**Name: Patel Virashree**

**Project #4: VGA**

Scope of the Project: The purpose of this project is to being able to display text using VGA on the monitor by using ROM.

Modules: ROM (Memory\_bb) , vga driver that provides timing and Shift register.

Results:

Compilation Report :



Time Quest analysis :





Waveforms: Here are the waveforms in ModelSim , it shows how shift register works and spits out bit by bit value on to the screen. We can see how the bits are printed on the screen line by line









Output on the Monitor: The monitor prints all the values that are in the Memory\_bb file i.e. ROM in different columns and then repeats the same in rows.



What did you learn: Of course learned how VGA works but also learned not to give up :D

**Code :**

// VGA driver design

// File vga.v

// Generates VGA timing and data signals to drive a VGA monitor

// in 640 x 480 mode

// Patel Virashree

// April 9 , 2015

module VGA(CLOCK\_50, ar, VGA\_R, VGA\_G, VGA\_B, VGA\_HS, VGA\_VS, VGA\_BLANK, VGA\_SYNC, VGA\_CLK);

 input CLOCK\_50; // 50 MHz input clock

 input ar; // Active low asynchronous reset

 output [9:0] VGA\_R, VGA\_G, VGA\_B; // Red, green, and blue outputs

 output VGA\_HS; // Output horizontal synch

 output VGA\_VS; // Output vertical synch

 output VGA\_BLANK, VGA\_SYNC;

 output VGA\_CLK;

 reg r, g, b;

 reg hsync, vsync;

 reg clk; // 25 MHz clock

 reg [9:0] hcount, vcount; // Counters used for creating the vsync and hsync signals

 reg video\_on\_h, video\_on\_v;

 assign VGA\_BLANK = 1'b1;//hsync & vsync;

 assign VGA\_SYNC = 1'b1;

 assign VGA\_HS = hsync;

 assign VGA\_VS = vsync;

 assign VGA\_R = {1'b0, r, 7'b0};

 assign VGA\_G = {1'b0, g, 7'b0};

 assign VGA\_B = {1'b0, b, 7'b0};

 parameter [9:0] h\_max = 10'd799;

 parameter [9:0] v\_max = 10'd524;

 reg [7:0]shift;

 reg count ;

 reg load ;

 wire [8:0] rom\_ad;

 wire [7:0] rom\_data;

 Memory\_bb rom (.address(rom\_ad) , .clock(CLOCK\_50) , .q(rom\_data));

 assign rom\_ad = {hcount[8:3] , vcount[2] , vcount[1] , vcount[0]};

 assign y = shift[7];

 always @(negedge ar or posedge CLOCK\_50)

 if(~ar)

 clk = 1'b0;

 else

 clk = ~clk;

 assign VGA\_CLK = clk;

 // Generate the hcount & hsync

 //initial

 // begin

 // load = 0;

 // count = 0;

 //end

 always @(negedge ar or posedge clk)

 if(~ar)

 begin

 hcount = 10'b0;

 video\_on\_h = 1'b0;

 end

 else

 begin

 if(hcount >= h\_max)

 hcount = 10'b0;

 else

 hcount = hcount + 1;

 if(hcount <= 755 && hcount >= 659)

 hsync = 1'b0;

 else

 hsync = 1'b1;

 if(hcount <= 639)

 video\_on\_h = 1'b1;

 else

 video\_on\_h = 1'b0;

 end // else: !if(~ar)

 always @(negedge ar or posedge clk)

 if(~ar)

 begin

 vcount = 10'b0;

 video\_on\_v = 1'b0;

 end

 else

 begin

 if(vcount >= v\_max)

 vcount = 10'b0;

 else

 if(hcount == 10'd699)

 vcount = vcount + 1;

 if(vcount <= 494 && vcount >= 493)

 vsync = 1'b0;

 else

 vsync = 1'b1;

 if(vcount <= 479)

 video\_on\_v = 1'b1;

 else

 video\_on\_v = 1'b0;

 end // else: !if(~ar)

 always @ (posedge clk or negedge ar)

 begin

 if(~ar) begin

 shift = 8'b00000000;

 //count = 0;

 end

 else if(hcount[2:0] == 3'b111)

 begin

 shift = rom\_data;

 end

 else

 shift = shift<<1;//{shift[7:1] , 1'b0};

 end

 // Generate R,G,B outputs for a colorbar pattern (just use the hcount position)

 always @(negedge ar or posedge clk)

 if(~ar)

 begin

 r = 1'b0;

 g = 1'b0;

 b = 1'b0;

 end

 else

 begin

 if(video\_on\_h && video\_on\_v) // Allow non-zero RGB only within the 640x480 area

 begin

 r = shift[7];

 g = r;

 b = g;

 //r = hcount[8];

 //g = hcount[7];

 //b = hcount[6];

 end

 else

 begin

 r = 1'b0;

 g = 1'b0;

 b = 1'b0;

 end // else: !if(video\_on\_v & video\_on\_h)

 end // else: !if(~ar)

endmodule // vga

**Memory\_bb.v**

// megafunction wizard: %ROM: 1-PORT%

// GENERATION: STANDARD

// VERSION: WM1.0

// MODULE: altsyncram

// ============================================================

// File Name: Memory\_bb.v

// Megafunction Name(s):

// altsyncram

//

// Simulation Library Files(s):

// altera\_mf

// ============================================================

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!

//

// 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Copyright (C) 1991-2013 Altera Corporation

//Your use of Altera Corporation's design tools, logic functions

//and other software and tools, and its AMPP partner logic

//functions, and any output files from any of the foregoing

//(including device programming or simulation files), and any

//associated documentation or information are expressly subject

//to the terms and conditions of the Altera Program License

//Subscription Agreement, Altera MegaCore Function License

//Agreement, or other applicable license agreement, including,

//without limitation, that your use is for the sole purpose of

//programming logic devices manufactured by Altera and sold by

//Altera or its authorized distributors. Please refer to the

//applicable agreement for further details.

// synopsys translate\_off

`timescale 1 ps / 1 ps

// synopsys translate\_on

module Memory\_bb (

 address,

 clock,

 q);

 input [8:0] address;

 input clock;

 output [7:0] q;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_off

`endif

 tri1 clock;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_on

`endif

 wire [7:0] sub\_wire0;

 wire [7:0] q = sub\_wire0[7:0];

 altsyncram altsyncram\_component (

 .address\_a (address),

 .clock0 (clock),

 .q\_a (sub\_wire0),

 .aclr0 (1'b0),

 .aclr1 (1'b0),

 .address\_b (1'b1),

 .addressstall\_a (1'b0),

 .addressstall\_b (1'b0),

 .byteena\_a (1'b1),

 .byteena\_b (1'b1),

 .clock1 (1'b1),

 .clocken0 (1'b1),

 .clocken1 (1'b1),

 .clocken2 (1'b1),

 .clocken3 (1'b1),

 .data\_a ({8{1'b1}}),

 .data\_b (1'b1),

 .eccstatus (),

 .q\_b (),

 .rden\_a (1'b1),

 .rden\_b (1'b1),

 .wren\_a (1'b0),

 .wren\_b (1'b0));

 defparam

 altsyncram\_component.clock\_enable\_input\_a = "BYPASS",

 altsyncram\_component.clock\_enable\_output\_a = "BYPASS",

 altsyncram\_component.init\_file = "tcgrom.mif",

 altsyncram\_component.intended\_device\_family = "Cyclone II",

 altsyncram\_component.lpm\_hint = "ENABLE\_RUNTIME\_MOD=NO",

 altsyncram\_component.lpm\_type = "altsyncram",

 altsyncram\_component.numwords\_a = 512,

 altsyncram\_component.operation\_mode = "ROM",

 altsyncram\_component.outdata\_aclr\_a = "NONE",

 altsyncram\_component.outdata\_reg\_a = "CLOCK0",

 altsyncram\_component.widthad\_a = 9,

 altsyncram\_component.width\_a = 8,

 altsyncram\_component.width\_byteena\_a = 1;

endmodule

// ============================================================

// CNX file retrieval info

// ============================================================

// Retrieval info: PRIVATE: ADDRESSSTALL\_A NUMERIC "0"

// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"

// Retrieval info: PRIVATE: AclrByte NUMERIC "0"

// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"

// Retrieval info: PRIVATE: BYTE\_ENABLE NUMERIC "0"

// Retrieval info: PRIVATE: BYTE\_SIZE NUMERIC "8"

// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"

// Retrieval info: PRIVATE: CLOCK\_ENABLE\_INPUT\_A NUMERIC "0"

// Retrieval info: PRIVATE: CLOCK\_ENABLE\_OUTPUT\_A NUMERIC "0"

// Retrieval info: PRIVATE: Clken NUMERIC "0"

// Retrieval info: PRIVATE: IMPLEMENT\_IN\_LES NUMERIC "0"

// Retrieval info: PRIVATE: INIT\_FILE\_LAYOUT STRING "PORT\_A"

// Retrieval info: PRIVATE: INIT\_TO\_SIM\_X NUMERIC "0"

// Retrieval info: PRIVATE: INTENDED\_DEVICE\_FAMILY STRING "Cyclone II"

// Retrieval info: PRIVATE: JTAG\_ENABLED NUMERIC "0"

// Retrieval info: PRIVATE: JTAG\_ID STRING "NONE"

// Retrieval info: PRIVATE: MAXIMUM\_DEPTH NUMERIC "0"

// Retrieval info: PRIVATE: MIFfilename STRING "tcgrom.mif"

// Retrieval info: PRIVATE: NUMWORDS\_A NUMERIC "512"

// Retrieval info: PRIVATE: RAM\_BLOCK\_TYPE NUMERIC "0"

// Retrieval info: PRIVATE: RegAddr NUMERIC "1"

// Retrieval info: PRIVATE: RegOutput NUMERIC "1"

// Retrieval info: PRIVATE: SYNTH\_WRAPPER\_GEN\_POSTFIX STRING "0"

// Retrieval info: PRIVATE: SingleClock NUMERIC "1"

// Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"

// Retrieval info: PRIVATE: WidthAddr NUMERIC "9"

// Retrieval info: PRIVATE: WidthData NUMERIC "8"

// Retrieval info: PRIVATE: rden NUMERIC "0"

// Retrieval info: LIBRARY: altera\_mf altera\_mf.altera\_mf\_components.all

// Retrieval info: CONSTANT: CLOCK\_ENABLE\_INPUT\_A STRING "BYPASS"

// Retrieval info: CONSTANT: CLOCK\_ENABLE\_OUTPUT\_A STRING "BYPASS"

// Retrieval info: CONSTANT: INIT\_FILE STRING "tcgrom.mif"

// Retrieval info: CONSTANT: INTENDED\_DEVICE\_FAMILY STRING "Cyclone II"

// Retrieval info: CONSTANT: LPM\_HINT STRING "ENABLE\_RUNTIME\_MOD=NO"

// Retrieval info: CONSTANT: LPM\_TYPE STRING "altsyncram"

// Retrieval info: CONSTANT: NUMWORDS\_A NUMERIC "512"

// Retrieval info: CONSTANT: OPERATION\_MODE STRING "ROM"

// Retrieval info: CONSTANT: OUTDATA\_ACLR\_A STRING "NONE"

// Retrieval info: CONSTANT: OUTDATA\_REG\_A STRING "CLOCK0"

// Retrieval info: CONSTANT: WIDTHAD\_A NUMERIC "9"

// Retrieval info: CONSTANT: WIDTH\_A NUMERIC "8"

// Retrieval info: CONSTANT: WIDTH\_BYTEENA\_A NUMERIC "1"

// Retrieval info: USED\_PORT: address 0 0 9 0 INPUT NODEFVAL "address[8..0]"

// Retrieval info: USED\_PORT: clock 0 0 0 0 INPUT VCC "clock"

// Retrieval info: USED\_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"

// Retrieval info: CONNECT: @address\_a 0 0 9 0 address 0 0 9 0

// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0

// Retrieval info: CONNECT: q 0 0 8 0 @q\_a 0 0 8 0

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory.v TRUE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory.inc FALSE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory.cmp FALSE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory.bsf FALSE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory\_inst.v FALSE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory\_bb.v TRUE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory\_bb.inc FALSE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory\_bb.cmp FALSE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory\_bb.bsf FALSE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory\_bb\_inst.v FALSE

// Retrieval info: GEN\_FILE: TYPE\_NORMAL Memory\_bb\_bb.v TRUE

// Retrieval info: LIB\_FILE: altera\_mf

**vga\_tb.v**

// Testbench

// tb\_vga.v

// Don M. Gruenbacher

// Feb. 10, 2000

`timescale 10 ns / 1 ns

module vga\_tb;

 reg clk;

 reg ar;

 wire [9:0] r, g, b;

 wire hsync, vsync, blank, sync;

// instantiate the vga design

 VGA vga1 (.CLOCK\_50(clk), .ar(ar), .VGA\_R(r), .VGA\_G(g), .VGA\_B(b), .VGA\_HS(hsync), .VGA\_VS(vsync), .VGA\_BLANK(blank),

 .VGA\_SYNC(sync));

// vga(CLOCK\_50, ar, VGA\_R, VGA\_G, VGA\_B, VGA\_HS, VGA\_VS, VGA\_BLANK, VGA\_SYNC)

initial

 begin

 clk = 1'b0; // set to 0 so toggling can occur

 ar = 1'b1; // Start reset at 1

 #5 ar = 1'b0; // Set reset to 0 after 5 ns

 #20 ar = 1'b1; // Set reset to 1

 end

// Controls the test clock

always

 #1 clk = ~clk; // For 20 ns period (50 MHz)

endmodule