BILKENT UNIVERSITY

COMPUTER ENGINEERING

CS 223 DIGITAL DESIGN

HIT THE BALL

BURAK MANDIRA

21301474 - SECTION 1

ÖMER MESUD TOKER

21302479 - SECTION 1

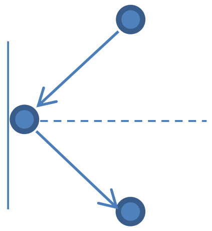
21.12.2015

Description of the Project

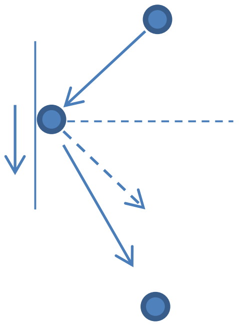
The project is a kind of two-player Power Pong game. There are two rackets which are controlled with BASYS 2 buttons. There are 4 of them so left two is for player 1 and others are for player 2. By pressing them rackets move up or down on the monitor according to pressed buttons. (Actually my partner and I thought a keyboard to control rackets but we changed our minds since we could not achieve to get two simultaneously pressed key scan code from PS2 keyboard.) Display of the game will be demonstrated on monitor and scores will be showed on 7-segment LED display unit.

The game starts with throwing the ball to first player. If rackets are not moving at the collision time with the ball, the ball bounces with the same angle (1). Otherwise, the ball bounces with a deviated angle associated with the direction of moving racket (2).

1



2



If a player cannot strike the ball, the opponent gets one point and the player who score up 7 points first wins. Scores are updated on the 2-digit 7-segment LED display whenever they change.

References to All the External Material and Sources Used in the Project

- http://io.smashthestack.org/arm/digitalcold/

Codes that are taken from this site are for displaying frames into VGA. We have used their codes only for VGA monitor display. We have implemented our own codes for displaying the ping pong game. So, we have changed most parts of codes that are taken from this source.

Clock_divider

It supplies us clocks whose frequencies are different. For 25 MHz clock it counts up to 2. For 200 MHz clock it counts up to 250000 and for 300 MHz clock it counts up to 166667.

Clock_shift

It shifts the clock by half a period (negates it). It just assigns clk_out to ~clk_in.

KeyboardReader

It is for getting the scan codes of W or S, and UP or DOWN keys to move the rackets of players. It outputs the scan code of pressed key.

Vga_driver

It is to display the game (rackets and ball). This module has an object of KeyboardReader so that according to pressed key it moves the rackets. And our with the help clock_divider object it displays the movement. It updates the paddles location according to the 300 MHz clock. Ball’s velocity is updated according to 200 MHz clock.

Implementation of the Project

```
// TOP MODULE
// Some of the codes below are taken from http://io.smashthestack.org/arm/digitalcold/. We took their codes for VGA
displaying.
// However, we have changed some parts of these codes to implement our project.
/* top.v - Top level module
        DigitalCold's ARM Challenge VGA Implementation
        Freefull's graphics demo reimplemented on the BASYS2 hardware.
        Additional information at http://io.smashthestack.org/arm/digitalcold/
Released 4/25/13
*/
module top(CLK_50MHz, VS, HS, RED, GREEN, BLUE, SWITCH, led, scanLED, CLK_PS2, DATA_PS2, B1, B2, B3, B4);
input CLK_50MHz;
input [2:0] SWITCH;
output VS;
output HS;
output [2:0] RED;
output [2:0] GREEN;
output [1:0] BLUE;

//keyboards
input CLK_PS2;
input DATA_PS2, B1, B2, B3, B4;
output [1:0] led;
output [7:0] scanLED;

//these signals are exported by the vga_driver
wire HBlank, VBlank;
wire [9:0] CurrentX;
wire [8:0] CurrentY;
wire CLK_DATA;
reg [7:0] DataIn = 0;

vga_driver vga(.CLK_50MHz(CLK_50MHz),
                .VS(VS),
                .HS(HS),
                .RED(RED),
                .GREEN(GREEN),
                .BLUE(BLUE),
                .VBLANK(VBlank),
                .HBLANK(HBlank),
                .CURX(CurrentX),
                .CURY(CurrentY),
                .CLK_DATA(CLK_DATA),
                .COLOR_DATA_IN(DataIn),
                .lpLEDs(led),
                .scanLEDS(scanLED),
                .clk(CLK_PS2),
                .data(DATA_PS2),
                .B1(B1),
                .B2(B2),
                .B3(B3),
                .B4(B4));

//so many damn flip-flops
```

```
reg [15:0] Time = 0;
reg [15:0] TimeTri = 0;

reg [23:0] TimeConst = 0;
reg [1:0] ColorSel = 0;

reg [31:0] pixel = 0;   //wasn't sure how much
reg [7:0] pixelTri = 0; //gradient pixel version

reg [9:0] NewX;
reg [8:0] NewY;
reg [9:0] Xored;
reg [15:0] Ysquared;

//time simulation
always @(posedge VBlank) begin
            if(SWITCH[2]) begin     //time "unlimited"
                if(SWITCH[1])       //2x speed using frame skipping
                    Time <= Time + 2;
                else            //normal
                    Time <= Time + 1;
            end
            else begin
                if(SWITCH[1])               //2x speed with frame skipping (limited)
                    Time <= (Time + 2) & 'hFF;
                else                    //normal (limited)
                    Time <= (Time + 1) & 'hFF;
            end

            if(SWITCH[0]) //stopped
                Time <= Time;
end

//turn time in to a triangle wave /\/\/\/\/\ (NOTICE negedge)
always @(negedge VBlank) begin
            if(Time >= 'h80)
                TimeTri <= 'hFF - Time;
            else
                TimeTri <= Time;
end

//calculate the TimeConst to be used for this frame
always @(posedge VBlank) begin
            TimeConst <= 'h200 - (TimeTri << 3);
end

//new scanline: calculate our new Y value
always @(posedge HBlank) begin //the -1 is to remove mirroring ugliness
            NewY = (CurrentY < 240 ) ? CurrentY : 480 - CurrentY - 1;
            Ysquared = NewY*NewY;
end

//new pixel: calculate our new X value and the pixel value
always @(posedge CLK_DATA) begin
            NewX = (CurrentX < 320) ? CurrentX : 640 - CurrentX - 1;
            Xored = NewX ^ NewY;

            //Yuck, but it meets timing! Wahooo!
            pixel = ((TimeConst + Xored)*Xored + Ysquared) >> 8;
            ColorSel = (pixel >> 8) & 2'b11;
            pixel = pixel & 'hff;

            //Make the pixel turn in to a gradient
```

```
                if(pixel >= 'h80)
                        pixelTri = 'hFF - pixel;
                else
                        pixelTri = pixel;

                if(!ColorSel)
                        DataIn = ((pixelTri) << 1) & 8'b11100000;
                else if(ColorSel == 1)
                        DataIn = ((pixelTri) >> 2) & 5'b11100;
                else
                        DataIn = ((pixelTri) >> 5) & 2'b11;
end

//ram_interface interface();
endmodule

/*******************************************************************************************/

// VGA_DRIVER
// Again, some parts of these codes are taken from http://io.smashthestack.org/arm/digitalcold
// but we have added some codes to display our game in the monitor.
/* vga_driver.v

    DigitalCold's ARM Challenge VGA Implementation
            Freefull's graphics demo reimplemented on the BASYS2 hardware.
            Additional information at http://io.smashthestack.org/arm/digitalcold/

    Released 4/25/13
*/

module vga_driver(CLK_50MHz, VS, HS, RED, GREEN, BLUE, HBLANK, VBLANK, CURX, CURY, CLK_DATA,
COLOR_DATA_IN, lpLEDs, scanLEDS, clk, data, B1, B2, B3, B4);
    //##### IO declarations
    input CLK_50MHz;
    output VS;
    output HS;
    output [2:0] RED;
    output [2:0] GREEN;
    output [1:0] BLUE;
    output HBLANK;
    output VBLANK;

    reg VS = 0; //vsync
    reg HS = 0; //hsync

    //client connection I/O
    input [7:0] COLOR_DATA_IN;
    output CLK_DATA;
    output [9:0] CURX;
    output [8:0] CURY;

    // from KeyboardReader
    wire [1:0] LP_DIR, RP_DIR;
    wire [7:0] SCAN_CODE;
    input clk, data, B1, B2, B3, B4;
    output [1:0] lpLEDs;
    output [7:0] scanLEDS;

    assign lpLEDs = LP_DIR;
    assign scanLEDS = SCAN_CODE;

    KeyboardReader kb( .lp_dir(LP_DIR), .rp_dir(RP_DIR), .scan_code(SCAN_CODE), .clk(clk), .data(data),
.CLK50(CLK_50MHz),
                                              .B1(B1), .B2(B2), .B3(B3), .B4(B4));

	//##### Module constants (http://tinyvga.com/vga-timing/640x480@60Hz)
	parameter HDisplayArea = 640;  // horizontal display area
	parameter HLimit = 800;        // maximum horizontal amount (limit)
	parameter HFrontPorch = 16;                // h. front porch
	parameter HBackPorch = 48;                 // h. back porch
	parameter HSyncWidth = 96;                 // h. pulse width

	parameter VDisplayArea = 480;  // vertical display area
	parameter VLimit = 525;                    // maximum vertical amount (limit)
	parameter VFrontPorch = 10;                // v. front porch
	parameter VBackPorch = 33;                 // v. back porch
	parameter VSyncWidth = 2;                  // v. pulse width

	//##### Local variables
	wire CLK_25MHz;
	wire CLK_200Hz;
	wire CLK_300Hz;

	reg [9:0] CurHPos = 0; //maximum of HLimit (2^10 - 1 = 1023)
	reg [9:0] CurVPos = 0; //maximum of VLimit
	reg HBlank, VBlank, Blank = 0;

	reg [9:0] CurrentX = 0;    //maximum of HDisplayArea
	reg [8:0] CurrentY = 0;    //maximum of VDisplayArea (2^9 - 1 = 511)

	// Ball's Borders
	reg [9:0] BallminX = 315;
	reg [9:0] BallmaxX = 325;
	reg [8:0] BallminY = 235;
	reg [8:0] BallmaxY = 245;

	// Left Paddle's Borders
	reg [9:0] LPminX = 40;
	reg [9:0] LPmaxX = 50;
	reg [8:0] LPminY = 160;
	reg [8:0] LPmaxY = 320;

	// Right Paddle's Borders
	reg [9:0] RPminX = 630;
	reg [9:0] RPmaxX = 640;
	reg [8:0] RPminY = 160;
	reg [8:0] RPmaxY = 320;

	//##### Submodule declaration
	clock_divider clk_div(.clk_in(CLK_50MHz), .clk_25mhz(CLK_25MHz), .clk_200hz(CLK_200Hz),
.clk_300hz(CLK_300Hz));

	//shifts the clock by half a period (negates it)
	//see timing diagrams for a better understanding of the reason for this
	clock_shift clk_shift(.clk_in(CLK_25MHz), .clk_out(CLK_DATA));

	//##### Procedural Code

	//simulate the vertical and horizontal positions
	always @(posedge CLK_25MHz) begin
		if(CurHPos < HLimit-1) begin
			CurHPos <= CurHPos + 1;
		end
		else begin
			CurHPos <= 0;
```

```
        if(CurVPos < VLimit-1)
            CurVPos <= CurVPos + 1;
        else
            CurVPos <= 0;
    end
end

//##### VGA Logic (http://tinyvga.com/vga-timing/640x480@60Hz)

//HSync logic
always @(posedge CLK_25MHz)
    if(CurHPos < HSyncWidth)
        HS <= 1;
    else
        HS <= 0;

//VSync logic
always @(posedge CLK_25MHz)
    if(CurVPos < VSyncWidth)
        VS <= 1;
    else
        VS <= 0;

//Horizontal logic
always @(posedge CLK_25MHz)
    if((CurHPos >= HSyncWidth + HFrontPorch) && (CurHPos < HSyncWidth + HFrontPorch + HDisplayArea))
        HBlank <= 0;
    else
        HBlank <= 1;

//Vertical logic
always @(posedge CLK_25MHz)
    if((CurVPos >= VSyncWidth + VFrontPorch) && (CurVPos < VSyncWidth + VFrontPorch + VDisplayArea))
        VBlank <= 0;
    else
        VBlank <= 1;

//Do not output any color information when we are in the vertical
//or horizontal blanking areas. Set a boolean to keep track of this.
always @(posedge CLK_25MHz)
    if(HBlank || VBlank)
        Blank <= 1;
    else
        Blank <= 0;

//Keep track of the current "real" X position. This is the actual current X
//pixel location abstracted away from all the timing details
always @(posedge CLK_25MHz)
    if(HBlank)
        CurrentX <= 0;
    else
        CurrentX <= CurHPos - HSyncWidth - HFrontPorch;

//Keep track of the current "real" Y position. This is the actual current Y
//pixel location abstracted away from all the timing details
always @(posedge CLK_25MHz)
    if(VBlank)
        CurrentY <= 0;
    else
        CurrentY <= CurVPos - VSyncWidth - VFrontPorch;

// VGA Display
reg [7:0] Color = {0};
```

```
always @(posedge CLK_25MHz) begin
    if(Blank) begin
        Color <= 0;
    end
    else begin
        //Color <= COLOR_DATA_IN
        Color[7:5] <= 3'b000;       //R
        Color[4:2] <= 3'b000;       //G
        Color[1:0] <= 2'b00;                //B
        // paddles
        if( CurrentY >= LPminY && CurrentY <= LPmaxY) // left paddle
            if( CurrentX >= LPminX && CurrentX <= LPmaxX) begin
                Color[7:5] <= 3'b111;       //R
                Color[4:2] <= 3'b000;       //G
                Color[1:0] <= 2'b00;                //B
            end
            /*else  begin       // reset RGB to the background
                Color[7:5] <= 3'b000;       //R
                Color[4:2] <= 3'b000;       //G
                Color[1:0] <= 2'b00;                //B
            end*/
        if( CurrentY >= RPminY && CurrentY <= RPmaxY) // right paddle
            if( CurrentX >= RPminX && CurrentX <= RPmaxX) begin     // right paddle
                Color[7:5] <= 3'b111;       //R
                Color[4:2] <= 3'b111;       //G
                Color[1:0] <= 2'b00;                //B
            end

        //          ball
        if( CurrentY >= BallminY && CurrentY <= BallmaxY) begin
                if( CurrentX >= BallminX && CurrentX <= BallmaxX) begin // draw the ball
                Color[7:5] <= 3'b111;       //R
                Color[4:2] <= 3'b111;       //G
                Color[1:0] <= 2'b11;                //B
                end
            end
    end // not in the blank
end

// update paddles' location
always@ ( posedge CLK_300Hz) begin
    case( LP_DIR)       // left paddle
        2'b01: begin        // LP moves up
            LPminY <= (LPminY > 0) ?  LPminY - 1 : LPminY;
            LPmaxY <= (LPmaxY > LPminY + 160) ? LPmaxY - 1: LPmaxY;
        end
        2'b10: begin        // LP moves down
        LPmaxY <= (LPmaxY < 480) ? LPmaxY + 1 : LPmaxY;
        LPminY <= (LPminY < LPmaxY - 160) ? LPminY + 1: LPminY;
        end
    endcase
    case( RP_DIR)       // right paddle
        2'b01: begin        // RP moves up
            RPminY <= (RPminY > 0) ?  RPminY - 1 : RPminY;
            RPmaxY <= (RPmaxY > RPminY + 160) ? RPmaxY - 1: RPmaxY;
        end
        2'b10: begin        // RP moves down
        RPmaxY <= (RPmaxY < 480) ? RPmaxY + 1 : RPmaxY;
        RPminY <= (RPminY < RPmaxY - 160) ? RPminY + 1: RPminY;
        end
    endcase
end
```

```
        //update ball's velocity
        reg [2:0] Vx = 3'b110;          // MSB for direction
        reg [4:0] Vy = 5'b00001;        // MSB for direction
        always@ ( posedge CLK_200Hz) begin
                if( BallmaxX > RPminX  && !((BallmaxY < RPminY && BallminY < RPminY) || (BallminY > RPmaxY &&
BallmaxY > RPmaxY))) // right
                begin
                        Vx[2] = ~Vx[2];
                        if( Vy[4] == 1) // ball moves up
                                if( RP_DIR == 2'b01)            // RP moves up (speed up the ball)
                                        if( Vy[2:0] < 4)
                                                Vy[2:0] = Vy[2:0] + 1;
                                        else
                                                Vy[2:0] = Vy[2:0];
                                else if( RP_DIR == 2'b10)       // RP moves down  (speed down the ball)
                                        if( Vy[2:0] > 1)
                                                Vy[2:0] = Vy[2:0] - 1;
                                        else
                                                Vy[2:0] = Vy[2:0];
                        else    // ball moves down
                                if( RP_DIR == 2'b01)            // RP moves up (speed down the ball)
                                        if( Vy[2:0] > 1)
                                                Vy[2:0] = Vy[2:0] - 1;
                                        else
                                                Vy[2:0] = Vy[2:0];
                                else if( RP_DIR == 2'b10)       // RP moves down  (speed up the ball)
                                        if( Vy[2:0] < 4)
                                                Vy[2:0] = Vy[2:0] + 1;
                                        else
                                                Vy[2:0] = Vy[2:0];
                end
                else if( BallminX < LPmaxX && !((BallmaxY < LPminY && BallminY < LPminY) || (BallminY > LPmaxY &&
BallmaxY > LPmaxY)))    // left
                begin
                        Vx[2] = ~Vx[2];
                        if( Vy[4] == 1) // ball moves up
                                if( LP_DIR == 2'b01)            // LP moves up (speed up the ball)
                                        if( Vy[2:0] < 4)
                                                Vy[2:0] = Vy[2:0] + 1;
                                        else
                                                Vy[2:0] = Vy[2:0];
                                else if( LP_DIR == 2'b10)       // LP moves down  (speed down the ball)
                                        if( Vy[3:0] > 1)
                                                Vy[2:0] = Vy[2:0] - 1;
                                        else
                                                Vy[2:0] = Vy[2:0];
                        else    // ball moves down
                                if( LP_DIR == 2'b01)            // LP moves up (speed down the ball)
                                        if( Vy[3:0] > 1)
                                                Vy[2:0] = Vy[2:0] - 1;
                                        else
                                                Vy[2:0] = Vy[2:0];
                                else if( LP_DIR == 2'b10)       // LP moves down  (speed up the ball)
                                        if( Vy[2:0] < 4)
                                                Vy[2:0] = Vy[2:0] + 1;
                                        else
                                                Vy[2:0] = Vy[2:0];
                end
                /*if( (BallmaxX > RPminX  && !((BallmaxY < RPminY && BallminY < RPminY) || (BallminY > RPmaxY &&
BallmaxY > RPmaxY))) || // right
                        ((BallminX < LPmaxX) && !((BallmaxY < LPminY && BallminY < LPminY) || (BallminY > LPmaxY
&& BallmaxY > LPmaxY)))         // left
                begin
```

```
            Vx[2] = ~Vx[2];
            if( Vy[4] == 1)       // ball moves up
            if( Vy[3:0] < 10)
                Vy[3:0] = Vy[3:0] + 1;
        end*/
        if( BallminY < 0 || BallmaxY > 480)
            Vy[4] = ~Vy[4];
        BallminX = (Vx[2]) ? BallminX - Vx[1:0] : BallminX + Vx[1:0];
        BallmaxX = (Vx[2]) ? BallmaxX - Vx[1:0] : BallmaxX + Vx[1:0];
        BallminY = (Vy[4]) ? BallminY - Vy[2:0] : BallminY + Vy[2:0];
        BallmaxY = (Vy[4]) ? BallmaxY - Vy[2:0] : BallmaxY + Vy[2:0];
    end

    //##### Combinatorial Code

    assign CURX = CurrentX;
    assign CURY = CurrentY;
    assign VBLANK = VBlank;
    assign HBLANK = HBlank;

    //Respect VGA blanking areas. Do not drive color outputs when blanked (bad things may happen).
    assign RED = (Blank) ? 3'b000 : Color[7:5];
    assign GREEN = (Blank) ? 3'b000 : Color[4:2];
    assign BLUE = (Blank) ? 2'b00 : Color[1:0];
endmodule

/*******************************************************************************************/

// KEYBOARD READER
// We have wrote these codes to read which key is pressed on the PS2 keyboard. We have searched on this topic and eventually
// we have achieved to recognize which key is pressed on the keyboard. However, since our project requires 2 player, we decided
// not to use keyboard because we could not achieved to get two simultanesouly pressed key from PS2 keyboard. It is not possible
// accroding to our last researches on the Internet. Thus, we have decided to use buttons on the BASYS2 FPGA to control the paddles.
module KeyboardReader(clk,data,scan_code,parity_error,rdy, lp_dir, rp_dir, B1, B2, B3, B4, CLK50);
// Port declarations
 input clk;          // PS_2 clock input
 input data;          // PS_2 data input
 output[7:0] scan_code;  // Scan_code output
 output parity_error;    // Parity output
 output rdy;          // Data ready output

 input B1, B2, B3, B4;
 input CLK50;
 output [1:0] lp_dir, rp_dir;    //        01-up     10-down  11-dm

// Internal Variables
 reg[9:0] register;
 reg[3:0] counter;
 reg parity_error, rdy;

 assign scan_code = register[9:2];
 assign parity = register[1];

// PS/2 logic
 always @ (negedge clk)
  begin
   register <= {register[8:0], data}; // receive data
   if (counter == 4'b1011)
     counter <= 4'b0000;
   else
     counter <= counter + 4'b1;
  end

// PS/2 parity logic
 always @ (posedge clk)
  begin
   if (counter == 4'b1011)
     if (!parity == ^scan_code) // parity check (odd parity)
      rdy <= 1'b1;
     else
      parity_error <= 1'b1;
   else  // not all 10 bits receiverd yet
    begin
     rdy <= 1'b0;
     parity_error <= 1'b0;
    end
  end

  // button direction
  reg [1:0] lp_dir, rp_dir;
  always@( posedge CLK50)
  begin
                  // left paddle
                  if( B1 == 1)
                                lp_dir <= 2'b01;
                  else if( B2 == 1)
                                lp_dir <= 2'b10;
                  else
                                lp_dir <= 2'b11;

                  //right paddle
                  if( B3 == 1)
                   rp_dir <= 2'b01;
                  else if( B4 == 1)
                                rp_dir <= 2'b10;
                  else
                                rp_dir <= 2'b11;
  end

  // keyboard direction
  //assign lp_dir = 2'b10;
 /*
  reg [1:0] lp_dir, rp_dir;
  reg isF0;

  always@( rdy)
  if( scan_code == 8'b10111000) // W(1D) 00011101
  begin
                if( isF0 == 1'b0) begin
                                lp_dir <= 2'b01;       // up
                                isF0 = 0;
                                end
                else          // F0 was send
                                lp_dir <= 2'b11;       // W is released -don't move-
  end
  else if( scan_code == 8'b11011000)    // S(1B) 00011011
  begin
                if( isF0 == 1'b0) begin
                                lp_dir <= 2'b10;       // down
                                isF0 = 0;
                                end
                else          // F0 was send
                lp_dir <= 2'b11;      // S is released -don't move-
 end
 else if( scan_code == 8'b00001111)    // F0(11110000)
        isF0 = 1;
 */

endmodule

    /******************************************************************************************/
    // CLOCK DIVIDER
    // These codes are for different clock frequencies.
    // clk_25mhz is used for VGA display.
    // clk_200hz is used for displaying ball.
    // clk_300hz is used for displaying paddles.

    module clock_divider(clk_in, clk_25mhz, clk_200hz, clk_300hz);
        input clk_in;
        output clk_25mhz;
        output clk_200hz;
        output clk_300hz;

        reg clk_25mhz = 0;
        reg clk_200hz = 0;
        reg clk_300hz = 0;
        reg [17:0] counter200 = {0};
        reg [17:0] counter300 = {0};

        always @(posedge clk_in) begin
                clk_25mhz <= ~clk_25mhz;
                if( counter200 < 18'b111101000010010000) begin // counter <= 250000
                        counter200 <= counter200 + 1;
                        clk_200hz <= 0;
                end
                else begin
                        counter200 <= 0;
                        clk_200hz <= 1;
                end
                if( counter300 < 18'b101000101100001011) begin // counter <= 166667
                        counter300 <= counter300 + 1;
                        clk_300hz <= 0;
                end
                else begin
                        counter300 <= 0;
                        clk_300hz <= 1;
                end
        end

    endmodule

    /******************************************************************************************/

    // CLOCK SHIFT
    // These codes are taken from http://io.smashthestack.org/arm/digitalcold/ as a part of their VGA controller.
    /* clock_shift.v - shifts the clk_in by half of the period

        DigitalCold's ARM Challenge VGA Implementation
                Freefull's graphics demo reimplemented on the BASYS2 hardware.
                Additional information at http://io.smashthestack.org/arm/digitalcold/
                All Verilog code written by DigitalCold.

        Released 4/25/13
    */
    module clock_shift(clk_in, clk_out);
        input clk_in;
        output clk_out;

        assign clk_out = ~clk_in;
    endmodule

    /****************************************************************************************************/
```

```
# UCF file of our project.

# BASYS 2 UCF

NET "CLK_50MHz" LOC = "B8";
NET "CLK_PS2" LOC = "B1";
NET "DATA_PS2" LOC = "C3";

NET "CLK_PS2" CLOCK_DEDICATED_ROUTE = FALSE;

# Pin assignment for VGA
NET "HS"   LOC = "J14"  ;
NET "VS"   LOC = "K13"  ;
NET "RED<2>"  LOC = "F13"  ;
NET "RED<1>"  LOC = "D13"  ;
NET "RED<0>"  LOC = "C14"  ;
NET "GREEN<2>"  LOC = "G14"  ;
NET "GREEN<1>"  LOC = "G13"  ;
NET "GREEN<0>"  LOC = "F14"  ;
NET "BLUE<1>"  LOC = "J13"  ;
NET "BLUE<0>"  LOC = "H13"  ;
NET "SWITCH<2>"  LOC = "K3"  ;
NET "SWITCH<1>"  LOC = "L3"  ;
NET "SWITCH<0>"  LOC = "P11"  ;

NET "B1"  LOC = "A7"  ;
NET "B2"  LOC = "M4"  ;
NET "B3"  LOC = "C11"  ;
NET "B4"  LOC = "G12"  ;

NET "led<0>"  LOC = "M5"  ;
NET "led<1>"  LOC = "M11"  ;
NET "scanLED<0>"  LOC = "B7";
NET "scanLED<1>"  LOC = "C5";
NET "scanLED<2>"  LOC = "B6";
NET "scanLED<3>"  LOC = "C6";
NET "scanLED<4>"  LOC = "B5";
NET "scanLED<5>"  LOC = "J3";
NET "scanLED<6>"  LOC = "A3";
NET "scanLED<7>"  LOC = "B2";

#Created by Constraints Editor (xc3s100e-cp132-4) - 2013/04/26
NET "CLK_50MHz" TNM_NET = CLK_50MHz;
TIMESPEC TS_CLK_50MHz = PERIOD "CLK_50MHz" 50 MHz HIGH 50%;
NET "vga/clk_div/clk_25mhz1" TNM_NET = vga/clk_div/clk_25mhz1;
TIMESPEC TS_CLK_25MHz = PERIOD "vga/clk_div/clk_25mhz1" 25 MHz HIGH 50%;
```