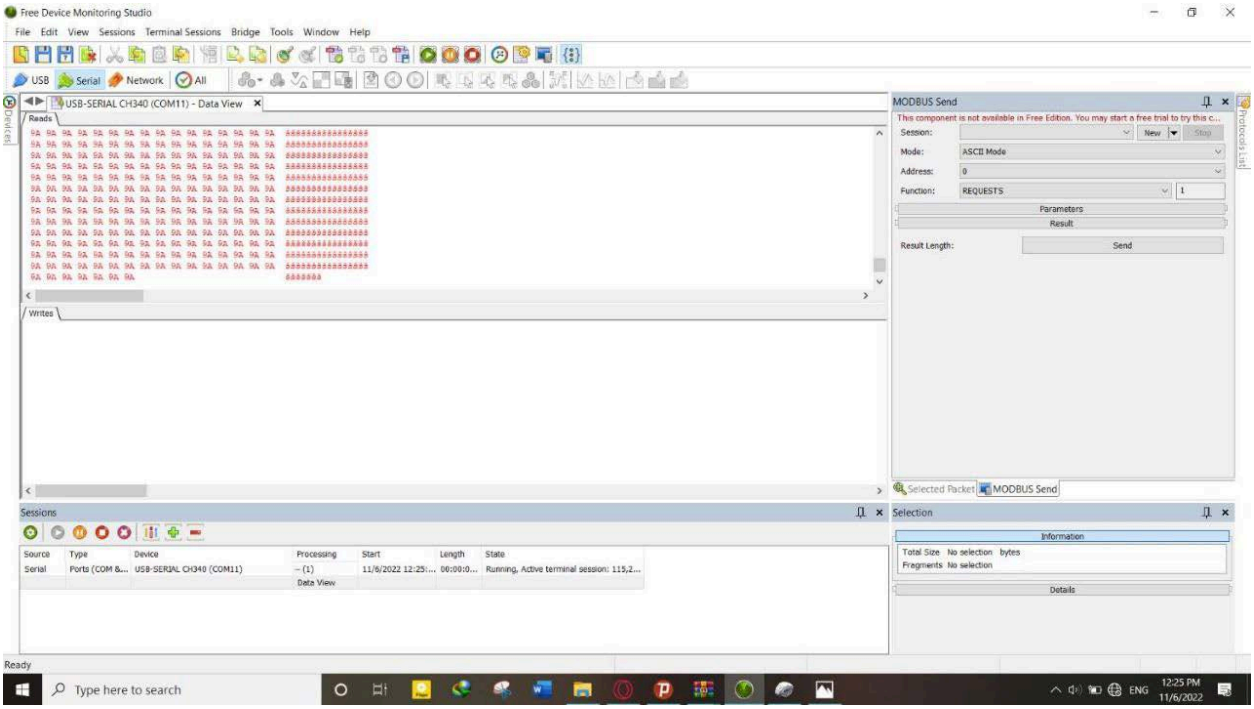
FPGA’s results

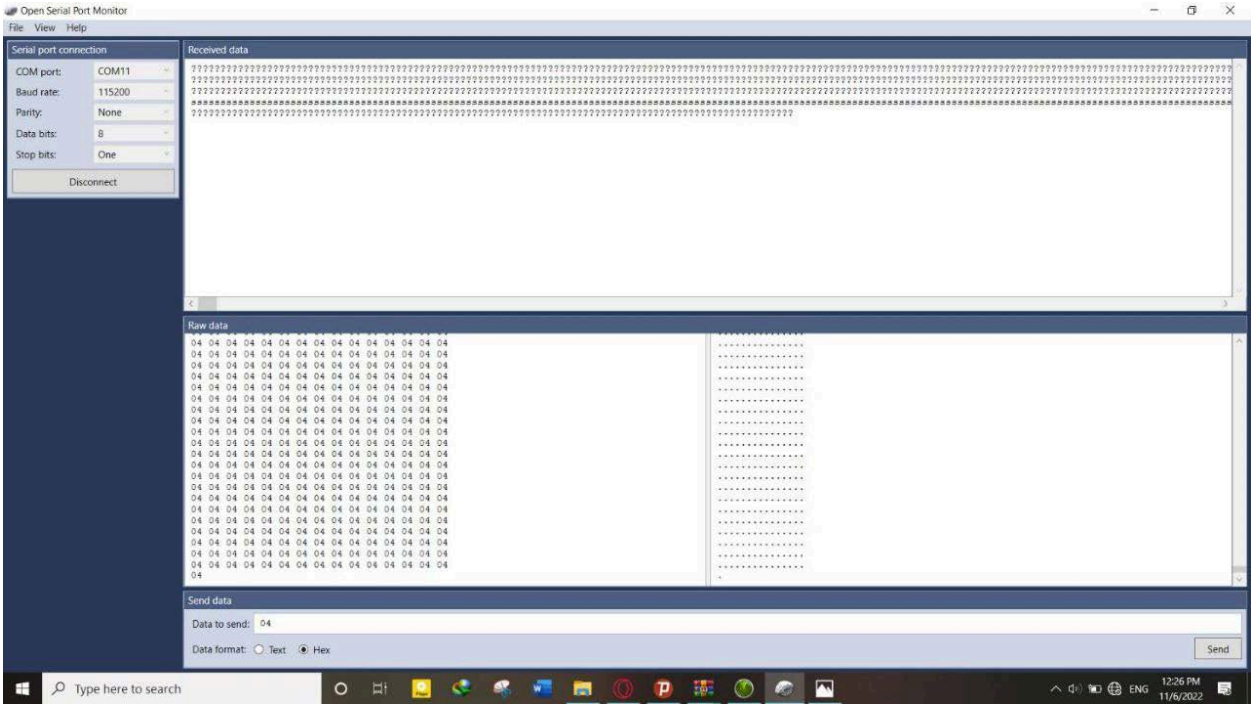
As you can see, in all below images, received data is the same as transmitted data which shows that the FPGA that is programmed by our UART module receives and sends correctly.

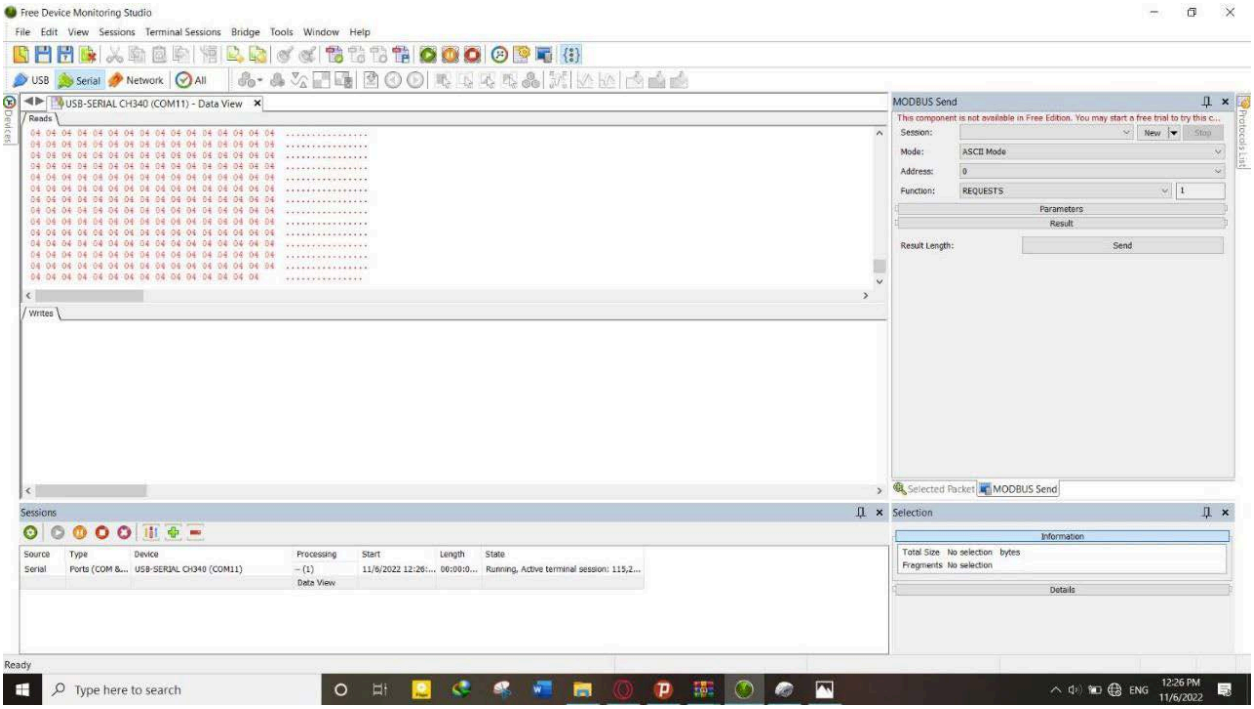


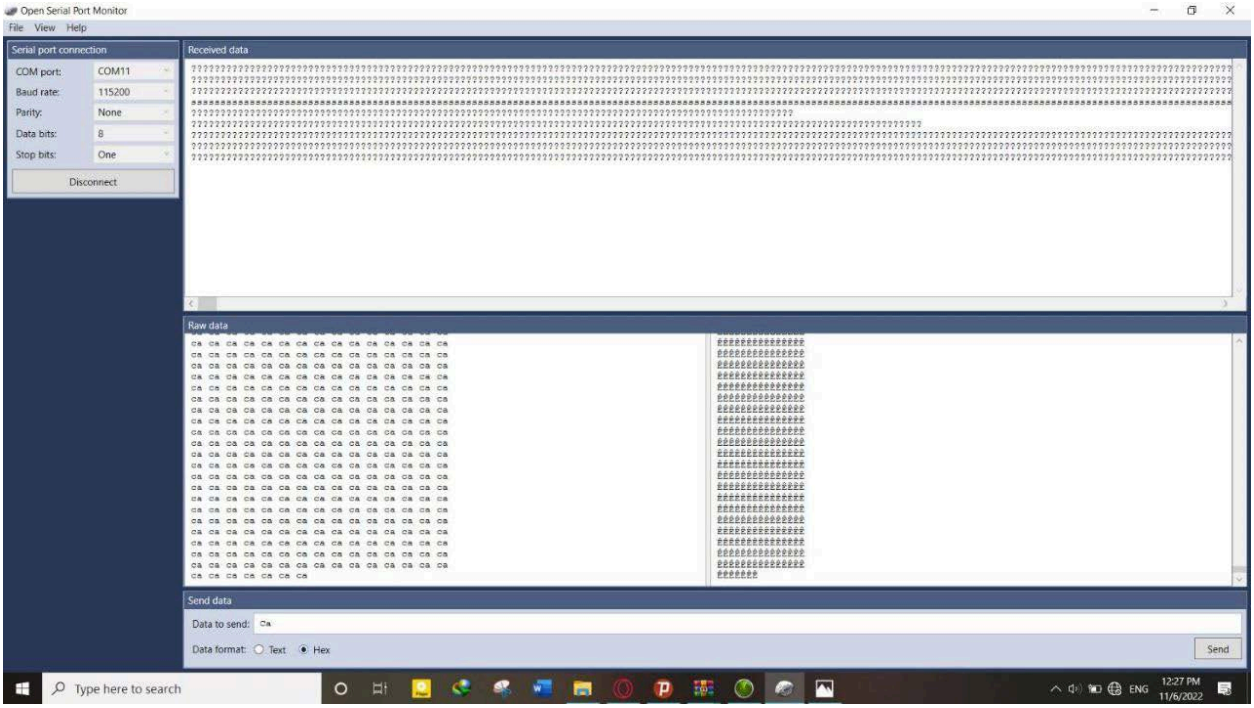


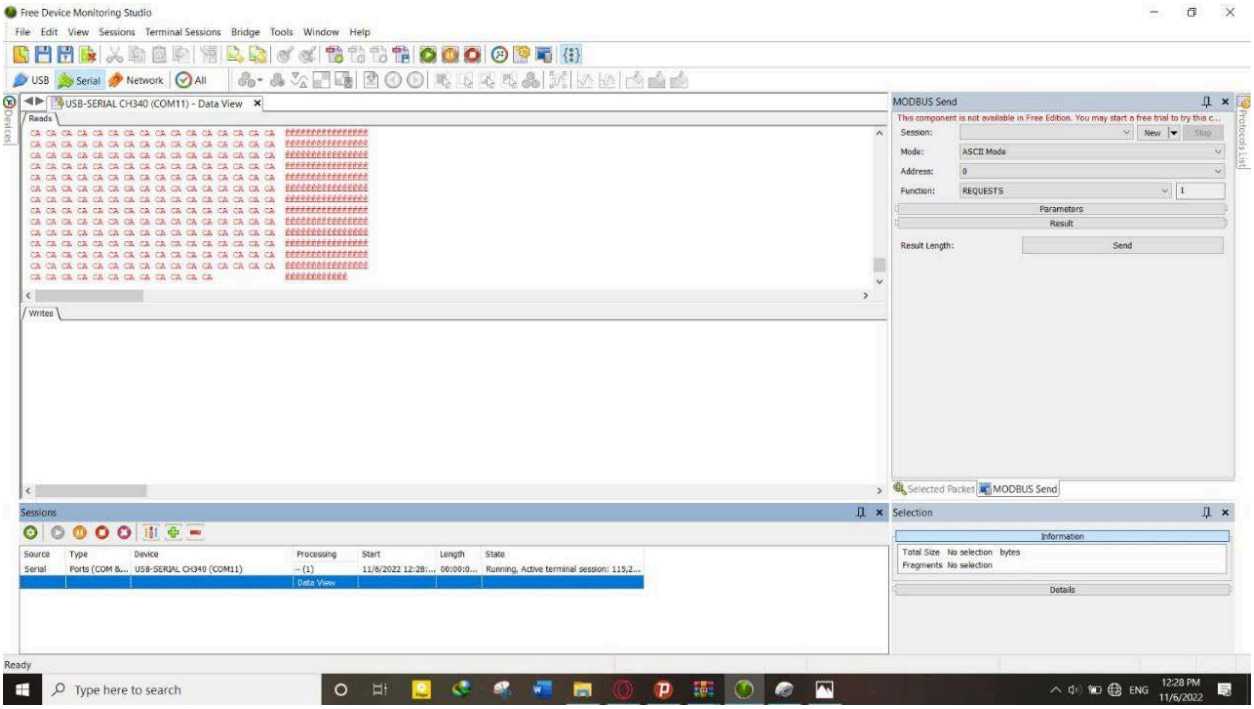

Open Serial Port Monitor
File View Help
Serial port connection
COM port: COM11
Baud rate: 115200
Parity: None
Data bits: 8
Stop bits: One
Disconnect
Received data
Raw data
Send data
Data to send: 9a
Data format: Text Hex
Send
Type here to search
ENG
12:25 PM
11/6/2022


Free Device Monitoring Studio
File Edit View Sessions Terminal Sessions Bridge Tools Window Help
USB Serial Network All
USB-SERIAL CH340 (COM11) - Data View
Reads
Writes
MODBUS Send
This component is not available in Free Edition. You may start a free trial to try this c...
Session:
New
Stop
Mode: ASCII Mode
Address: 0
Function: REQUESTS
1
Parameters
Result
Result Length:
Send
Selected Packet
MODBUS Send
Sessions
Source Type Device Processing Start Length State
Serial Ports (COM &... USB-SERIAL CH340 (COM11) – (1) Data View 11/6/2022 12:25:... 00:00:0... Running, Active terminal session: 115,2...
Selection
Information
Total Size No selection bytes
Fragments No selection
Details
Ready
Type here to search
ENG
12:25 PM
11/6/2022


Open Serial Port Monitor
File View Help
Serial port connection
COM port: COM11
Baud rate: 115200
Parity: None
Data bits: 8
Stop bits: One
Disconnect
Received data
Raw data
Send data
Data to send: 04
Data format: Text Hex
Send
Type here to search
ENG
12:26 PM
11/6/2022


Free Device Monitoring Studio
File Edit View Sessions Terminal Sessions Bridge Tools Window Help
USB Serial Network All
USB-SERIAL CH340 (COM11) - Data View
Reads
Writes
MODBUS Send
This component is not available in Free Edition. You may start a free trial to try this c...
Session:
New
Stop
Mode: ASCII Mode
Address: 0
Function: REQUESTS
1
Parameters
Result
Result Length:
Send
Selected Packet
MODBUS Send
Sessions
Source Type Device Processing Start Length State
Serial Ports (COM &... USB-SERIAL CH340 (COM11) – (1) Data View 11/6/2022 12:26:... 00:00:0... Running, Active terminal session: 115,2...
Selection
Information
Total Size No selection bytes
Fragments No selection
Details
Ready
Type here to search
ENG
12:26 PM
11/6/2022

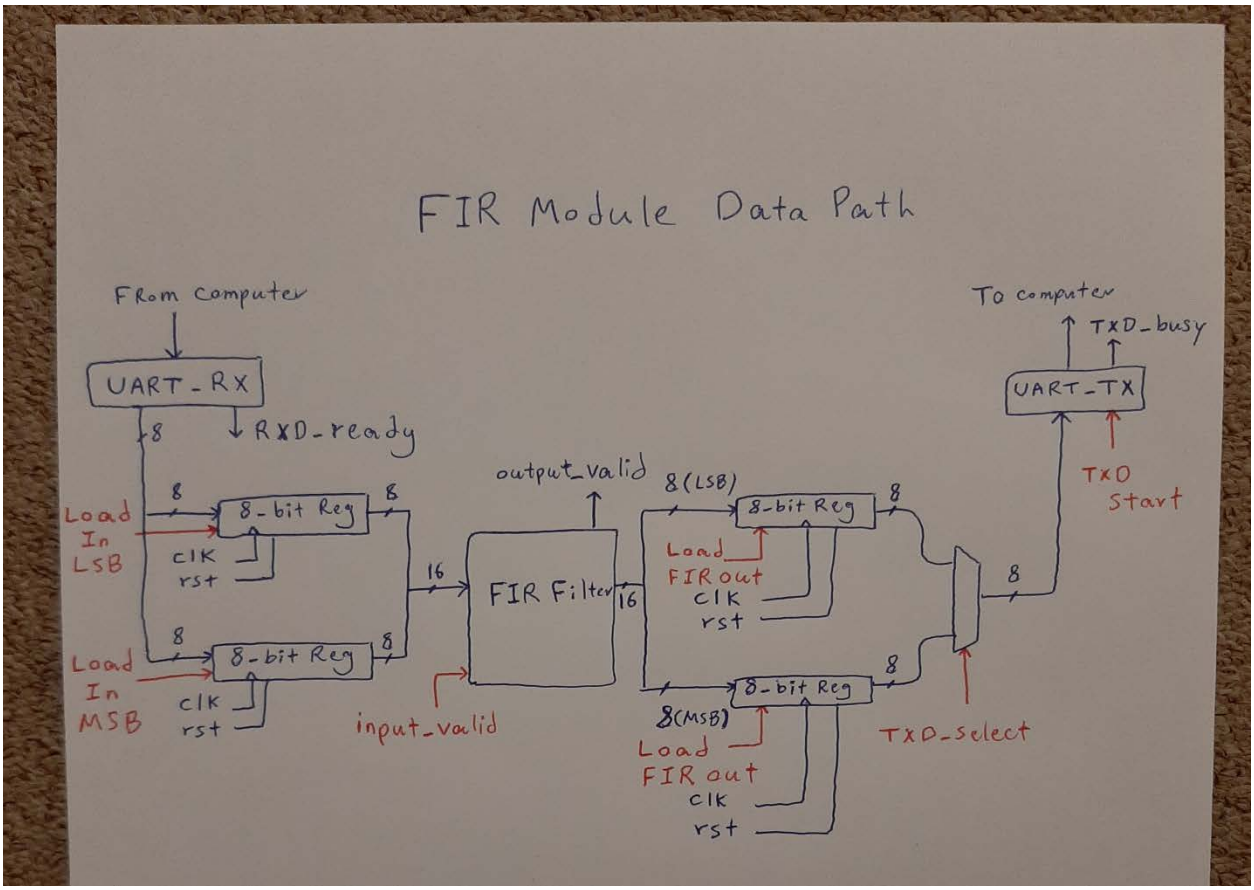

Open Serial Port Monitor
File View Help
Serial port connection
COM port: COM11
Baud rate: 115200
Parity: None
Data bits: 8
Stop bits: One
Disconnect
Received data
Raw data
Send data
Data to send: Ca
Data format: Text Hex
Send
Type here to search
ENG
12:27 PM
11/6/2022


Free Device Monitoring Studio
File Edit View Sessions Terminal Sessions Bridge Tools Window Help
USB Serial Network All
USB-SERIAL CH340 (COM11) - Data View
Reads
Writes
MODBUS Send
This component is not available in Free Edition. You may start a free trial to try this c...
Session:
New
Stop
Mode: ASCII Mode
Address: 0
Function: REQUESTS
1
Parameters
Result
Result Length:
Send
Selected Packet
MODBUS Send
Sessions
Source Type Device Processing Start Length State
Serial Ports (COM &... USB-SERIAL CH340 (COM11) – (1) 11/6/2022 12:28:... 00:00:0... Running, Active terminal session: 115,2...
Data View
Selection
Information
Total Size No selection bytes
Fragments No selection
Details
Ready
Type here to search
ENG
12:28 PM
11/6/2022

FIR Module (Top Level) Implementation

Data Path and Controller Implementation

Data Path:



1. First UART's Asynchronous Receiver receives the input of FIR Module.

2. Received input is stored in 2 8-bit registers (D – Flip Flop).

3. Now main FIR Filter unit has its input (input_valid is asserted) and after a few clock cycles FIR Filter output is calculated (output_valid is asserted).

4. Calculated output is stored in 2 8-bit registers (D – Flip Flop).

5. At the end UART's Asynchronous Transmitter sends the output of FIR Module.

D:/modelsim_ase/Projects/FIR_Module_DataPath.v (/FIR_Module_Test_Bench/FIR/DataPath)

File Edit View Tools Bookmarks Window Help

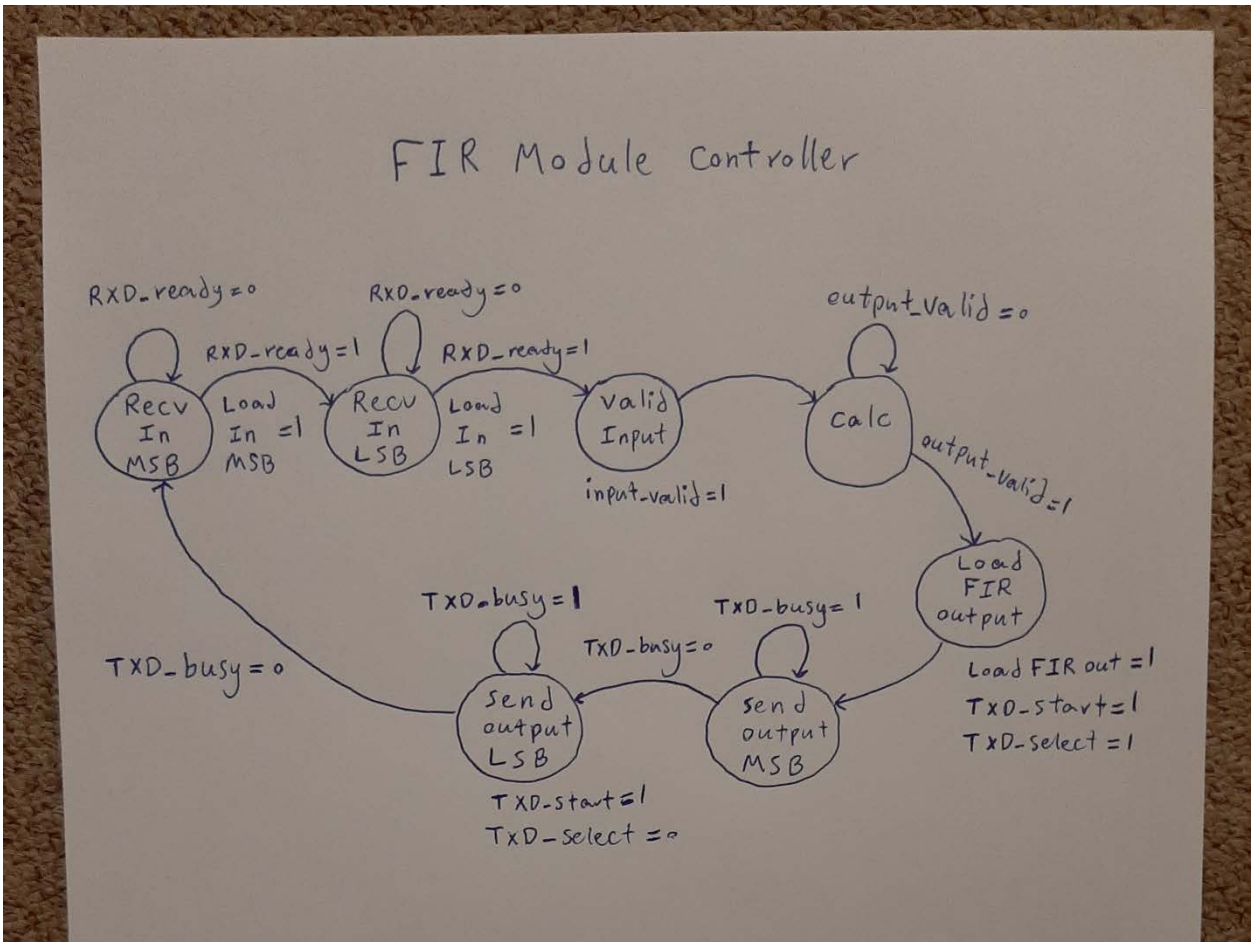
D:/modelsim_ase/Projects/FIR_Module_DataPath.v (/FIR_Module_Test_Bench/FIR/DataPath) - Default

Ln# 1208697 ns

```
1  module FIR_Module_DataPath(clk, reset, RxD, TxD_start, input_valid, Load_FIR_In_LSB, Load_FIR_In_MSB, TxD_data_select, Load_FIR_OUT, output_valid, RxD_data_ready, TxD, TxD_busy);
2          input clk, reset, RxD, TxD_start, input_valid, Load_FIR_In_LSB, Load_FIR_In_MSB, TxD_data_select, Load_FIR_OUT;
3          output output_valid, RxD_data_ready, TxD, TxD_busy;
4
5  wire [7 : 0]RxD_data, TxD_data, LSB_FIR_Out, MSB_FIR_Out;
6  wire [15 : 0]FIR_input;
7  wire [37 : 0]FIR_output;
8
9  UART uart(.clk(clk), .RxD(RxD), .TxD_start(TxD_start), .TxD_data(TxD_data), .RxD_data_ready(RxD_data_ready), .RxD_data(RxD_data), .TxD(TxD), .TxD_busy(TxD_busy));
10
11 Register_D_Flip_Flop FIR_In_LSB(clk, reset, Load_FIR_In_LSB, RxD_data, FIR_input[7 : 0]);
12 Register_D_Flip_Flop FIR_In_MSB(clk, reset, Load_FIR_In_MSB, RxD_data, FIR_input[15 : 8]);
13
14 //FIR_LENGTH = 64
15 //FIR_INPUT_WIDTH = 16
16 //FIR_OUTPUT_WIDTH = 38
17 FIR_Filter #(64, 16, 38) FIR(.clk(clk), .reset(reset), .FIR_input(FIR_input), .input_valid(input_valid), .FIR_output(FIR_output), .output_valid(output_valid));
18
19 Register_D_Flip_Flop FIR_Out_LSB(clk, reset, Load_FIR_OUT, FIR_output[7 : 0], LSB_FIR_Out);
20 Register_D_Flip_Flop FIR_Out_MSB(clk, reset, Load_FIR_OUT, FIR_output[15 : 8], MSB_FIR_Out);
21
22 assign TxD_data = TxD_data_select ? MSB_FIR_Out: LSB_FIR_Out;
23
24 endmodule
```

Ln: 24 Col: 9

Controller:



Receive input MSB: Whenever RxD_data_ready is asserted 8 bits of input data are received by UART's Asynchronous Receiver, which is the MSB of the input, so Load_FIR_In_MSB is asserted to store the received data.

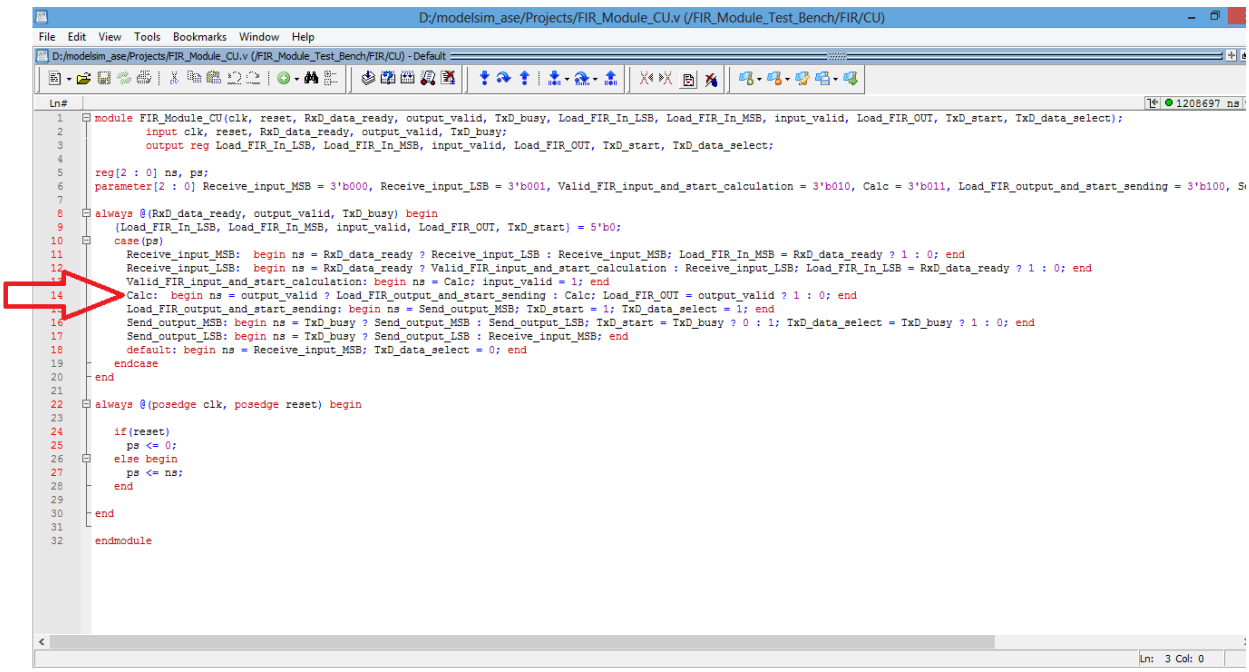
Receive input LSB: Again whenever RxD_data_ready is asserted 8 bits of input data are received by UART's Asynchronous Receiver, which is the LSB of the input, so Load_FIR_In_MSB is asserted to store the received data.

Valid FIR input and start calculation: Now main FIR unit input is ready so input_valid is asserted.

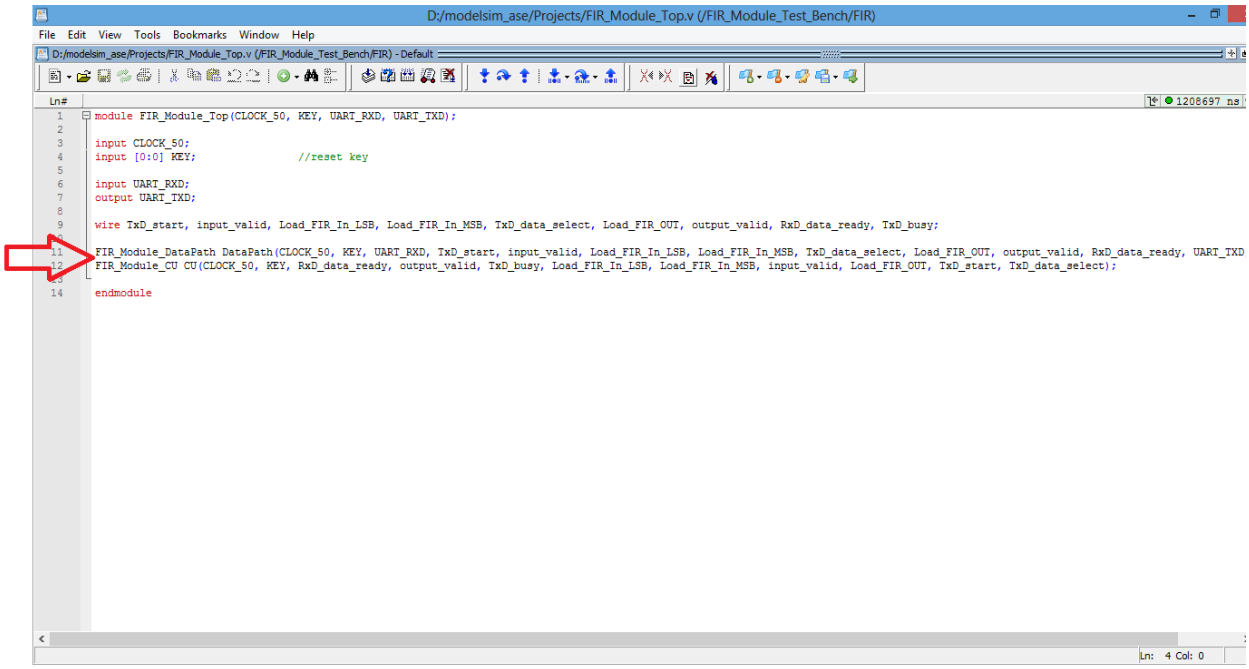
Load FIR output and start sending: After a few clock cycles FIR Filter output is calculated (output_valid is asserted), so Load_FIR_Out is asserted to store the main FIR Filter unit output.

Send output MSB: TxD_select is set to 1 to send the MSB of FIR Filter output and Txd_start is asserted. Whenever Txd_busy gets 0 the MSB of FIR Filter output will be sent.

Send output LSB: TxD_select is set to 0 to send the LSB of FIR Filter output and Txd_start is asserted. Whenever Txd_busy gets 0 the LSB of FIR Filter output will be sent.



FIR Module (Top Level)

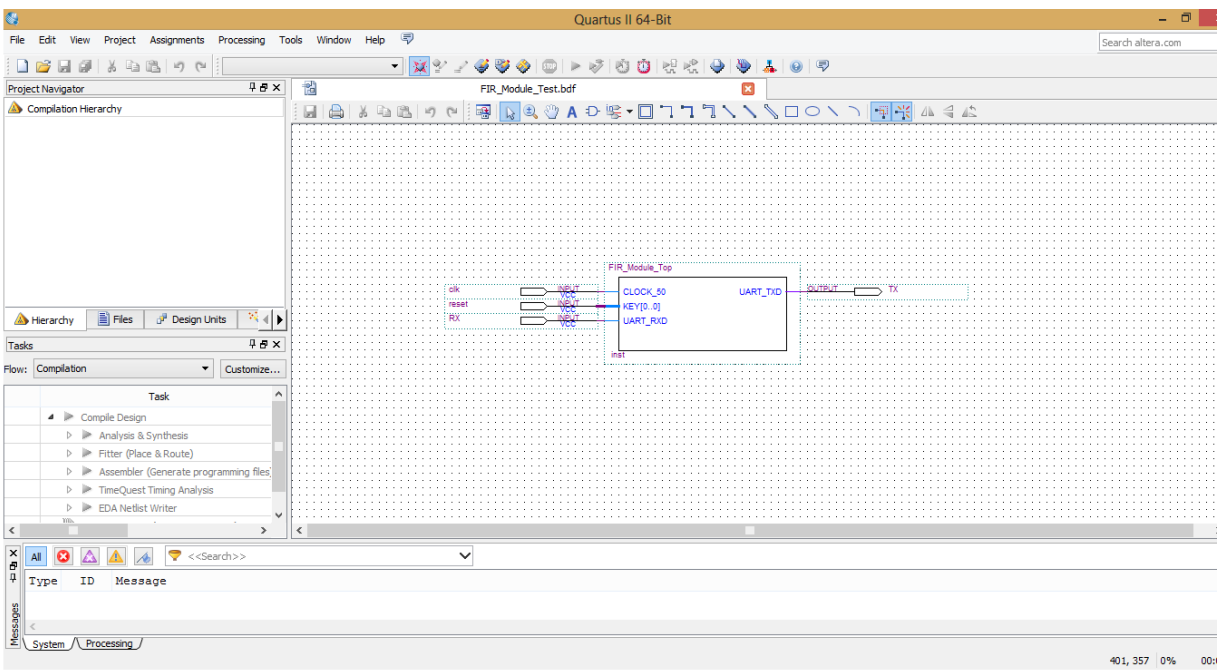


FIR Module (Top Level) Implementation

FIR Module (Top Level) Implementation and Verification

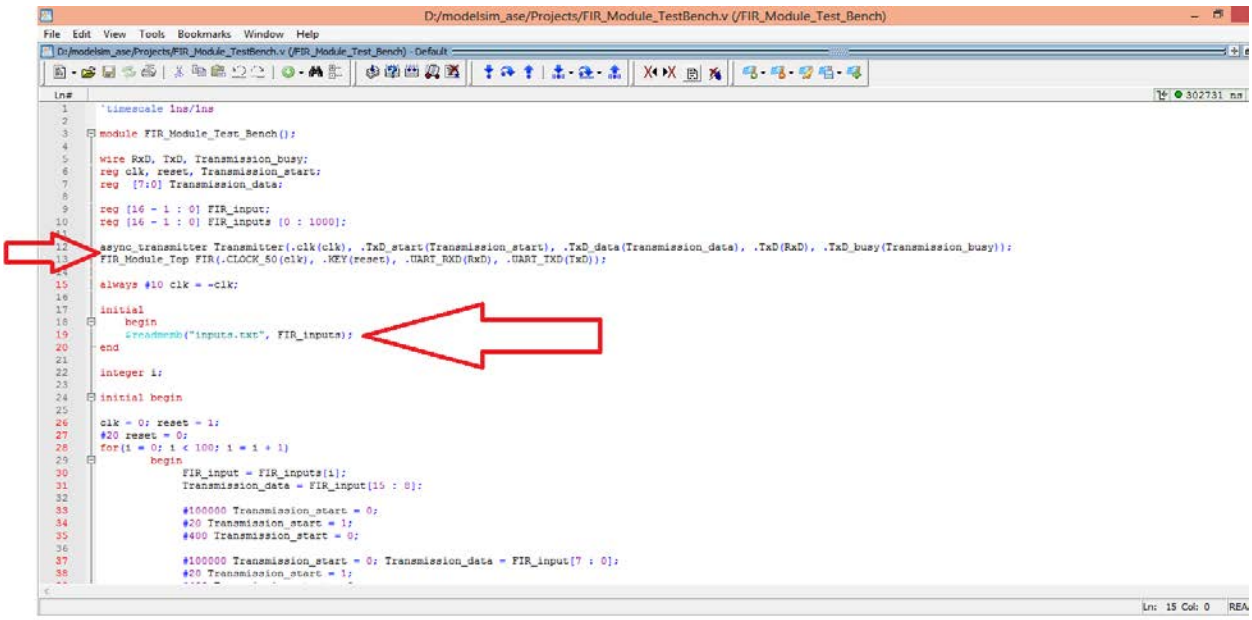
To test FIR Module (Top Level), inputs are read from “inputs.txt” then, in a loop data is transferred to FIR Module by the Asynchronous Transmitter, First MSB of input data is transferred then LSB. Finally we expect the FIR module to send the correct outputs (shown in “outputs.txt”) through its TxD port.

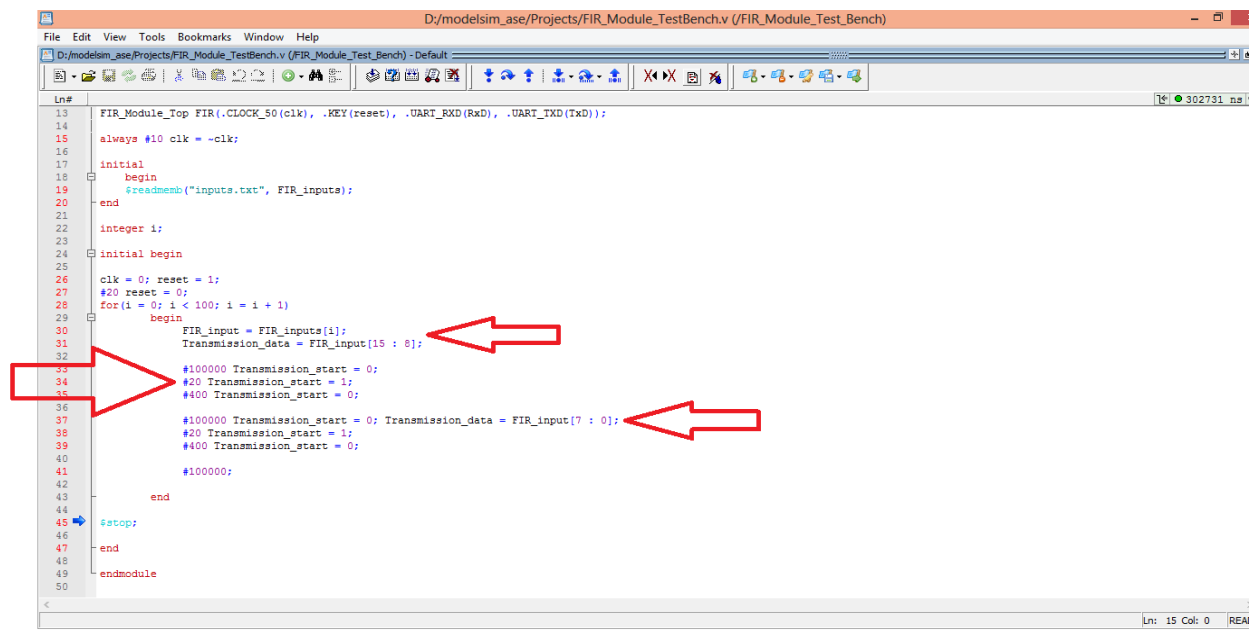
Quartus Synthesized FIR Module (Top Level)

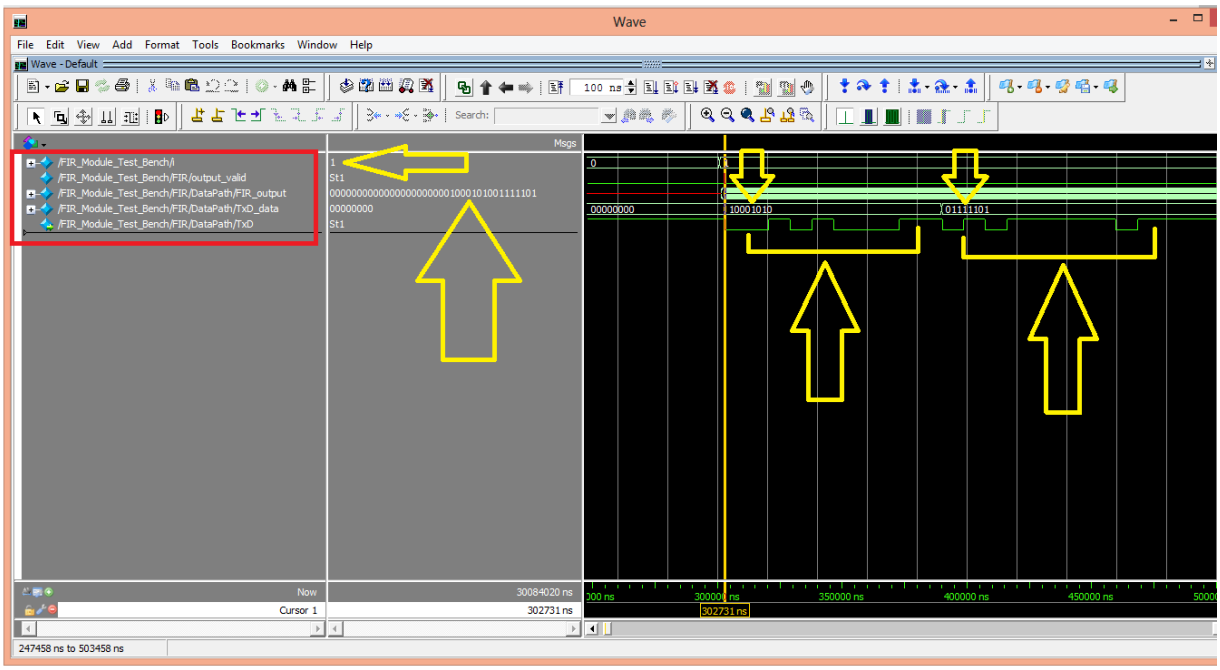


FIR Module (Top Level) Verilog Test Bench

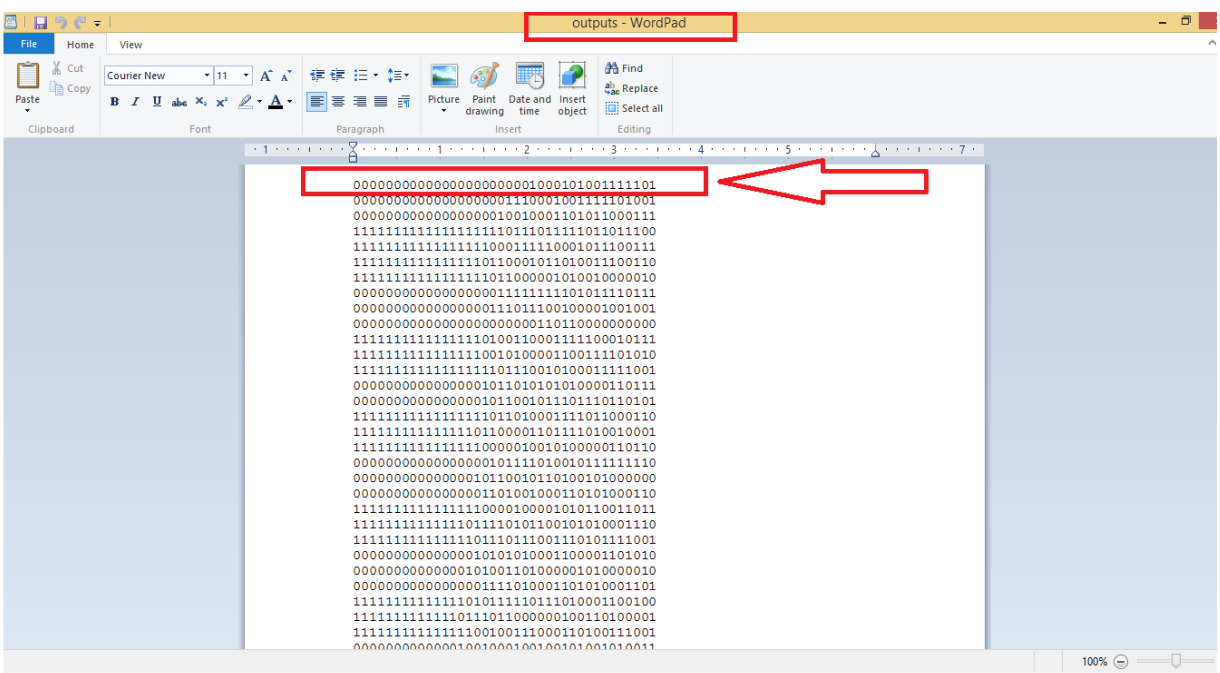
To test FIR Module (Top Level), inputs are read from “inputs.txt” then, in a loop data is transferred to FIR Module by the Asynchronous Transmitter, First MSB of input data is transferred then LSB. Finally we expect the FIR module to send the correct outputs (shown in “outputs.txt”) through its TxD port.

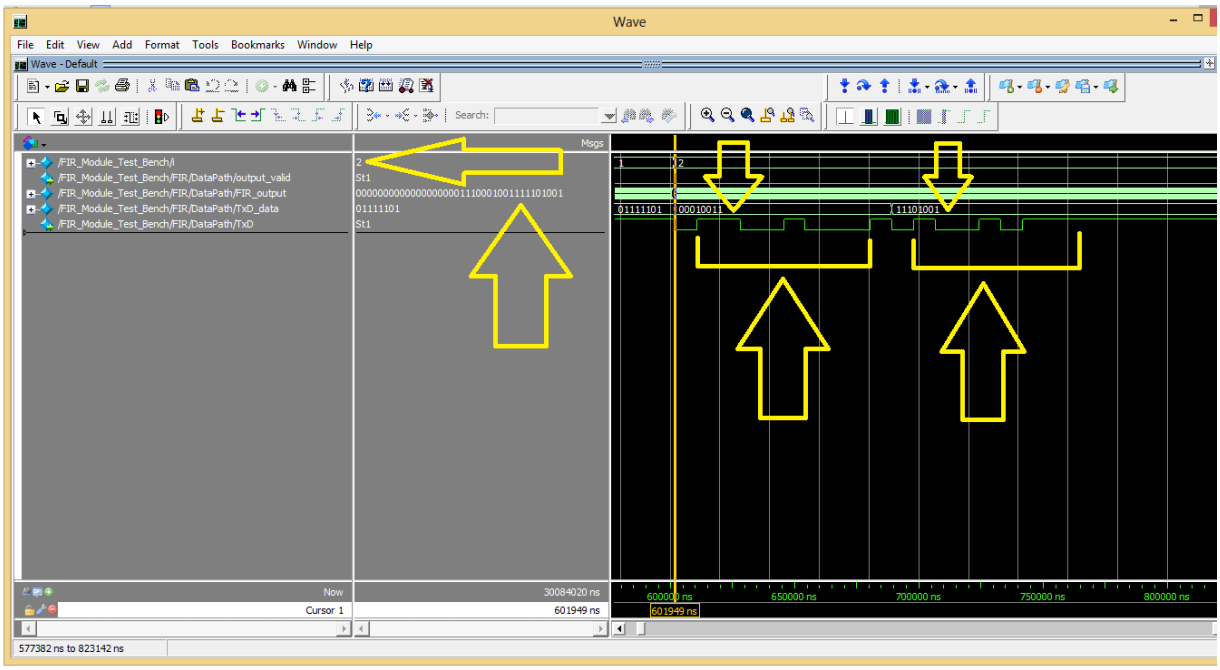




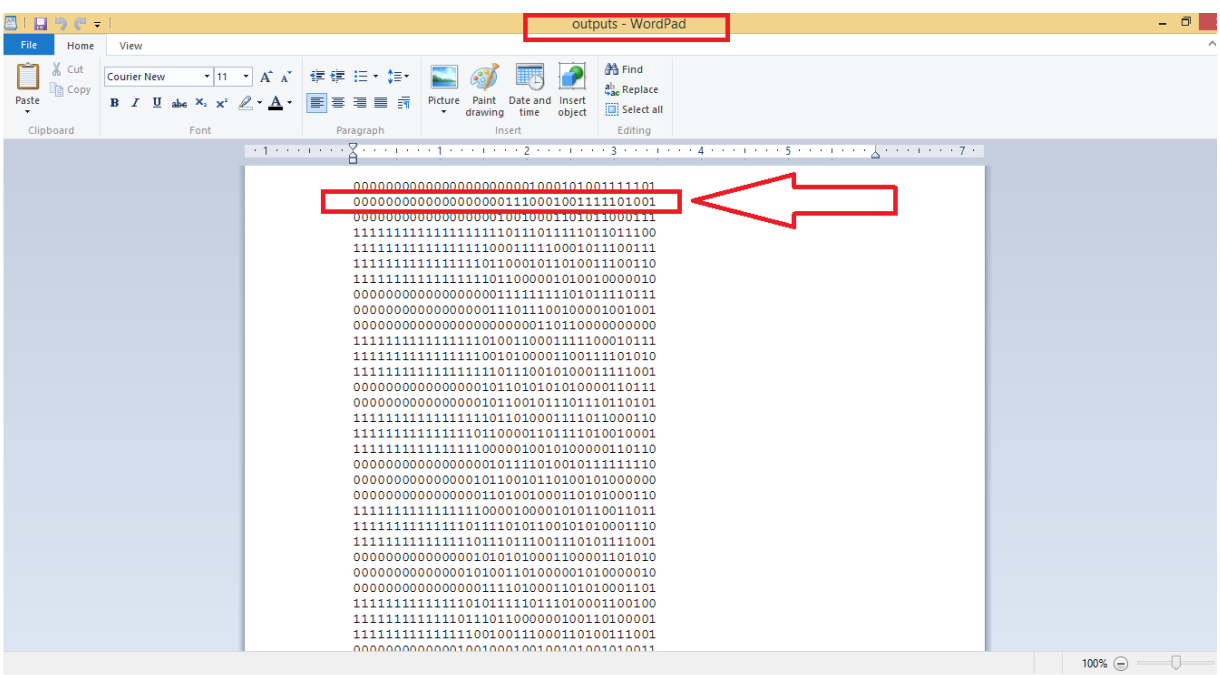


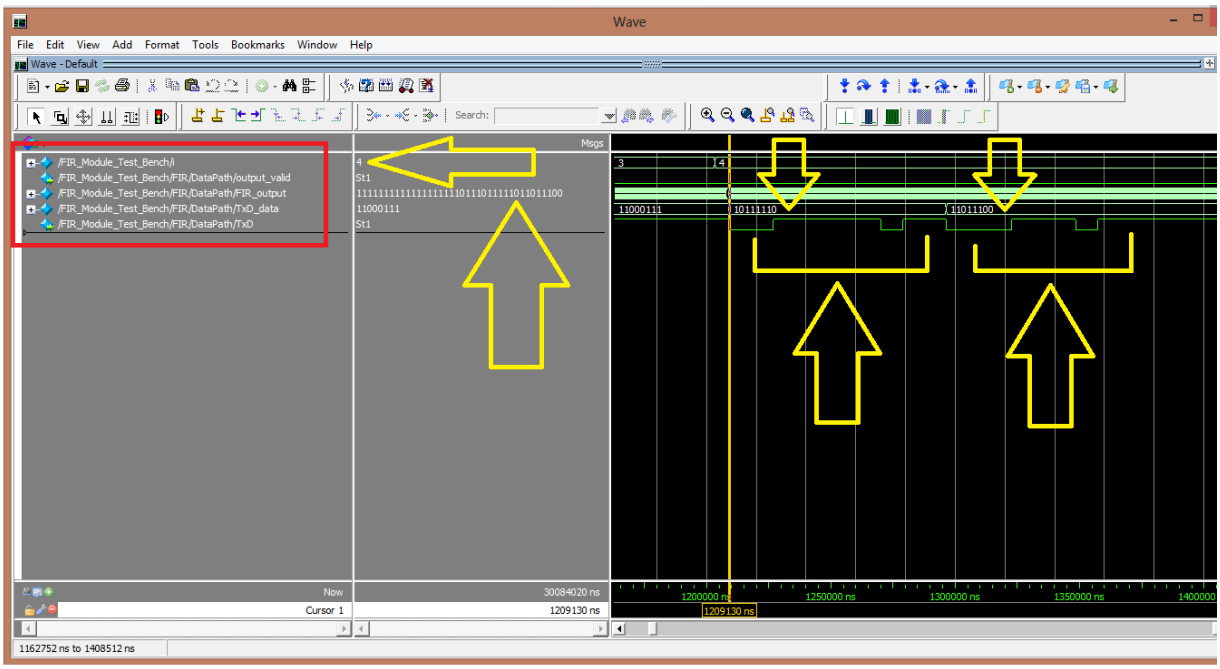
As you can see, the FIR Module output (FIR_output) is “1000101001111101” for first input (i = 1) which is equal to the first expected output shown in the first line of “outputs.txt” (see below image). Finally you see that the FIR Module output is sent through the TxD port serially (First the MSB of output is sent then the LSB).



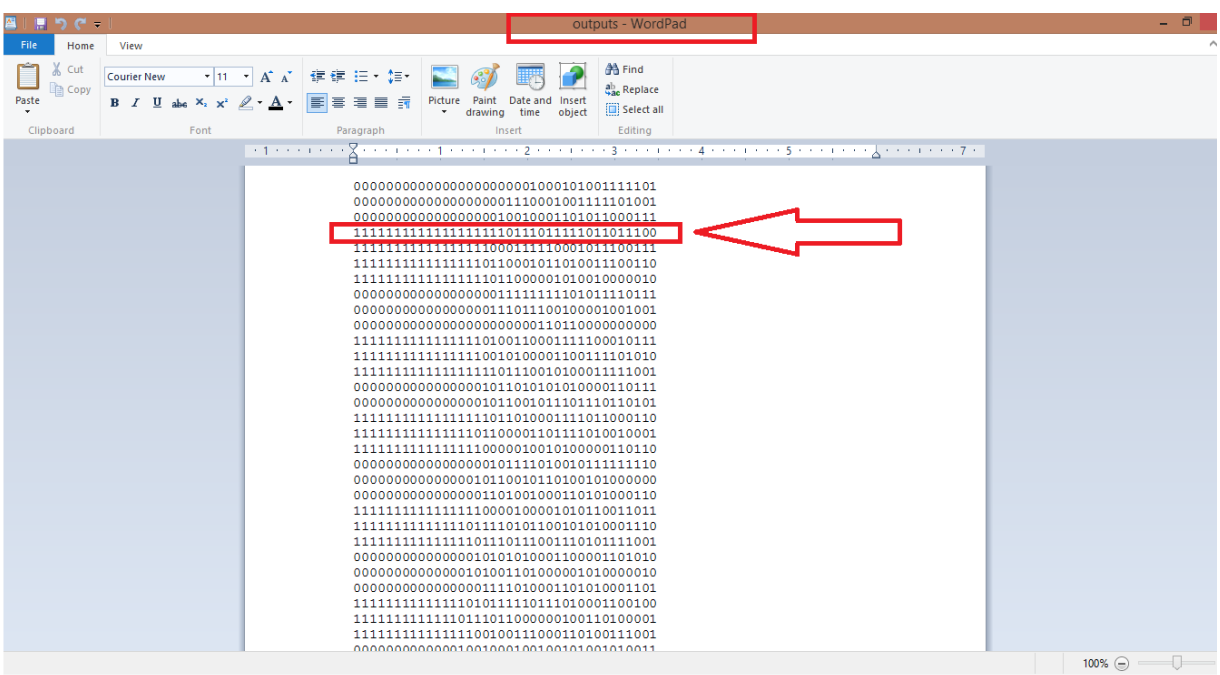


As you can see, the FIR Module output (FIR_output) is “0001001111101001” for second input (i = 2) which is equal to the second expected output shown in the second line of “outputs.txt” (see below image). Finally you see that the FIR Module output is sent through the TxD port serially (First the MSB of output is sent then the LSB).



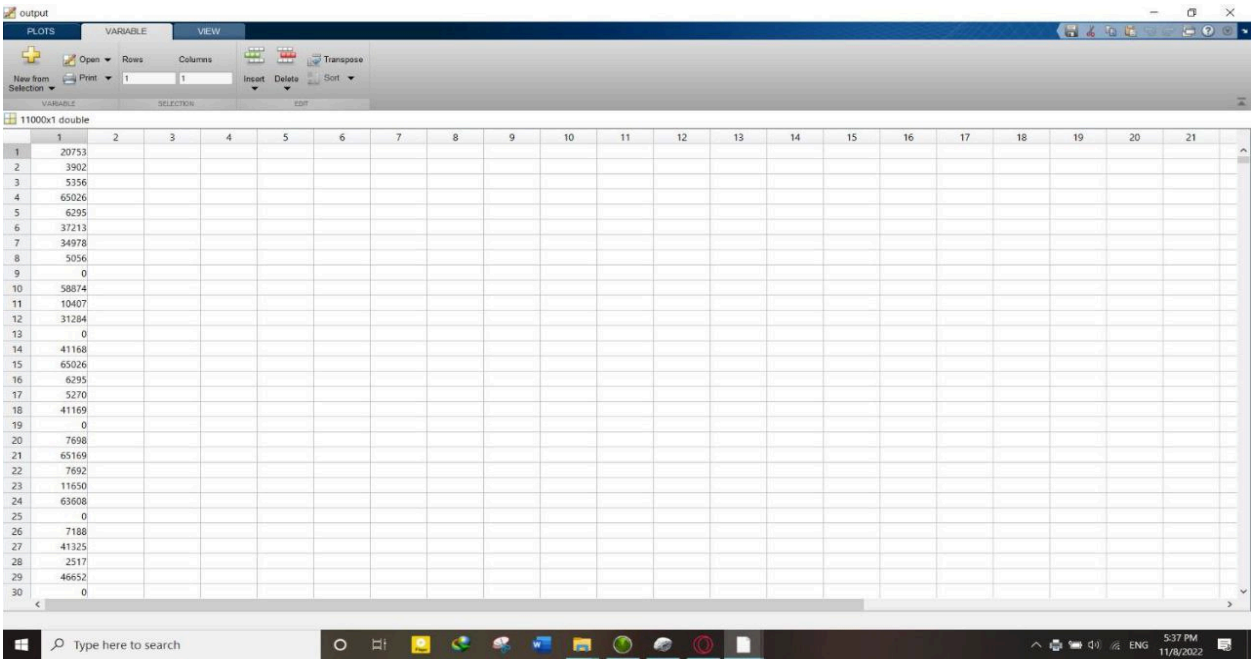


As you can see, the FIR Module output (FIR_output) is “1011111011011100” for fourth input (i = 4) which is equal to the fourth expected output shown in the fourth line of “outputs.txt” (see below image). Finally you see that the FIR Module output is sent through the TxD port serially (First the MSB of output is sent then the LSB).

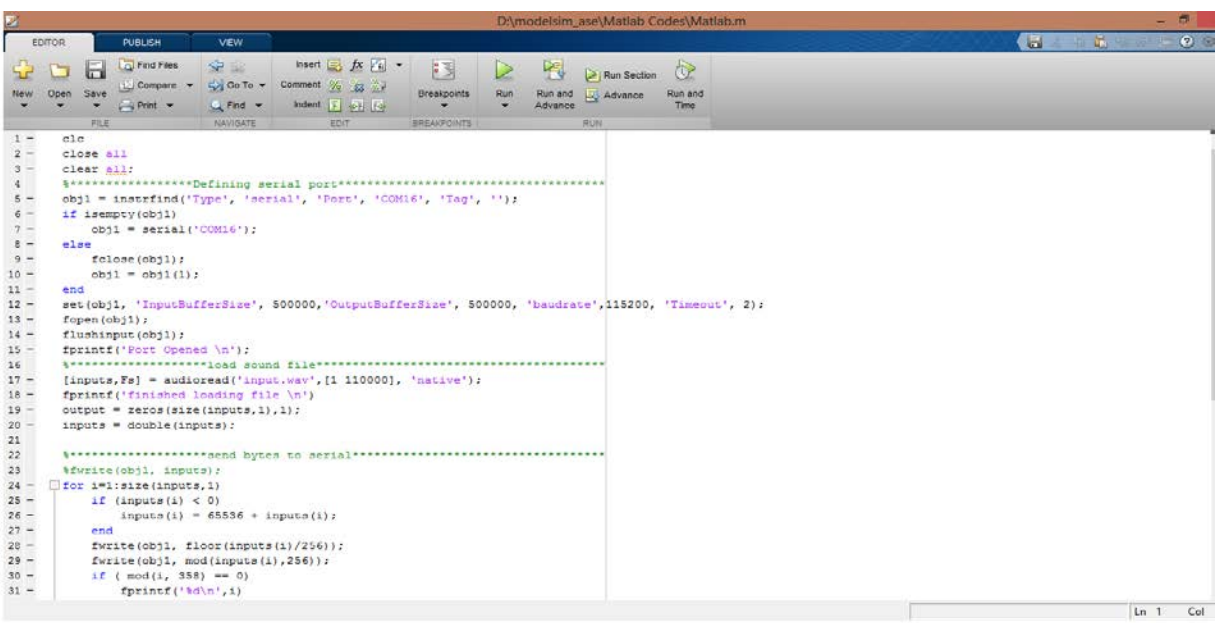


FPGA’s results

As you can see, in below image, FPGA which is programmed by our FIR Module sends the output of FIR Module (Top Level) and the output is received and stored in “output” variable by MATLAB code (shown below).



	1
1	20753
2	3902
3	5356
4	65026
5	6295
6	37213
7	34978
8	5056
9	0
10	58874
11	10407
12	31284
13	0
14	41168
15	65026
16	6295
17	5270
18	41169
19	0
20	7698
21	65169
22	7692
23	11650
24	63608
25	0
26	7188
27	41325
28	2517
29	46652
30	0




```
clc
close all
clear all;
%****************Defining serial port*********************************
obj1 = instrfind('Type', 'serial', 'Port', 'COM16', 'Tag', '');
if isempty(obj1)
    obj1 = serial('COM16');
else
    fclose(obj1);
    obj1 = obj1(1);
end
set(obj1, 'InputBufferSize', 500000,'OutputBufferSize', 500000, 'baudrate',115200, 'Timeout', 2);
fopen(obj1);
flushinput(obj1);
fprintf('Port Opened \n');
%******************load sound file*************************************
[inputs,Fs] = audioread('input.wav',[1 110000], 'native');
fprintf('finished loading file \n')
output = zeros(size(inputs,1),1);
inputs = double(inputs);

%******************send bytes to serial*********************************
%fwrite(obj1, inputs);
for i=1:size(inputs,1)
    if (inputs(i) < 0)
        inputs(i) = 65536 + inputs(i);
    end
    fwrite(obj1, floor(inputs(i)/256));
    fwrite(obj1, mod(inputs(i),256));
    if ( mod(i, 358) == 0)
        fprintf('%d\n',i)
```

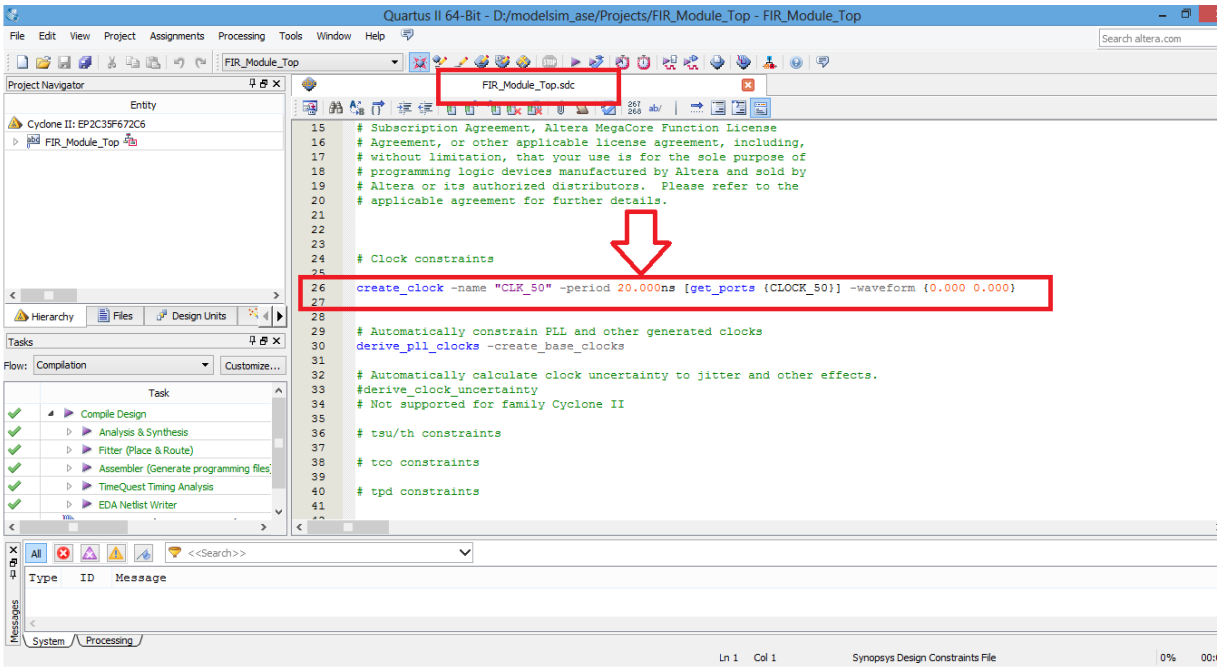
D:\modelsim_ase\Matlab Codes\Matlab.m

```
for i=1:size(inputs,1)
    if (inputs(i) < 0)
        inputs(i) = 65536 + inputs(i);
    end
    fwrite(obj1, floor(inputs(i)/256));
    fwrite(obj1, mod(inputs(i),256));
    if ( mod(i, 358) == 0)
        fprintf('%d\n',i)
    end
end
fprintf('all data sent successfully \n')
%************************read received data*************************************
raw_data = fread(obj1);
%**********************append all data*********************************
for i=1:size(inputs,1)
    output(i) = raw_data(2*i-1)*256 + raw_data(2*i);
end
fprintf('all data received successfully \n')
%fclose(obj1);

output_converted = zeros(length(output), 1);
for i = 1 : 15
   output_converted = output_converted + mod(output, 2) .* ((2^(i-16))*ones(length(output), 1));
   output = output / 2;
end
output_converted = output_converted + mod(output, 2) .* ((-1)*ones(length(output), 1));
output = output / 2;

audiowrite('Filtered_signal.wav',output_converted ,Fs);
fprintf('Filtered sound saved successfully \n');
```

script Ln 21 Col

In-Text Questions Answers

Step 1 Questions



The code in line 26 says that:

Create a clock signal with the following properties:

Clock signal's name: "CLK_50"

Period: 20 ns (so the frequency is 1/20 ns which is 50 MHz)

This clock signal is assigned to "CLOCK_50" port

Rise time: 0 ns

Fall time: 0 ns

There are other constrains such as tsu/th, tco and tpd that can be added to clock signal's constrains.

In-Text Questions Answers

Step 2 Questions

Routing

Routing of clock signal on FPGA's PCB is different from other signals due to the higher frequency of the clock signal. At high frequency, the impedance of capacitor and inductor changes so it needs different routing unless the different value of impedance of capacitors and inductors of clock's routing will affect the quality of the clock signal including its skew, frequency and etc. the clock signal is used all over the circuit and PCB so if it has a problem, the whole circuit will be damaged thus it is necessary to create a good quality clock signal.

To detect the clock signal, Quartus searches the pin assignment to find out which signal is assigned to "PIN_N2" (CLOCK_50 MHz) or "PIN_D13" (CLOCK_27 MHz). FPGA can also use an external clock signal in addition to its internal clocks (27 MHz and 50 MHz).