Techfest™
INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

VYORIUS

VLSI DESIGN PROJECTS/INTERNSHIP

College Name

→SRM Institute of Science and Technology.

Company Name

→Vyorius.

Domain

→VLSI design.

Project topics

1. Basic circuits written in Verilog/VHDL, simulated and implemented on the FPGA
2. UART communication to print a single character
3. FSM Designs: Mealy & Moore Machines and Up Down Counter

Hardware and software used

Hardware,

Mimas A7 Mini FPGA Development Board
Mimas A7 Mini is an easy-to-use FPGA Development board featuring Artix 7 FPGA (XC7A35T – FTG256C package) with FTDI's FT2232H Dual-Channel USB device. It is an Artix-7 based replacement and upgrades of Mimas Spartan 6 FPGA Board. It is specially designed for the development and integration of FPGA based accelerated features to other designs. The USB 2.0 host interface based on popular FT2232H offers high bandwidth data

transfer and board programming without the need for any external programming adapters.

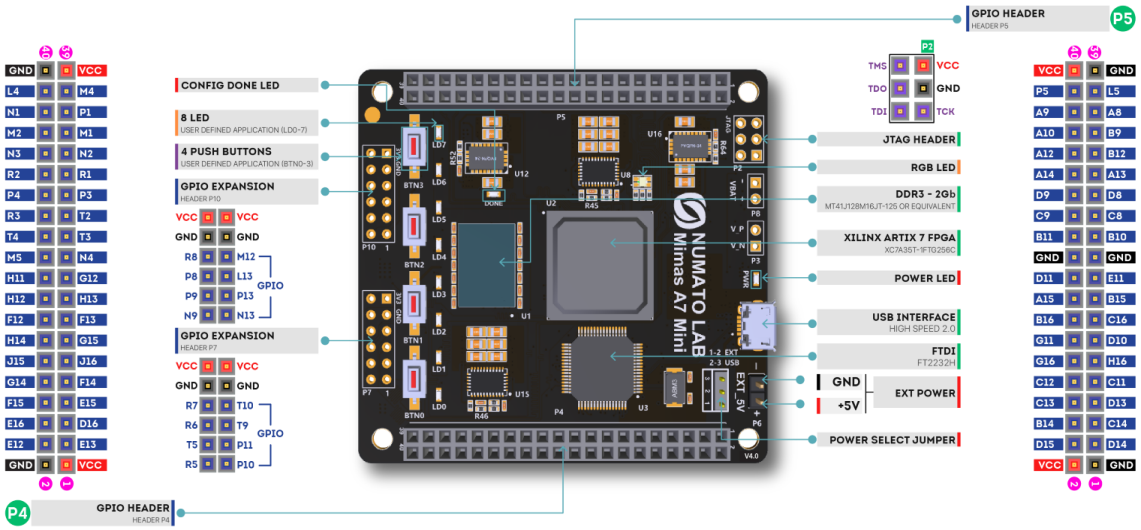
Features,

→ Device: Xilinx Artix 7 FPGA (XC7A35T-1FTG256C)
→ DDR3: 2Gb DDR3 (MT41J128M16JT-125 or equivalent)
→ Built-in programming interface. No expensive JTAG adapters needed for programming the board
→ Onboard 128Mb flash memory for FPGA configuration storage and custom user data storage
→ High-Speed USB 2.0 interface for On-board flash programming. FT2232H Channel B is dedicated to JTAG Programming. Channel A can be used for custom applications
→ 100MHz CMOS oscillator
→ 8 LEDs, 1 RGB LED and 4 Push Buttons for user-defined purposes
→ FPGA configuration via JTAG and USB
→ Maximum IOs for user-defined purposes

FPGA – 70 IOs (35 professionally length matched Differential Pairs) and two 2×6 Expansion Headers

Applications,

→ Product Prototype Development
→ Accelerated computing integration
→ Development and testing of custom embedded processors
→ Communication devices development
→ Educational tool for Schools and Universities



Specifications,

Attribute	Value
Dimensions	6 × 4 × 1 in
FPGA	**XC7A35T – 1FTG256C**
Memory	**DDR3 – 2Gb**
Configuration Options	**JTAG**, **USB**
Host Interface	**USB 2.0**
Primary Clock Frequency	**100MHz**
Number Of GPIOs (Max)	**70**
Number Of Diff Pairs	**35**

Software

Icarus Verilog is a Verilog simulation and synthesis tool. It operates as a compiler, compiling source code written in Verilog (IEEE-1364) into some target format. For batch simulation, the compiler can

generate an intermediate form called *vvp assembly*. This intermediate form is executed by the ``vvp'' command. For synthesis, the compiler generates netlists in the desired format.

The compiler proper is intended to parse and elaborate design descriptions written to the IEEE standard *IEEE Std 1364-2005*. This is a fairly large and complex standard, so it will take some time to fill all the dark alleys of the standard, but that's the goal.

Icarus Verilog is a work in progress, and since the language standard is not standing still either, it probably always will be. That is as it should be. However, I will make stable releases from time to time, and will endeavour to not retract any features that appear in these stable releases. The quick links above will show the current stable release.

The main porting target is Linux, although it works well on many similar operating systems. Various people have contributed precompiled binaries of stable releases for a variety of targets. These releases are ported by volunteers, so what binaries are available depends on who takes the time to do the packaging. *Icarus Verilog* has been ported to That Other Operating System, as a command line tool, and there are installers for users without compilers. You can compile it entirely with free tools, too, although there are precompiled binaries of stable releases.

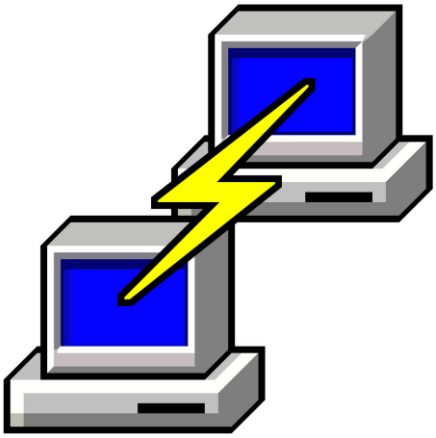
Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of HDL designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis.[1][5][6][7] Vivado represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE).[8][9][10]

Like the later versions of ISE, Vivado includes the in-built logic simulator ISIM.[11] Vivado also introduces high-level synthesis, with a toolchain that converts C code into programmable logic.[6]

Replacing the 15-year-old ISE with Vivado Design Suite took 1000 person-years and cost US$200 million.

Tenagra is an FPGA System management tool for configuring and communicating with Numato Lab's supported FPGA modules and development platforms. This software is designed to be a single interface for managing the devices and exercising some of the available features. Currently, Tenagra supports configuring the FPGA module/board (programming) and Memory Exerciser that can transfer data between various memories on the device. This includes both external DDR Memory and Block RAM available within the FPGA device. With Tenagra, you can create multiple configuration setups with different bitstreams and settings for each device model so that switching between multiple bitstreams is a breeze. This is especially helpful during development where the device may need to be reprogrammed with various bitstreams repeatedly.

Driver for Mimas A7 Mini Module

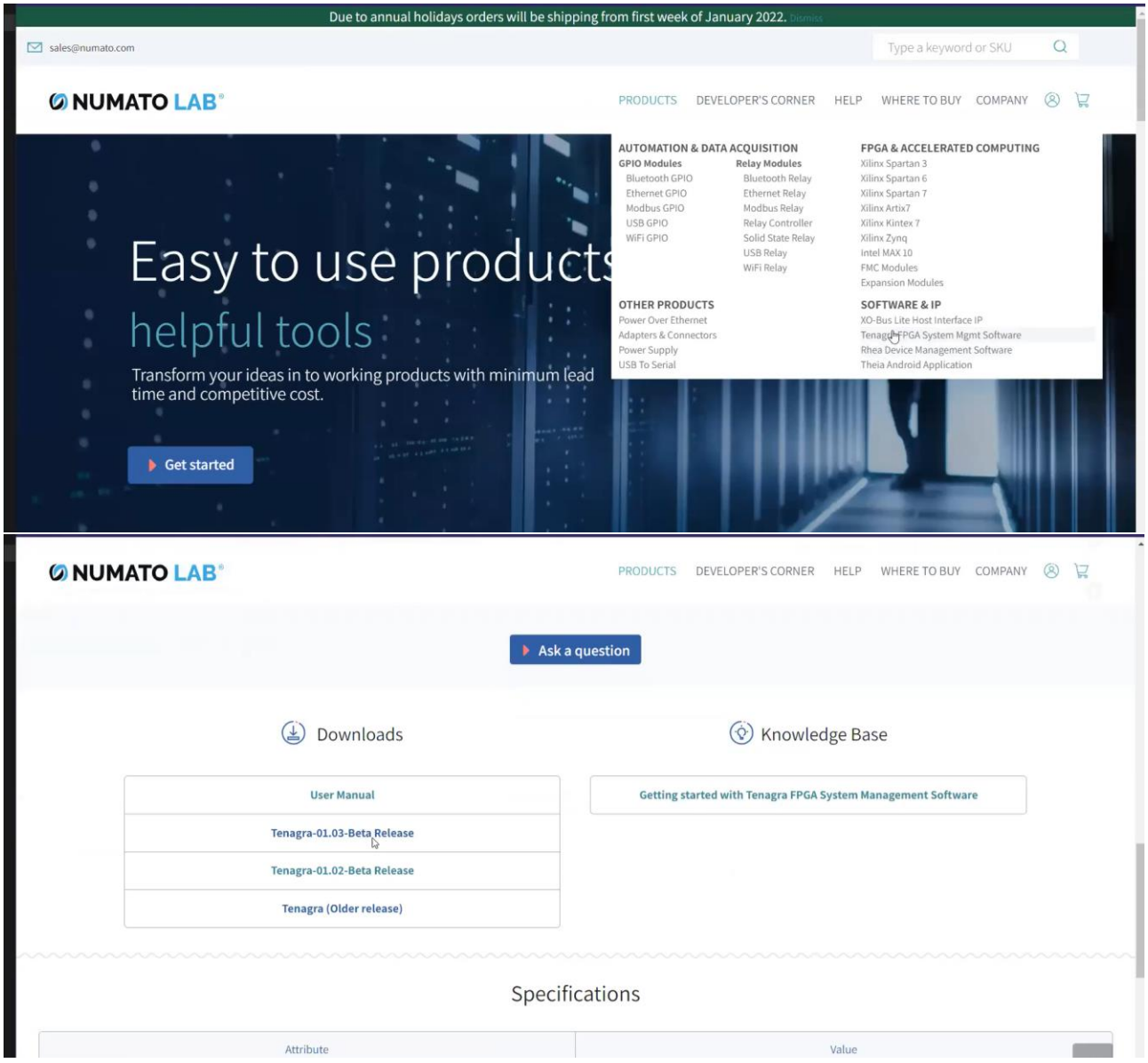


PuTTY is a free implementation of SSH **(and telnet) for PCs running Microsoft Windows** (it also includes an xterm terminal emulator). You will find PuTTY useful if you want to access an account on a Unix or other multi-user system from a PC (for example your own or one in an internet cafe).

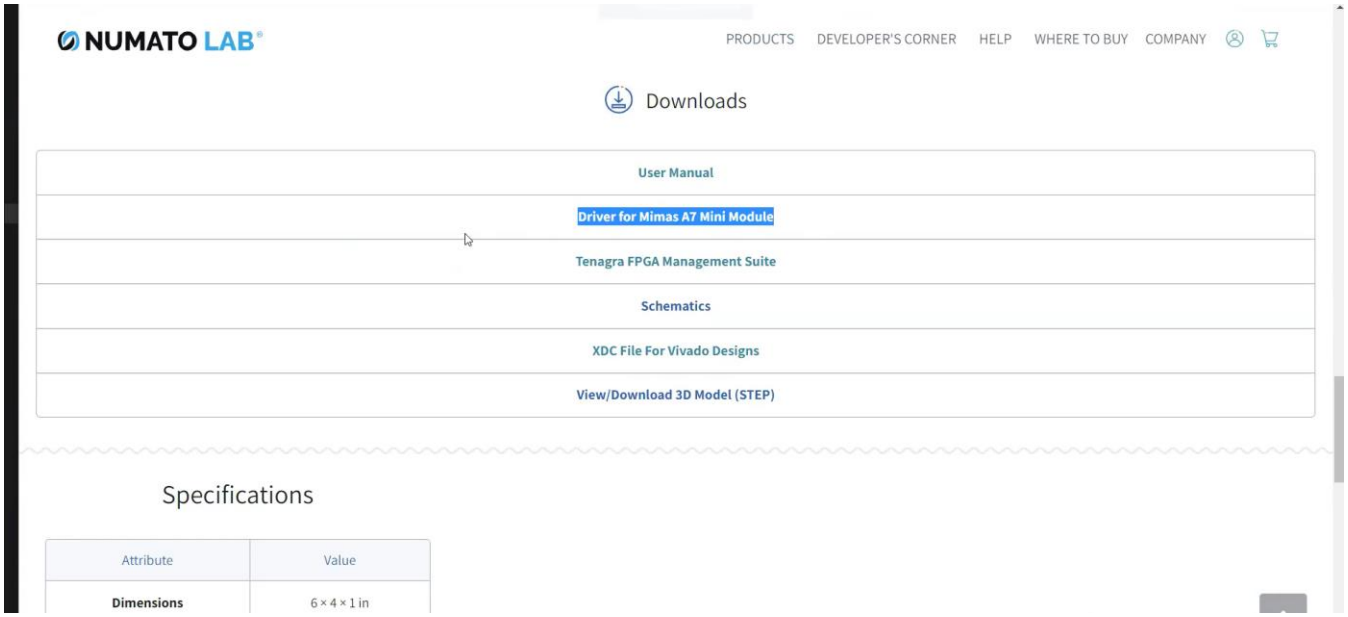
3. FSM Designs: Mealy & Moore Machines and Up Down Counter

Step 1- Connect the real time FPGA Mimas A7 mini board, open the device manager to set its configuration. Now we have to get

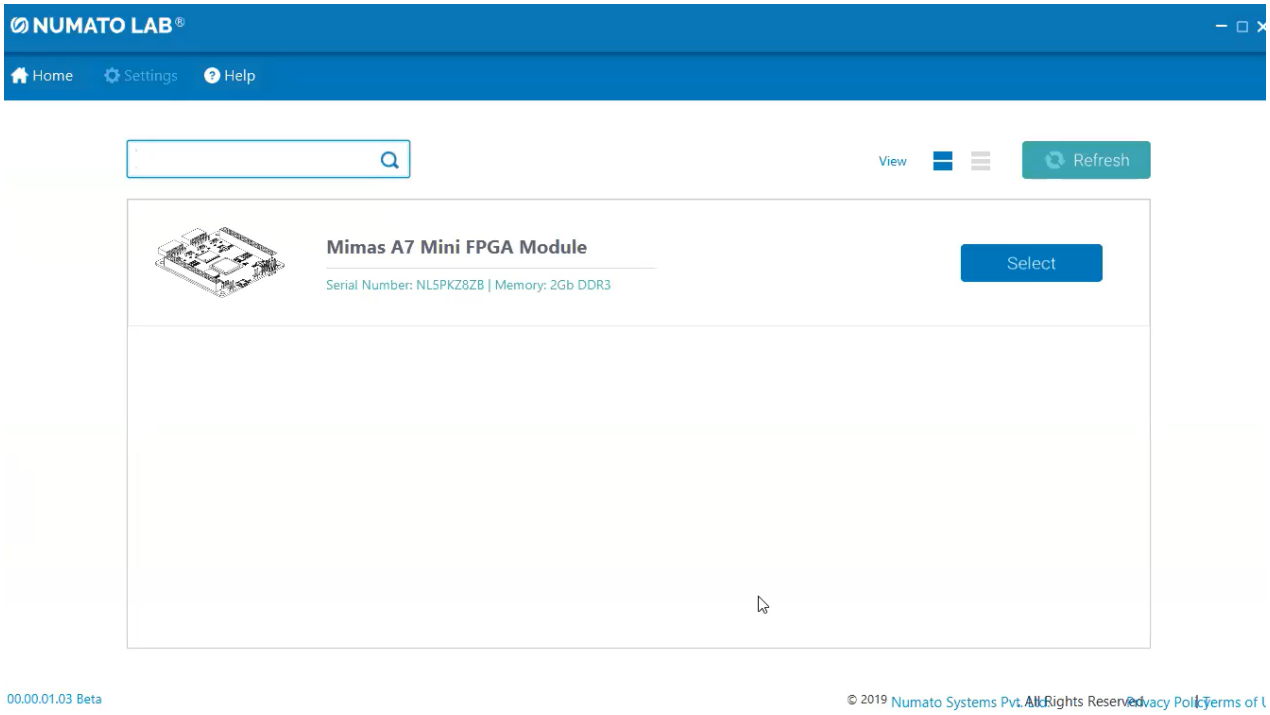
tenegra a software which acts like a bridge between the FPGA board and the pc, where the program will be uploaded. You can get tenegra under Numato Lab



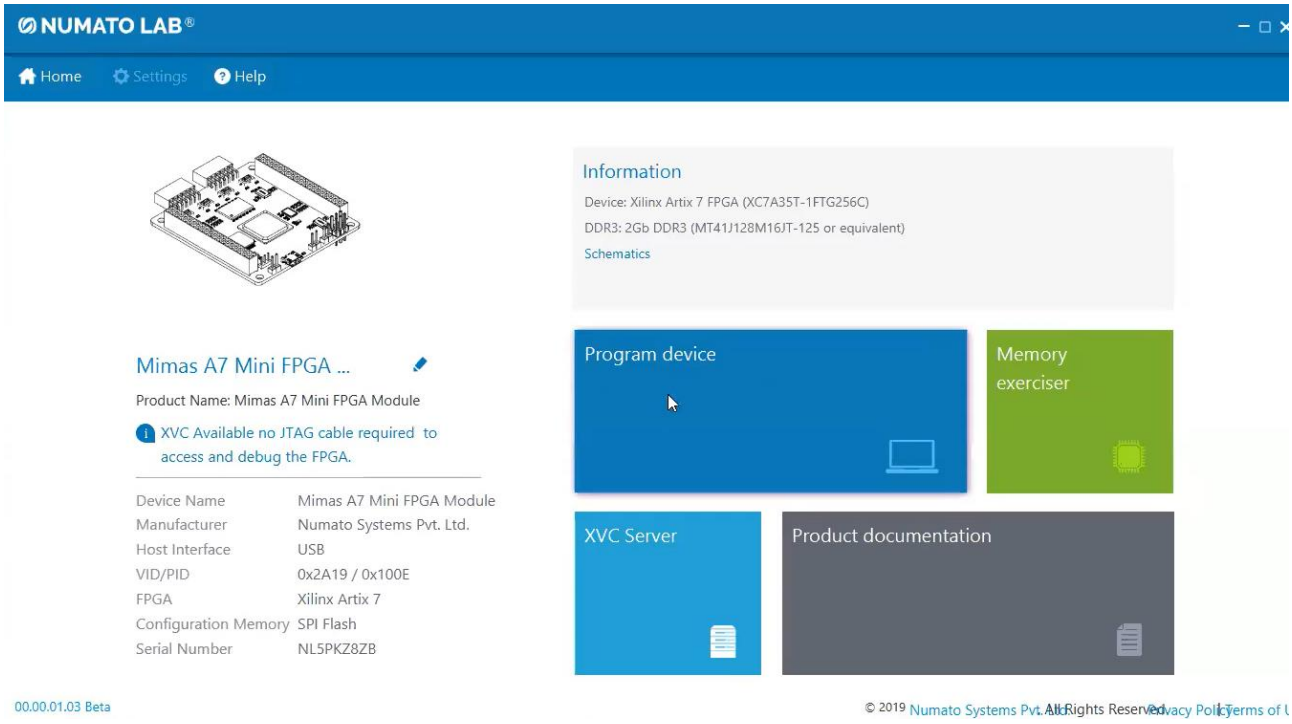
After this we have to download few drivers' fir Mimas A7 Mini Module.



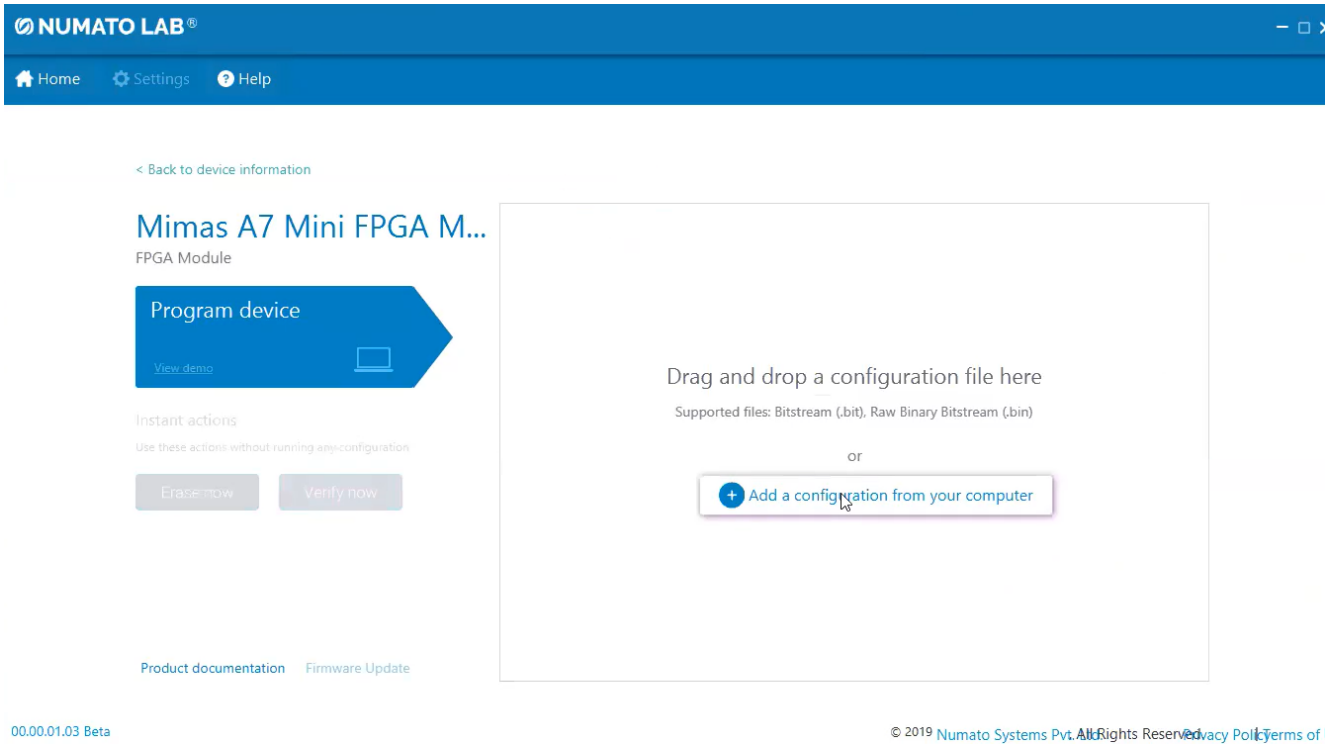
Launch tenegra where we will be viewing two ways or methods to put the program in our FPGA .



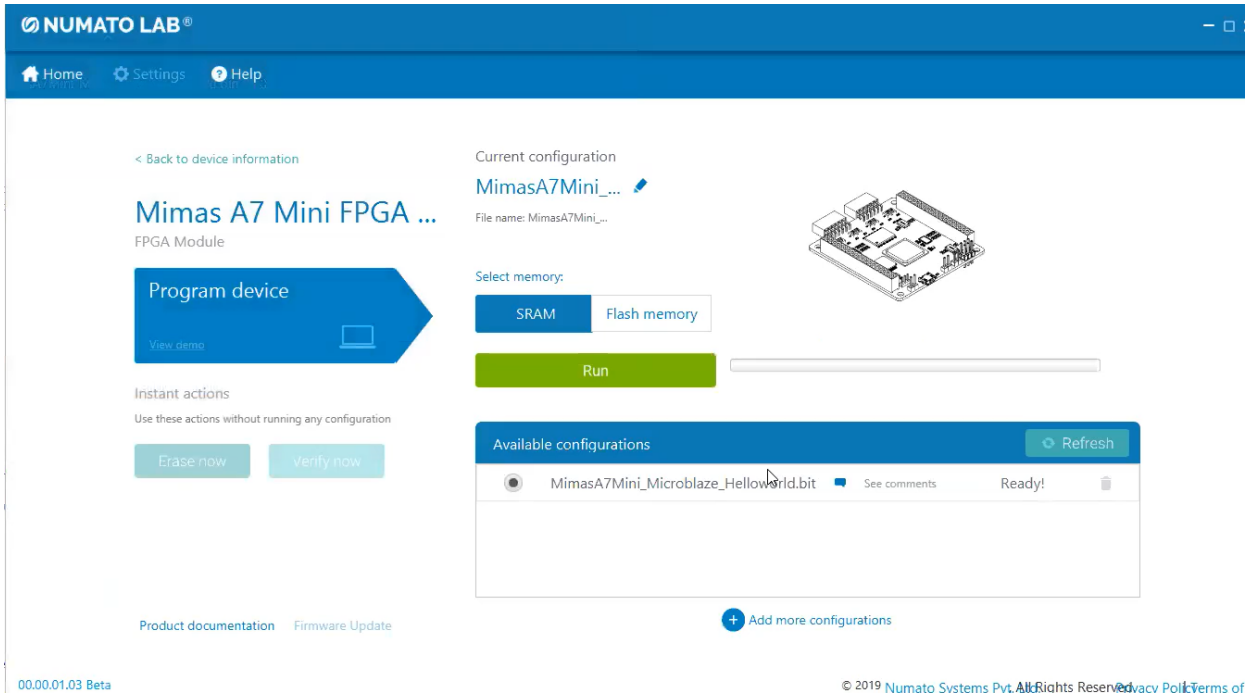
We could have used Xilinx jtag , a Xilinx platform cable, as it is quite expensive, we will not be using it rather we will using either of the two ways, XVC Server or Program device.



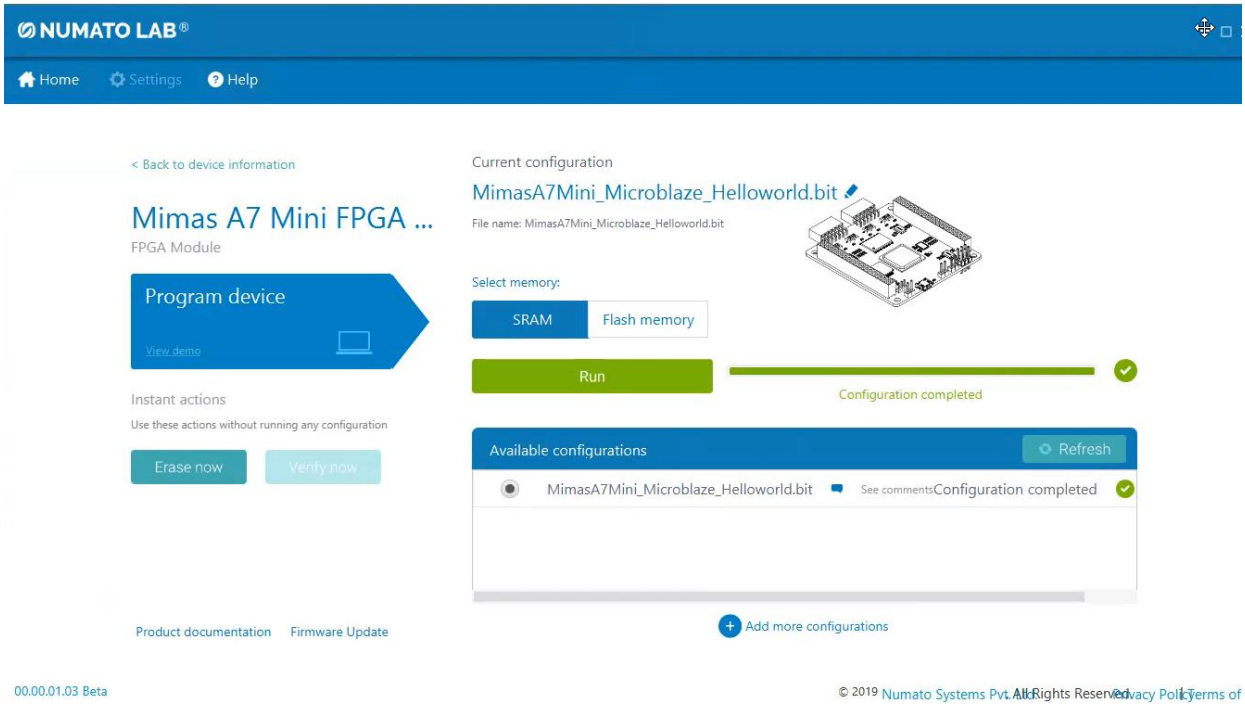
XVC server is nothing but Xilinx virtual cable, you can research and find out what it is. So, this is one way another way which we will be using to program is by an integrated circuit, FT2232HQ an onboard chip already available on Mimas A7 Mini, used for usb communication usb 2.0 communication not usb 3.0 with a speed of 480Mb/s. So, if we conclude we are using normal option that is program device to install the program on the FPGA.



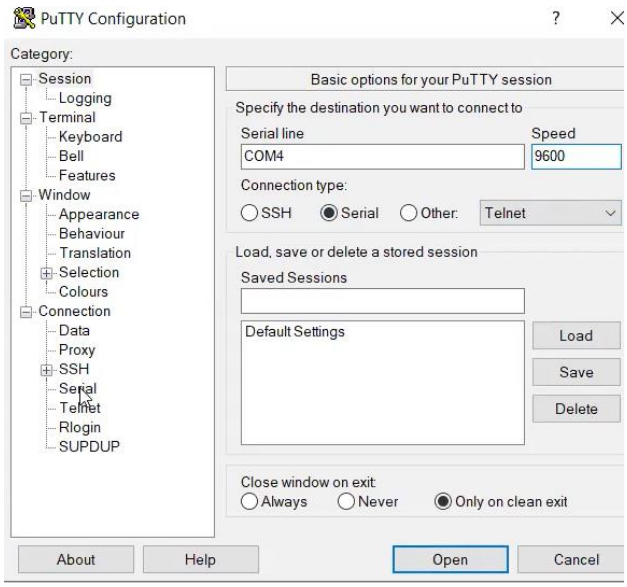
Add the bit file saved from Xilinx vivado integrated with this FPGA.

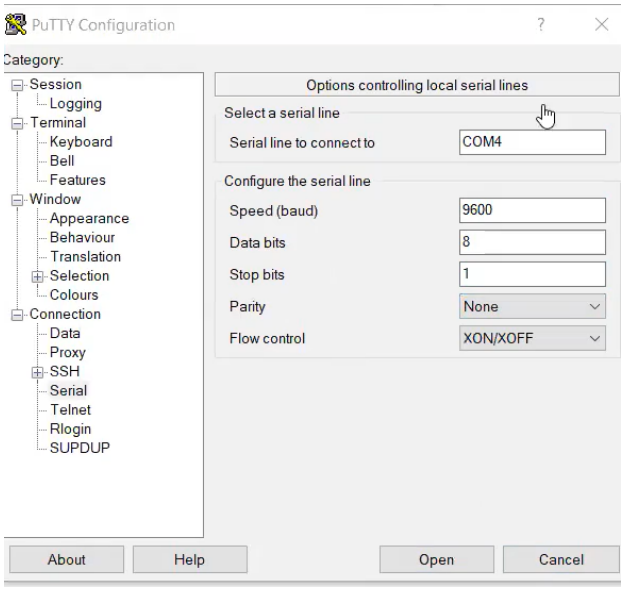


Run and configure the board with the program and after that if we have to launch putty software.

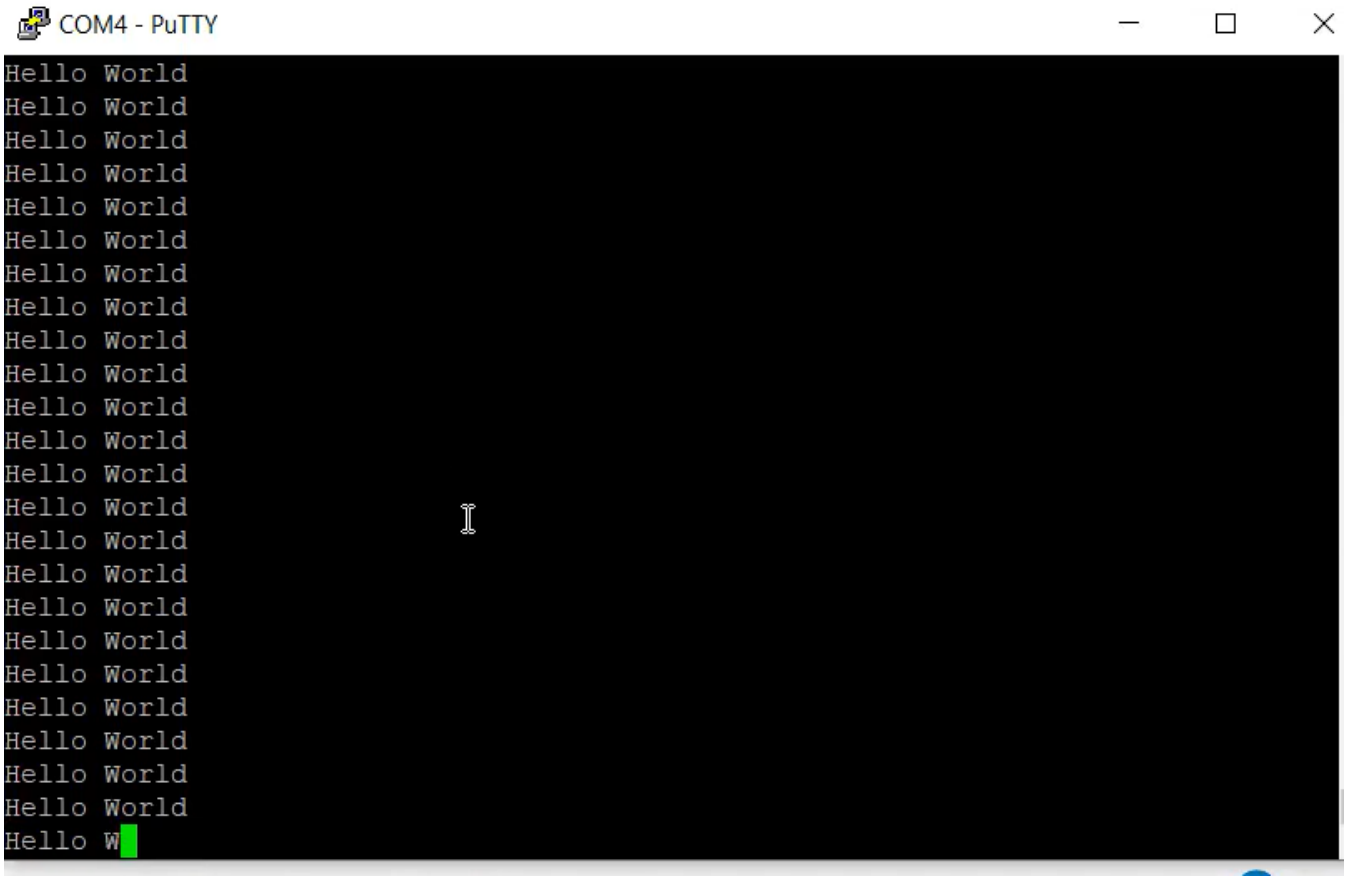


Step 2- Launch putty our serial monitors where it will receive data and make necessary changes with respect to port and connection type and so on.





Putty session will continuously print Hello world after we upload the program to our FPGA and even, we can stop it from happening with a reset button present on the board.



Open the C file where the program for hello world is written.

helloworld.c - Notepad

File Edit Format View Help

```
/******************************************************************************
*
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* Use of the Software is limited solely to applications:
* (a) running on a Xilinx device, or
* (b) that interact with a Xilinx device through a bus or interconnect.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
```

helloworld.c - Notepad

File Edit Format View Help

```
* (a) running on a Xilinx device, or
* (b) that interact with a Xilinx device through a bus or interconnect.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
* XILINX  BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* Except as contained in this notice, the name of the Xilinx shall not be used
* in advertising or otherwise to promote the sale, use or other dealings in
* this Software without prior written authorization from Xilinx.
*
******************************************************************************/

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
```

helloworld.c - Notepad

File Edit Format View Help

```
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *  uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
   while(1)
   {
```

helloworld.c - Notepad

File Edit Format View Help

```
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
   while(1)
   {

   print("Hello World\n\r");
   }
}
```

Things to know before proceeding,

Sequential circuits

Slide Show
Resume Slide Show

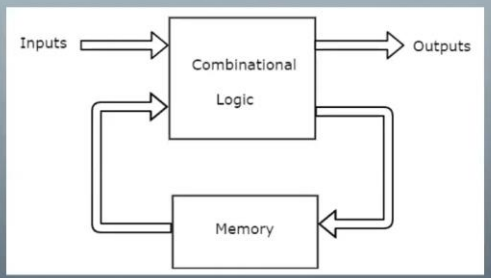
- Finite State Machines
  - The behavior of synchronous sequential circuits can be represented in the graphical form and it is known as **state diagram**.
  - Decision making logic and Control of the Digital System.
  - There are two types of FSMs.
    - Mealy State Machine
    - Moore State Machine

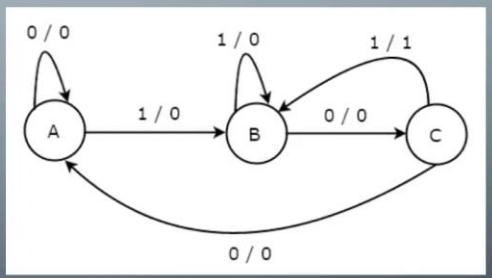
Sequential circuits

Slide Show
Resume Slide Show

- Finite State Machines
  - Mealy State Machine

    A Finite State Machine is said to be Mealy state machine, if outputs depend on both present inputs & present states.

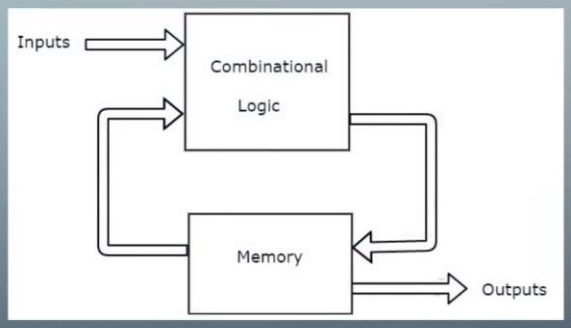


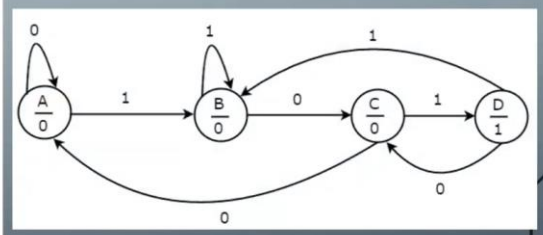


Sequential circuits

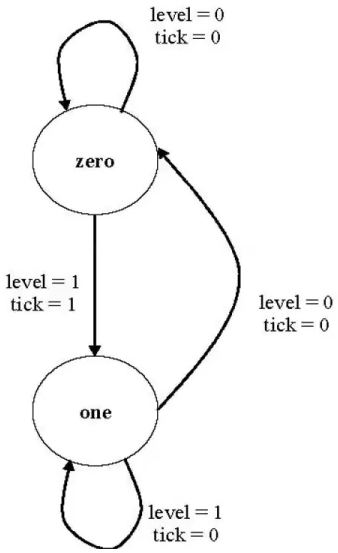
- Finite State Machines
  - Moore State Machine

    A Finite State Machine is said to be Moore state machine, if outputs depend only on present states.

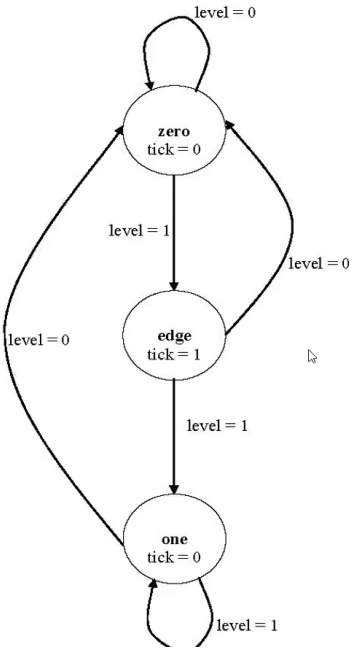




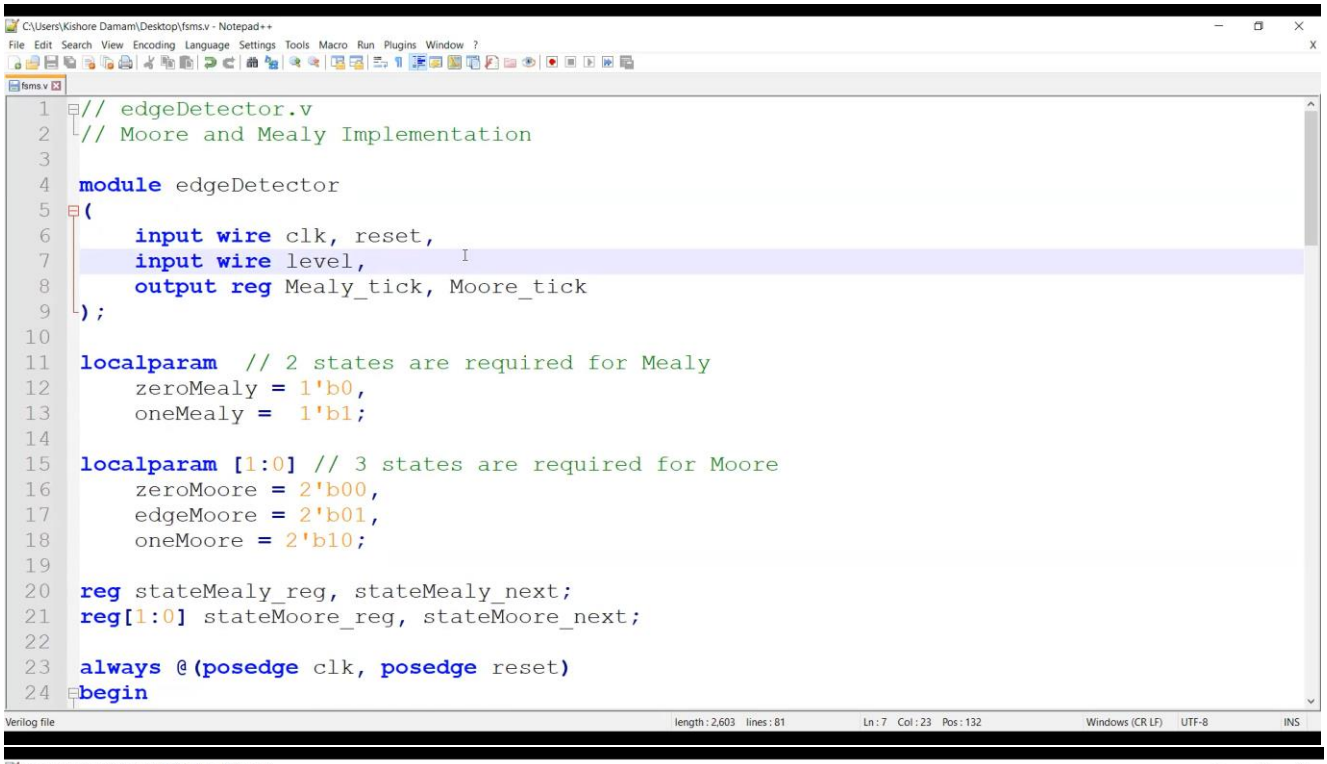
Mealy state machines



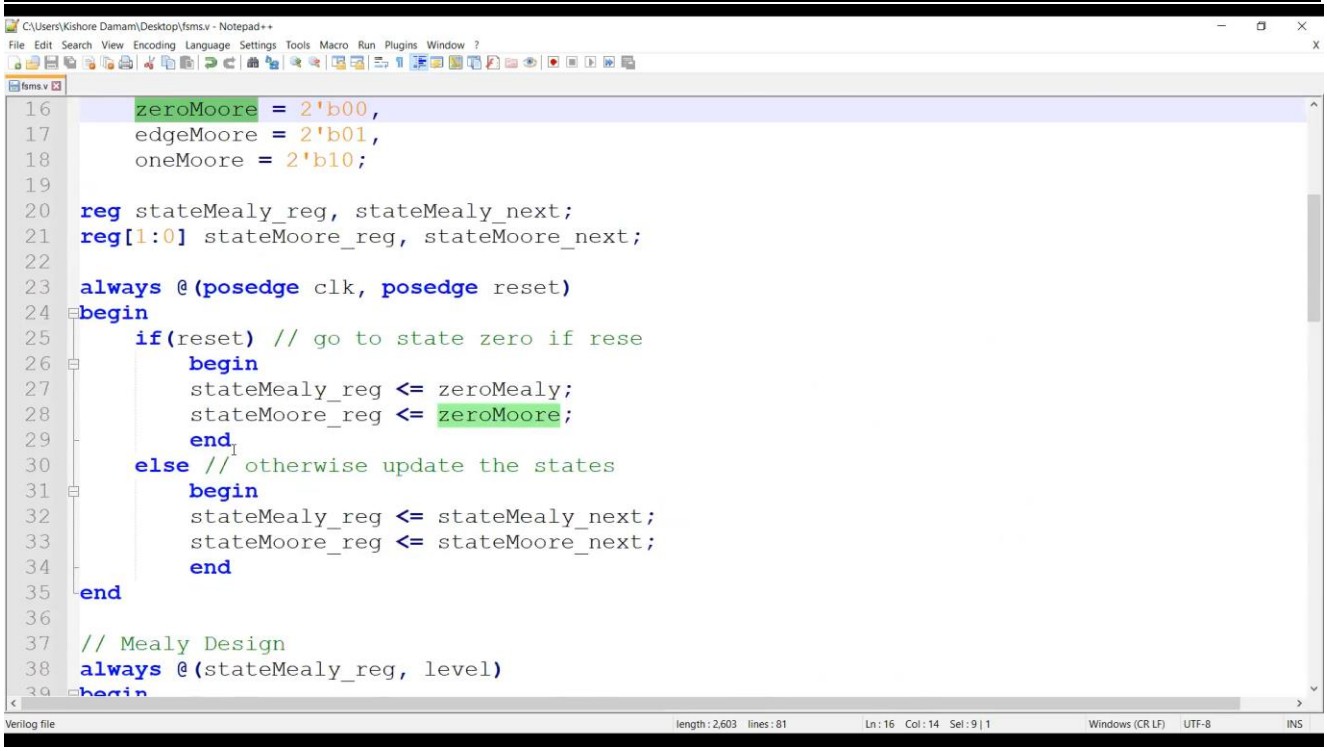
Moore state machines



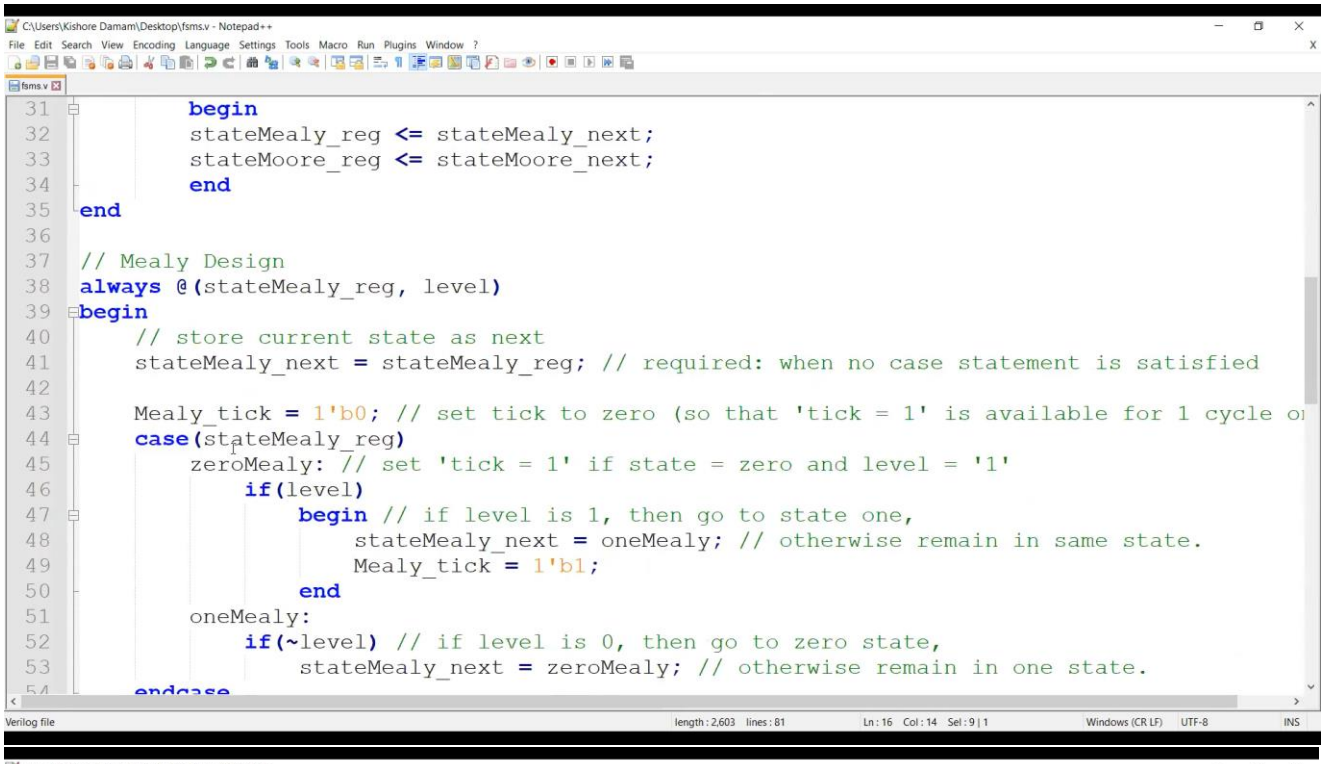
Step 3- Open notepad to write the Verilog code, a code to explain the basics of mealy and Moore state machine

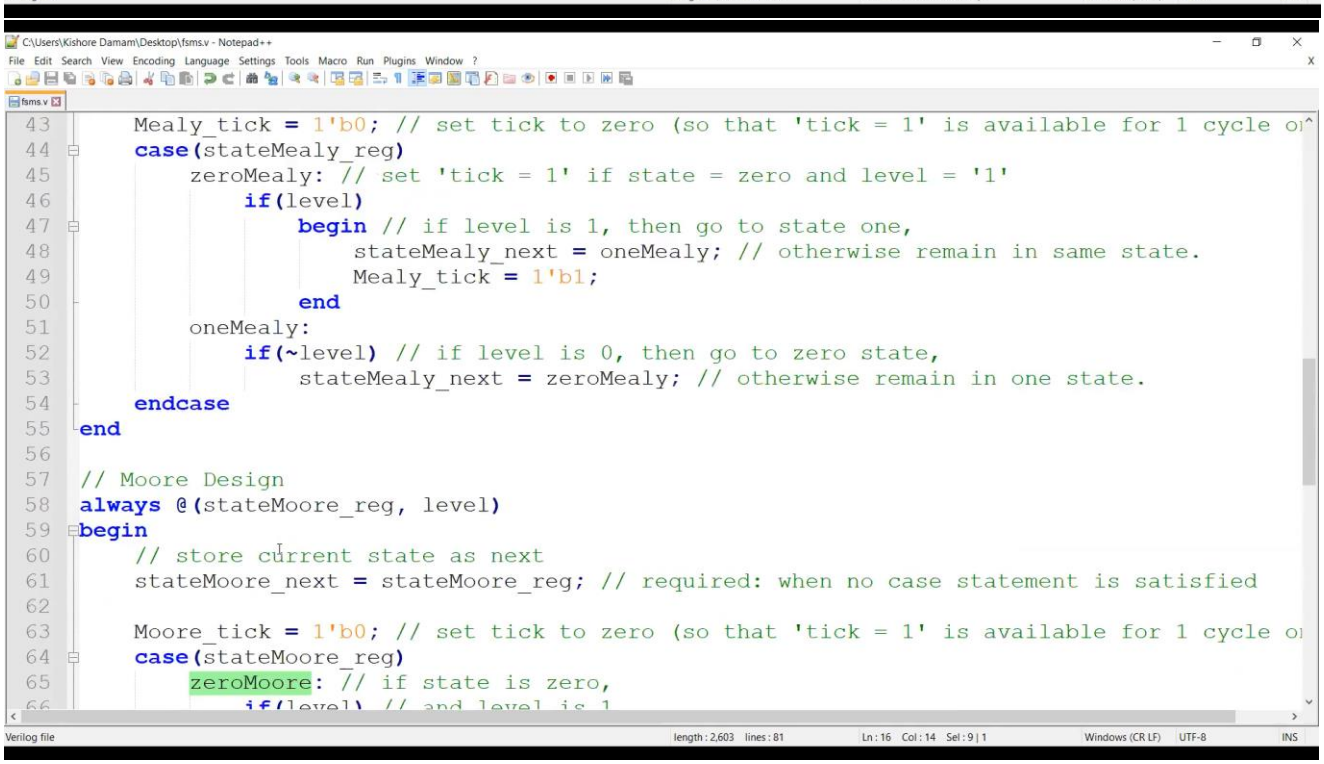


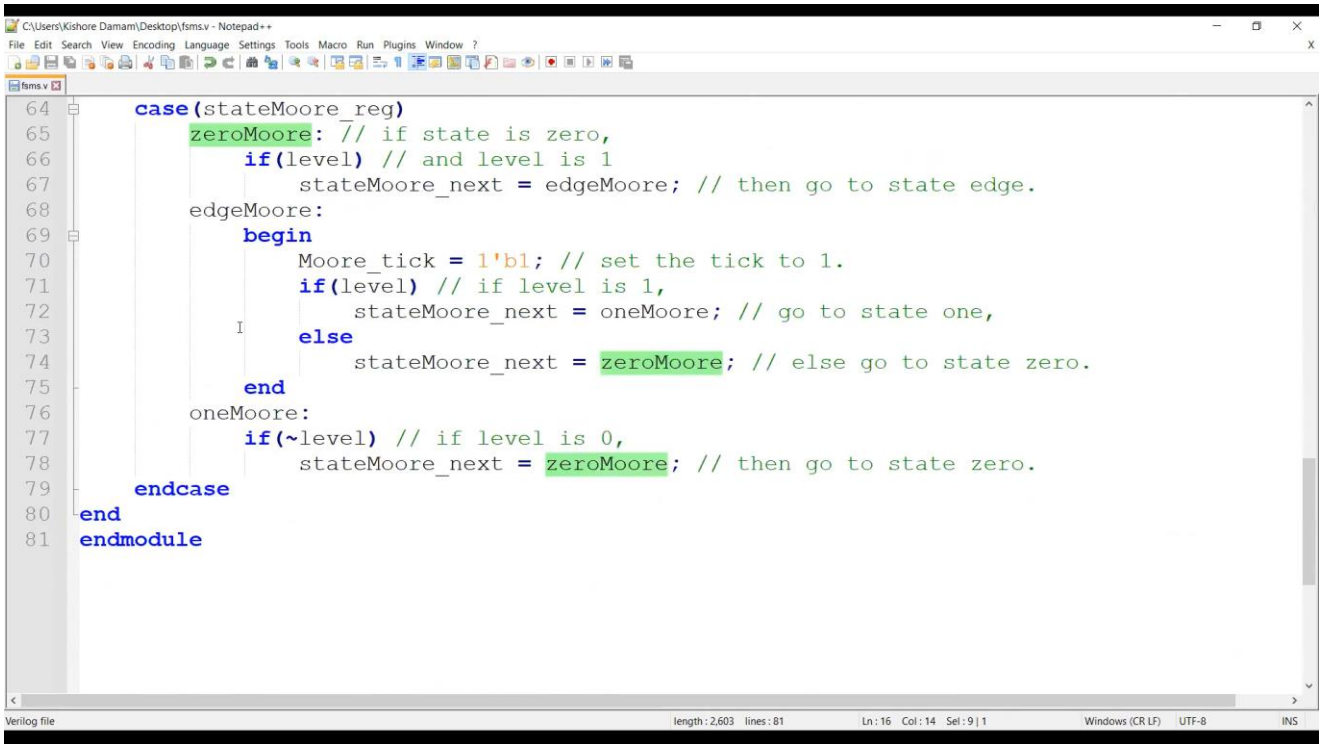
```
1  // edgeDetector.v
2  // Moore and Mealy Implementation
3
4  module edgeDetector
5  (
6      input wire clk, reset,
7      input wire level,
8      output reg Mealy_tick, Moore_tick
9  );
10
11 localparam  // 2 states are required for Mealy
12     zeroMealy = 1'b0,
13     oneMealy =  1'b1;
14
15 localparam [1:0] // 3 states are required for Moore
16     zeroMoore = 2'b00,
17     edgeMoore = 2'b01,
18     oneMoore = 2'b10;
19
20 reg stateMealy_reg, stateMealy_next;
21 reg[1:0] stateMoore_reg, stateMoore_next;
22
23 always @(posedge clk, posedge reset)
24 begin
```

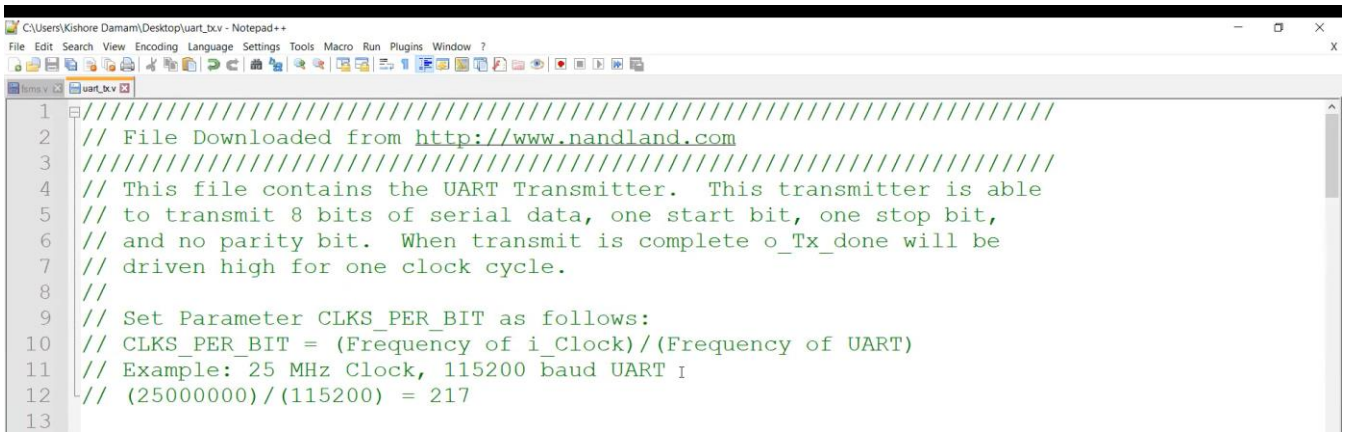
```
16     zeroMoore = 2'b00,
17     edgeMoore = 2'b01,
18     oneMoore = 2'b10;
19
20 reg stateMealy_reg, stateMealy_next;
21 reg[1:0] stateMoore_reg, stateMoore_next;
22
23 always @(posedge clk, posedge reset)
24 begin
25     if(reset) // go to state zero if rese
26         begin
27         stateMealy_reg <= zeroMealy;
28         stateMoore_reg <= zeroMoore;
29         end
30     else // otherwise update the states
31         begin
32         stateMealy_reg <= stateMealy_next;
33         stateMoore_reg <= stateMoore_next;
34         end
35 end
36
37 // Mealy Design
38 always @(stateMealy_reg, level)
39 begin
```

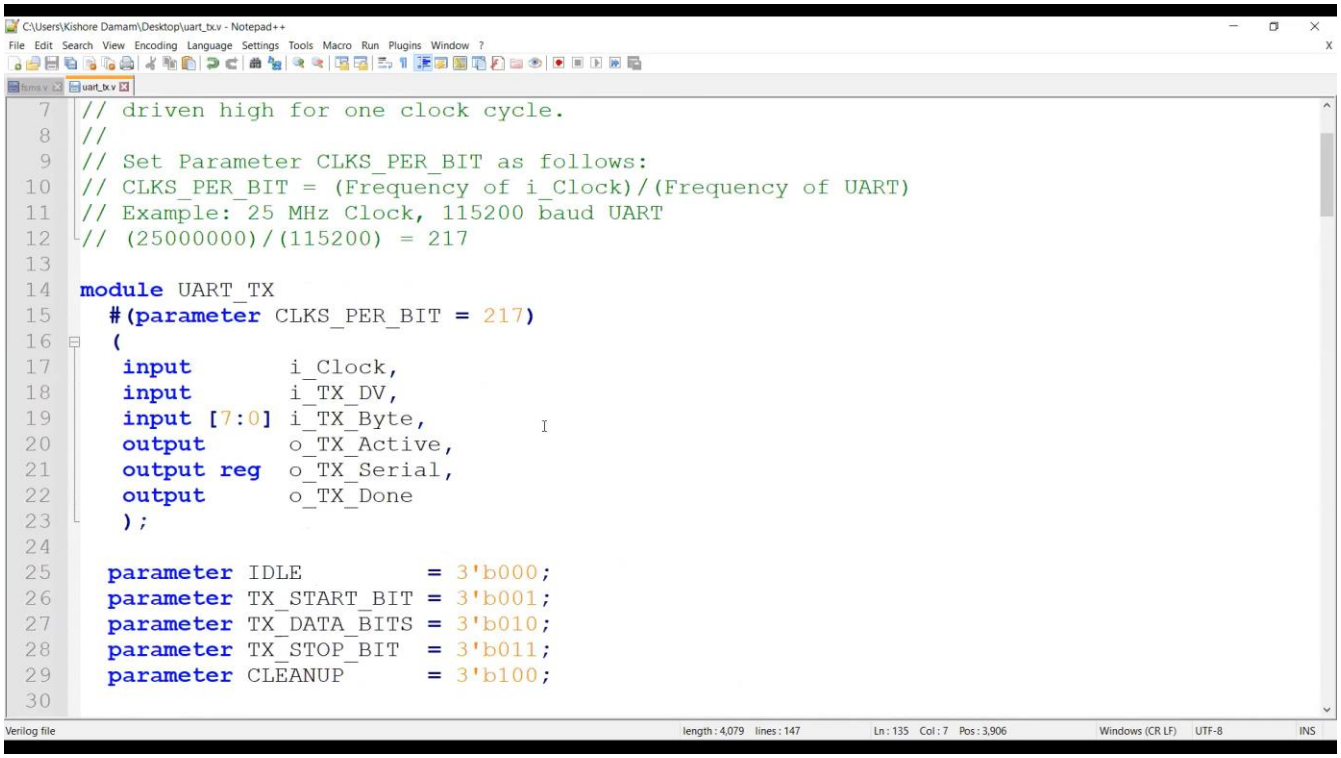

C:\Users\Kishore Damam\Desktop\fsms.v - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
fsms.v
begin
stateMealy_reg <= stateMealy_next;
stateMoore_reg <= stateMoore_next;
end
end
// Mealy Design
always @(stateMealy_reg, level)
begin
// store current state as next
stateMealy_next = stateMealy_reg; // required: when no case statement is satisfied
Mealy_tick = 1'b0; // set tick to zero (so that 'tick = 1' is available for 1 cycle o
case(stateMealy_reg)
zeroMealy: // set 'tick = 1' if state = zero and level = '1'
if(level)
begin // if level is 1, then go to state one,
stateMealy_next = oneMealy; // otherwise remain in same state.
Mealy_tick = 1'b1;
end
oneMealy:
if(~level) // if level is 0, then go to zero state,
stateMealy_next = zeroMealy; // otherwise remain in one state.
endcase
Verilog file
length : 2,603 lines : 81
Ln : 16 Col : 14 Sel : 9 | 1
Windows (CR LF)
UTF-8
INS


C:\Users\Kishore Damam\Desktop\fsms.v - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
fsms.v
Mealy_tick = 1'b0; // set tick to zero (so that 'tick = 1' is available for 1 cycle o
case(stateMealy_reg)
zeroMealy: // set 'tick = 1' if state = zero and level = '1'
if(level)
begin // if level is 1, then go to state one,
stateMealy_next = oneMealy; // otherwise remain in same state.
Mealy_tick = 1'b1;
end
oneMealy:
if(~level) // if level is 0, then go to zero state,
stateMealy_next = zeroMealy; // otherwise remain in one state.
endcase
end
// Moore Design
always @(stateMoore_reg, level)
begin
// store current state as next
stateMoore_next = stateMoore_reg; // required: when no case statement is satisfied
Moore_tick = 1'b0; // set tick to zero (so that 'tick = 1' is available for 1 cycle o
case(stateMoore_reg)
zeroMoore: // if state is zero,
Verilog file
length : 2,603 lines : 81
Ln : 16 Col : 14 Sel : 9 | 1
Windows (CR LF)
UTF-8
INS

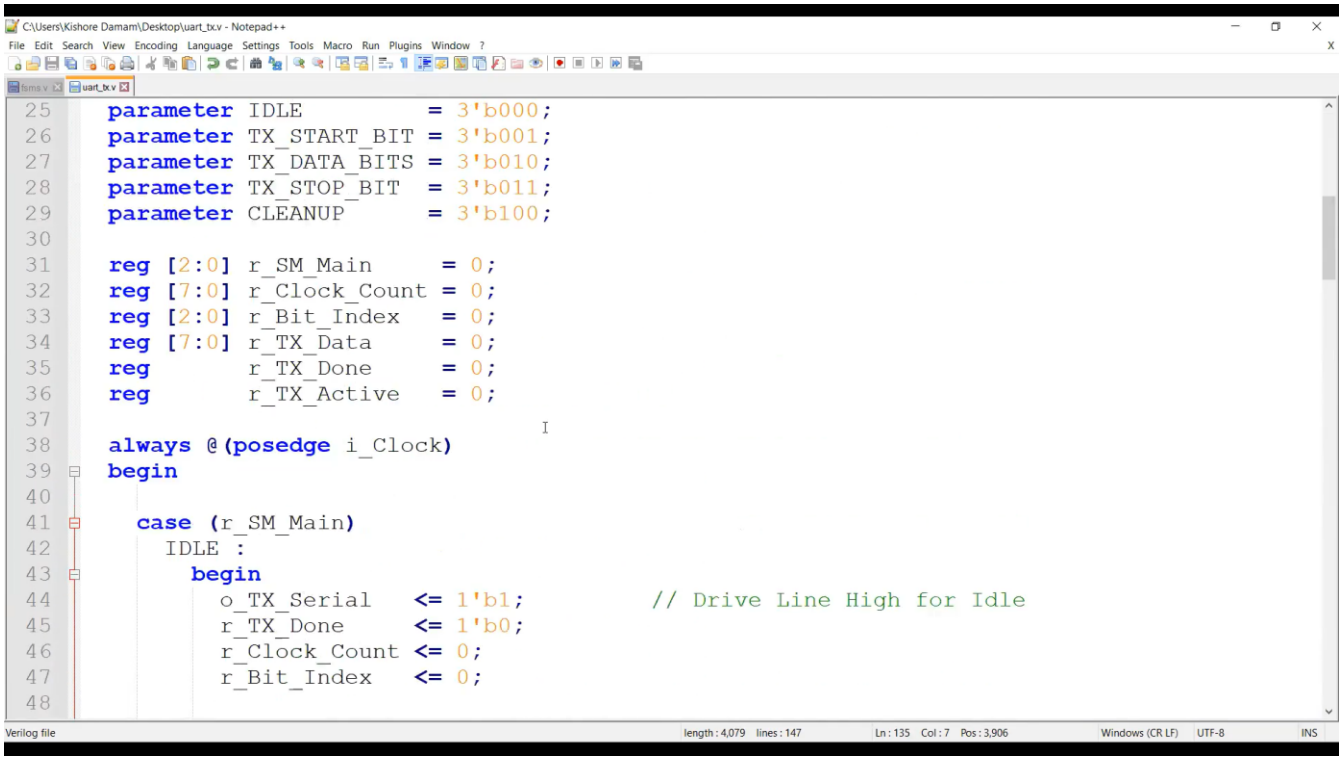


Open notepad to write the verilog code, a code to explain the UART module

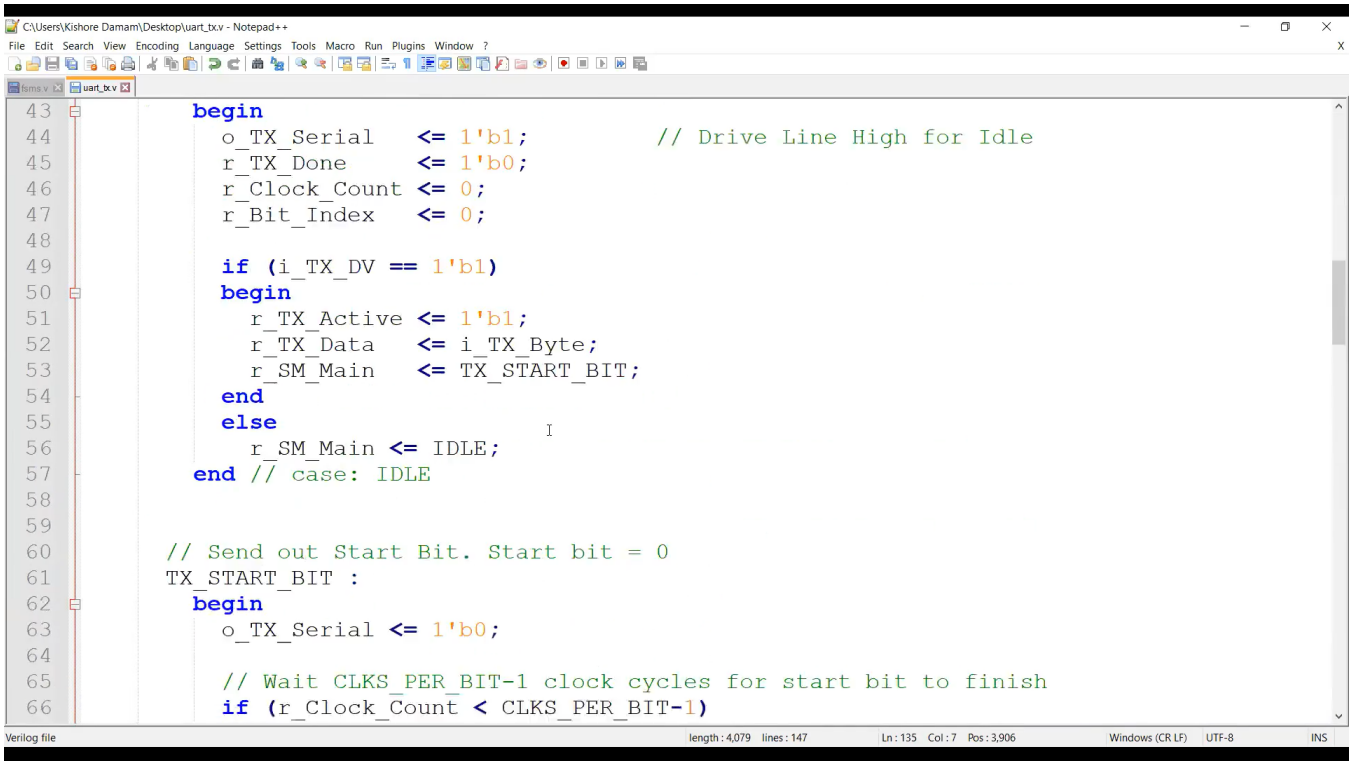


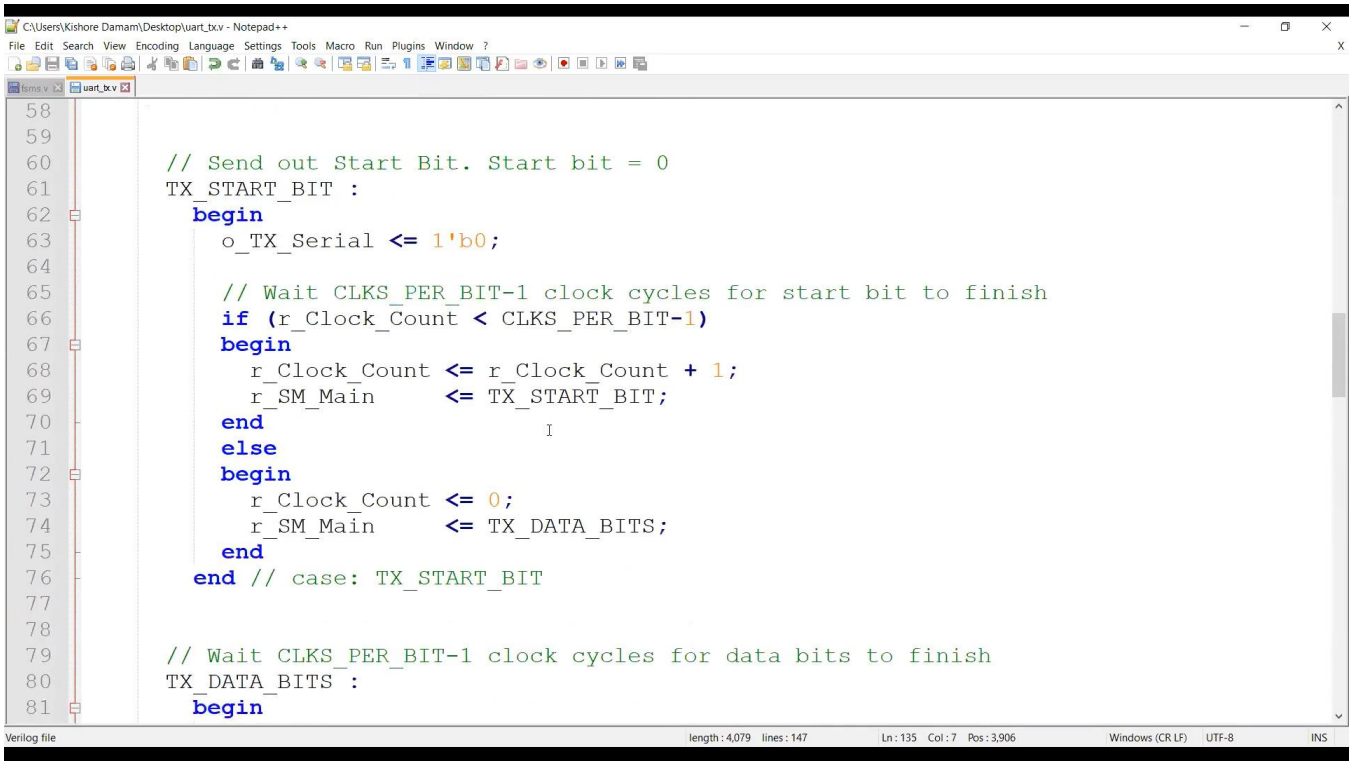


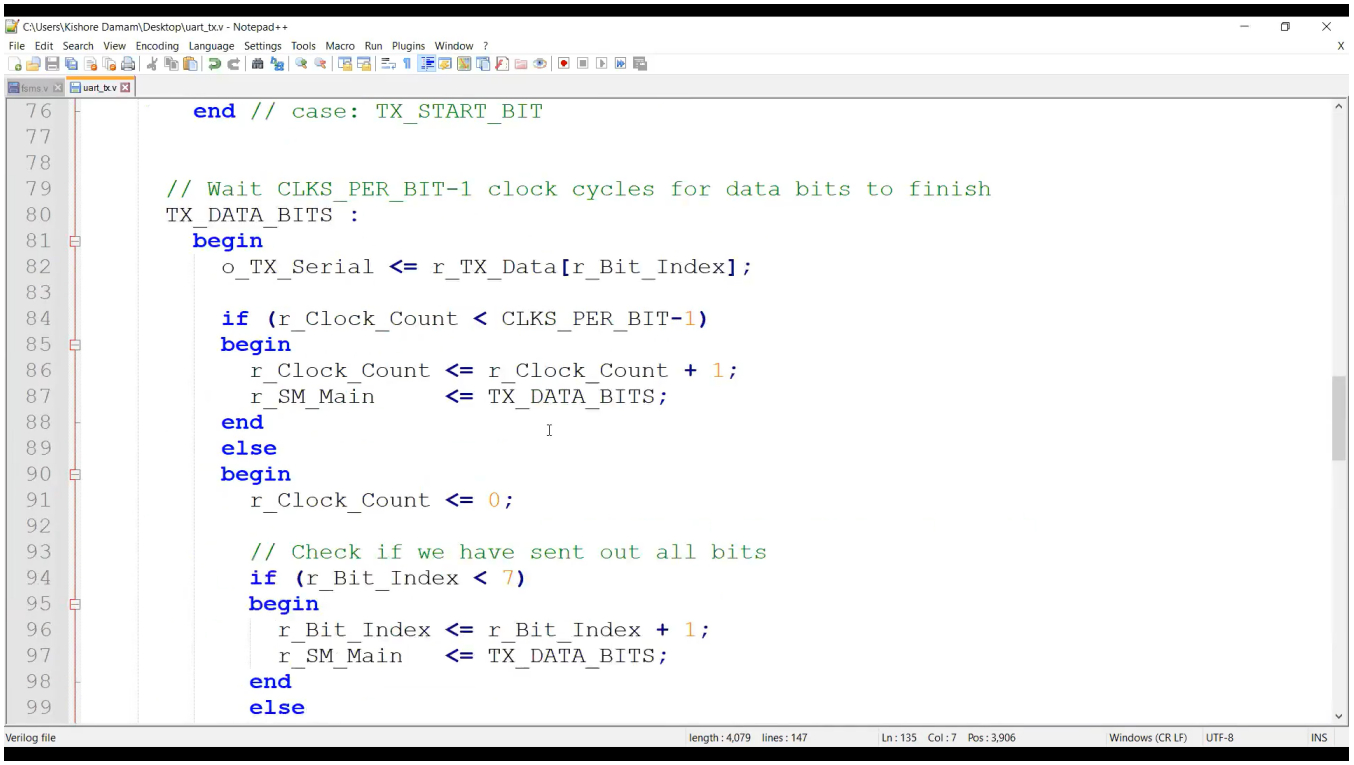
```
// driven high for one clock cycle.
//
// Set Parameter CLKS_PER_BIT as follows:
// CLKS_PER_BIT = (Frequency of i_Clock)/(Frequency of UART)
// Example: 25 MHz Clock, 115200 baud UART
// (25000000)/(115200) = 217

module UART_TX
  #(parameter CLKS_PER_BIT = 217)
  (
   input       i_Clock,
   input       i_TX_DV,
   input [7:0] i_TX_Byte,
   output      o_TX_Active,
   output reg  o_TX_Serial,
   output      o_TX_Done
   );

  parameter IDLE         = 3'b000;
  parameter TX_START_BIT = 3'b001;
  parameter TX_DATA_BITS = 3'b010;
  parameter TX_STOP_BIT  = 3'b011;
  parameter CLEANUP      = 3'b100;
```



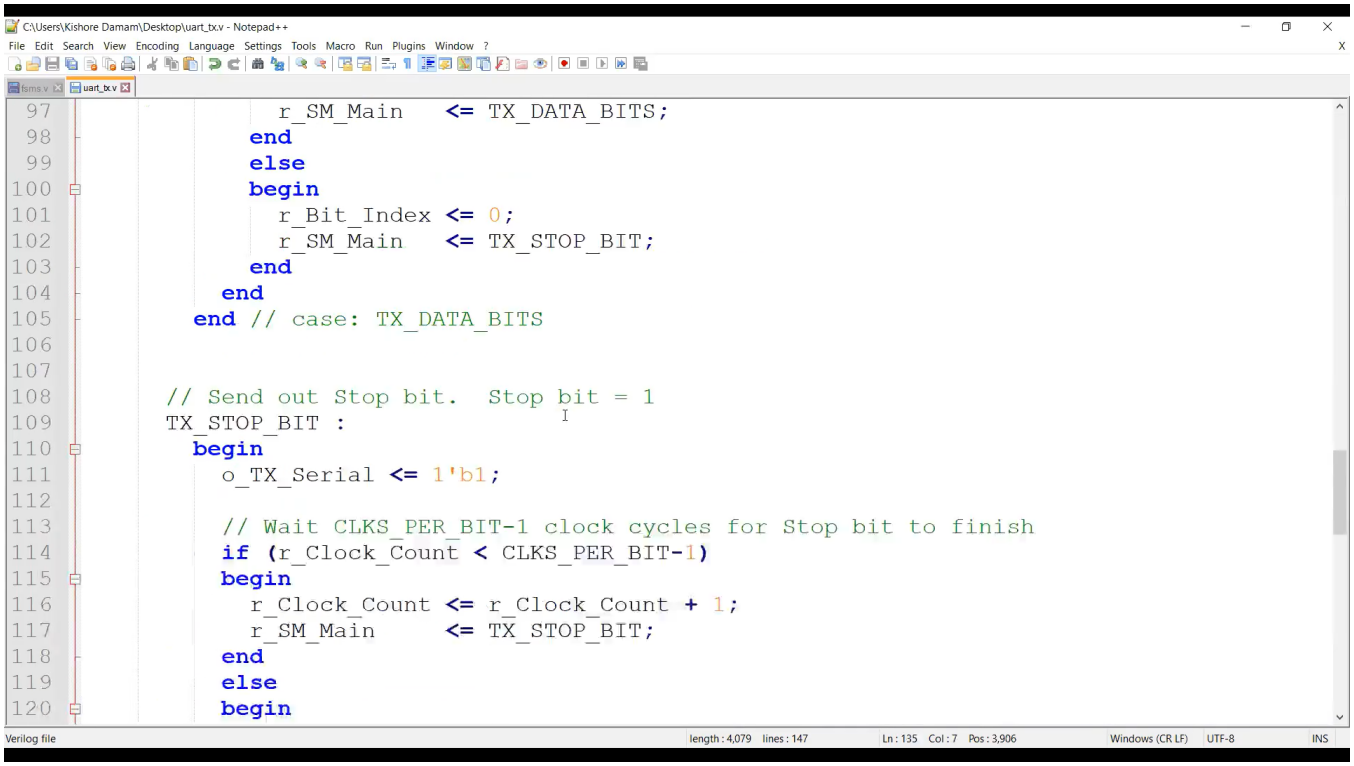
```
  parameter IDLE         = 3'b000;
  parameter TX_START_BIT = 3'b001;
  parameter TX_DATA_BITS = 3'b010;
  parameter TX_STOP_BIT  = 3'b011;
  parameter CLEANUP      = 3'b100;

  reg [2:0] r_SM_Main     = 0;
  reg [7:0] r_Clock_Count = 0;
  reg [2:0] r_Bit_Index   = 0;
  reg [7:0] r_TX_Data     = 0;
  reg       r_TX_Done     = 0;
  reg       r_TX_Active   = 0;

  always @(posedge i_Clock)
  begin

    case (r_SM_Main)
      IDLE :
        begin
          o_TX_Serial   <= 1'b1;         // Drive Line High for Idle
          r_TX_Done     <= 1'b0;
          r_Clock_Count <= 0;
          r_Bit_Index   <= 0;
```


C:\Users\Kishore Damam\Desktop\uart_tx.v - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
uart_tx.v
43 begin
44 o_TX_Serial <= 1'b1; // Drive Line High for Idle
45 r_TX_Done <= 1'b0;
46 r_Clock_Count <= 0;
47 r_Bit_Index <= 0;
48
49 if (i_TX_DV == 1'b1)
50 begin
51 r_TX_Active <= 1'b1;
52 r_TX_Data <= i_TX_Byte;
53 r_SM_Main <= TX_START_BIT;
54 end
55 else
56 r_SM_Main <= IDLE;
57 end // case: IDLE
58
59
60 // Send out Start Bit. Start bit = 0
61 TX_START_BIT :
62 begin
63 o_TX_Serial <= 1'b0;
64
65 // Wait CLKS_PER_BIT-1 clock cycles for start bit to finish
66 if (r_Clock_Count < CLKS_PER_BIT-1)
Verilog file
length : 4,079 lines : 147
Ln : 135 Col : 7 Pos : 3,906
Windows (CR LF)
UTF-8
INS


C:\Users\Kishore Damam\Desktop\uart_tx.v - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
uart_tx.v
58
59
60 // Send out Start Bit. Start bit = 0
61 TX_START_BIT :
62 begin
63 o_TX_Serial <= 1'b0;
64
65 // Wait CLKS_PER_BIT-1 clock cycles for start bit to finish
66 if (r_Clock_Count < CLKS_PER_BIT-1)
67 begin
68 r_Clock_Count <= r_Clock_Count + 1;
69 r_SM_Main <= TX_START_BIT;
70 end
71 else
72 begin
73 r_Clock_Count <= 0;
74 r_SM_Main <= TX_DATA_BITS;
75 end
76 end // case: TX_START_BIT
77
78
79 // Wait CLKS_PER_BIT-1 clock cycles for data bits to finish
80 TX_DATA_BITS :
81 begin
Verilog file
length : 4,079 lines : 147
Ln : 135 Col : 7 Pos : 3,906
Windows (CR LF)
UTF-8
INS



```
76          end // case: TX_START_BIT
77
78
79        // Wait CLKS_PER_BIT-1 clock cycles for data bits to finish
80        TX_DATA_BITS :
81          begin
82            o_TX_Serial <= r_TX_Data[r_Bit_Index];
83
84            if (r_Clock_Count < CLKS_PER_BIT-1)
85            begin
86              r_Clock_Count <= r_Clock_Count + 1;
87              r_SM_Main     <= TX_DATA_BITS;
88            end
89            else
90            begin
91              r_Clock_Count <= 0;
92
93              // Check if we have sent out all bits
94              if (r_Bit_Index < 7)
95              begin
96                r_Bit_Index <= r_Bit_Index + 1;
97                r_SM_Main   <= TX_DATA_BITS;
98              end
99              else
```



```
97                r_SM_Main   <= TX_DATA_BITS;
98              end
99              else
100             begin
101               r_Bit_Index <= 0;
102               r_SM_Main   <= TX_STOP_BIT;
103             end
104           end
105         end // case: TX_DATA_BITS
106
107
108       // Send out Stop bit.  Stop bit = 1
109       TX_STOP_BIT :
110         begin
111           o_TX_Serial <= 1'b1;
112
113           // Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
114           if (r_Clock_Count < CLKS_PER_BIT-1)
115           begin
116             r_Clock_Count <= r_Clock_Count + 1;
117             r_SM_Main     <= TX_STOP_BIT;
118           end
119           else
120           begin
```

```
124             r_TX_Active   <= 1'b0;
125           end
126         end // case: TX_STOP_BIT
127
128
129       // Stay here 1 clock
130       CLEANUP :
131         begin
132           r_TX_Done <= 1'b1;
133           r_SM_Main <= IDLE;
134         end
135
136
137       default :
138         r_SM_Main <= IDLE;
139
140     endcase
141   end
142
143   assign o_TX_Active = r_TX_Active;
144   assign o_TX_Done   = r_TX_Done;
145
146 endmodule
147
```

Output