ECE 540 Winter 2018 Final Project
Rojobot Game: Bridge Runner

Design Report

Rakhee Bhojakar, Noah Brummer, Sanika Chaoji, Poonam Ganoskar

Table of Contents

Project Description:

Project involved designing of a game using Nexys4 FPGA DDR board. We built a custom world map with lines for two Rojobots to follow with several gaps/bridges in the line. Users have a controllable “Bridge” they use to close these gaps to allow their respective Rojobot to get to the end. The game allows each user to close gaps as they choose so that their Rojobot can get to a goal at the end of the world map. Whomever gets their Rojobot to the goal quicker will win. The project was divided in two parts: hardware development and software development.

Hardware development:

The following is the block diagram that shows the design hierarchy of the Rojobot Bridge Runner:

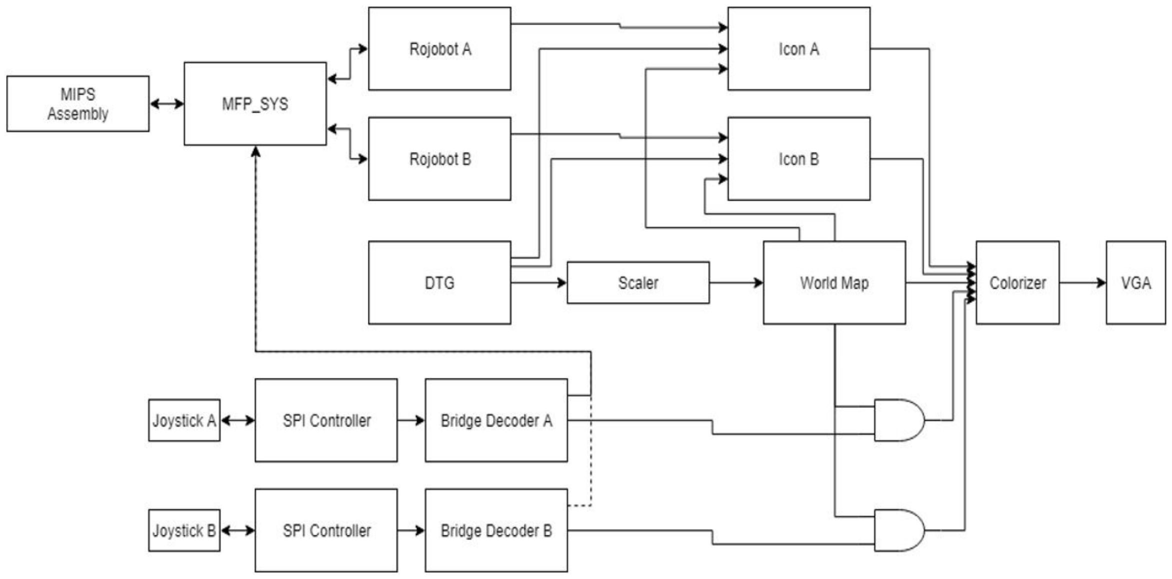


Figure 1: Block Diagram of Bridge Runner

The robot platform and its world are emulated in an IP (intellectual-property) block called Rojobot31. The black line follower robot is nothing but the icon following the world map is created by us in the form of a verilog module on the basis of all the instructions and specifications provided to us. The Rojobot provides information about its position in the virtual world through a set of registers. These registers indicate the location, orientation (heading), its motion (stopped, moving forward or reverse, turning left or right) and the values of its proximity and line tracking sensors. We created a memory mapped I/O interface to the Rojobot registers by implementing an additional AHB-Lite slave peripheral for the Rojobot registers. The world map memory was generated from a graphic bitmap and the colorizer module which we wrote made use of a 4-bit signal coming out of this memory along with the display timing generator (DTG) module (provided to us), its outputs deals with controlling the electron beam directed towards the VGA monitor by generating horizontal and vertical synchronization signals. These signals are connected to the horizontal and vertical deflection plates, which come in the path of the electron beam directed towards the screen thereby controlling its focus on the screen giving us the required map pattern on VGA monitor.

World_map: World_map is a dual port memory storing the “world” that the Rojobot is moving through. It provides the black line following path for the two robot with bridges in the VGA monitor. The VGA connector component on the FPGA board converts all digital signals to analog signals by representing the icon and world map on the VGA monitor. The following is the worldmap created for our project.

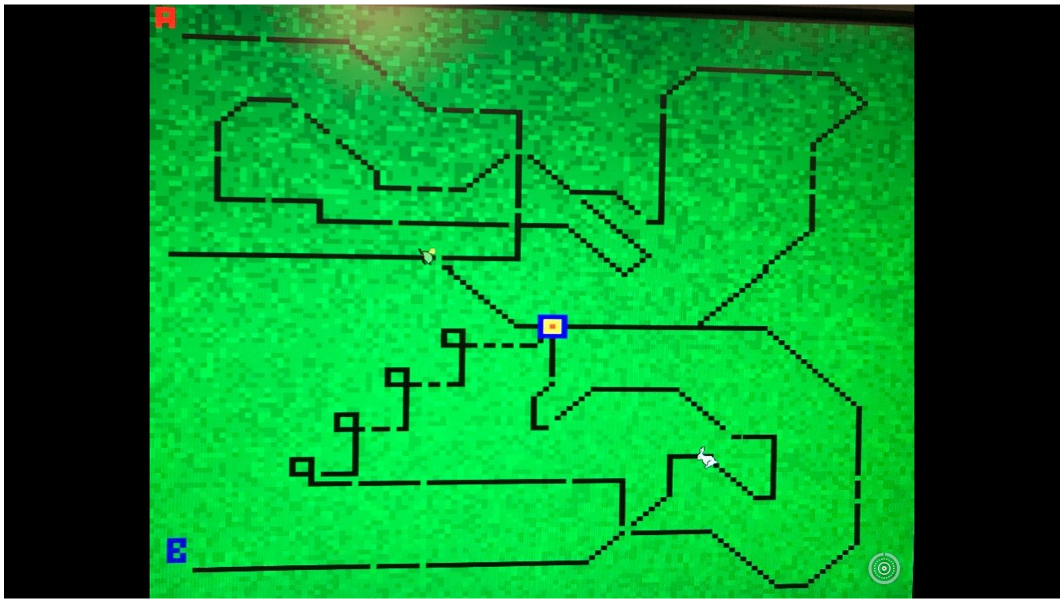


Figure 2: Worldmap

Handshake flip-flop: It is a Set/Reset flip-flop that can be queried by the program to determine whether the Rojobot emulation has updated the state of the Rojobot. This flip-flop needs to be accessible on the AHB-Lite bus as a register bit that can be read and cleared under program control. The hand-shaking flip flop is used to synchronize with the MIPS system as we want to create an abstraction that every signal gets updated only once.

Display Time Generator: The Display Timing Generator (dtg.v) is provided to us in the specification for project 2. It produces the pixel row and pixel column addresses. These addresses can be combined and driven to the port B address input to the ROM. The resulting pixel stream is sent to the Colorizer to draw an image of the world map

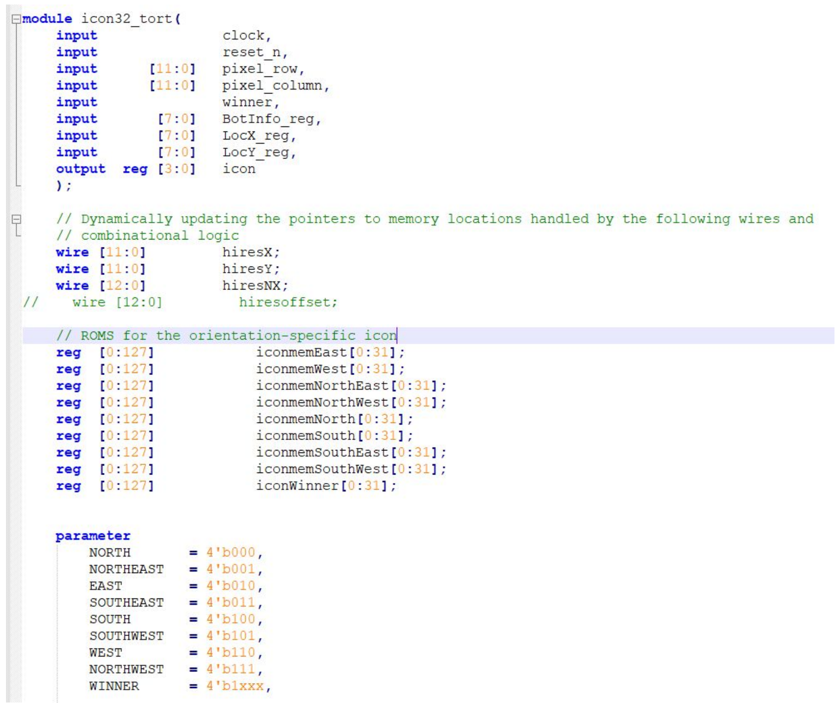
Icon Module:

- Icons were built in an online application called piskelapp. This allows us to easily draw icons with however many colors we choose.
- Piskelapp files are exported with each pixel encoded with a 32-bit word. These are run through a Python script that maps the 32-bit word to a 4-bit word according to a predefined color palette. The python script produces a textfile with the 4-bit encoded pixels. These files are converted to ROMs in our icon modules with the verilog $memfileb function.
- Pixel_row and pixel_column are the addresses provided by the display time generator module which indicates the current position of the pixel.
- The X and Y position input values are compared with the DTG row/column values. Thus, with the proper scaling between the Rojobot’s location (Loc_X and Loc_Y), the icon can be positioned anywhere on the screen with a resolution of one pixel, vertically or horizontally. LocX and LocY

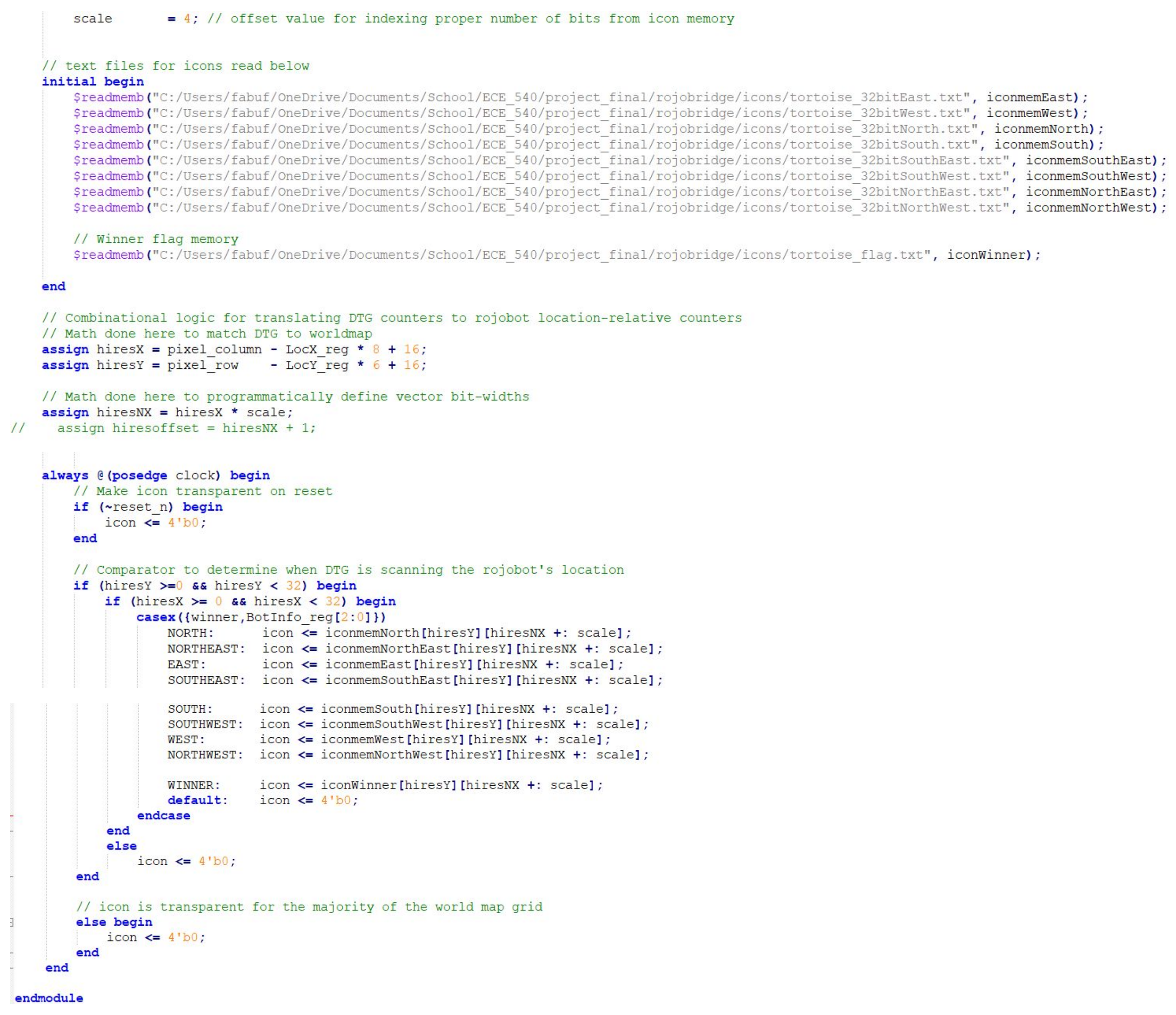
indicate the current location of bot in X-Y co-ordinate system. Orientation of the bot is obtained from BotInfo register of bot.v module.

- As the Rojobot icon moves through the world map, the icon is drawn at the Rojobot's current location. The icon module also shows the Rojobot's orientation. The BotInfo_reg output provides eight headings: 0, 45, 90, 135, 180, 225, 270, and 315 degree encoded into 3 bits [0 (0) – 315 (7)].

Following is the code snippet of icon_tort module.



```
module icon32_tort(
    input                   clock,
    input                   reset_n,
    input         [11:0]    pixel_row,
    input         [11:0]    pixel_column,
    input                   winner,
    input          [7:0]    BotInfo_reg,
    input          [7:0]    LocX_reg,
    input          [7:0]    LocY_reg,
    output  reg    [3:0]    icon
    );

    // Dynamically updating the pointers to memory locations handled by the following wires and
    // combinational logic
    wire [11:0]             hiresX;
    wire [11:0]             hiresY;
    wire [12:0]             hiresNX;
//    wire [12:0]             hiresoffset;

    // ROMS for the orientation-specific icon
    reg  [0:127]                iconmemEast[0:31];
    reg  [0:127]                iconmemWest[0:31];
    reg  [0:127]                iconmemNorthEast[0:31];
    reg  [0:127]                iconmemNorthWest[0:31];
    reg  [0:127]                iconmemNorth[0:31];
    reg  [0:127]                iconmemSouth[0:31];
    reg  [0:127]                iconmemSouthEast[0:31];
    reg  [0:127]                iconmemSouthWest[0:31];
    reg  [0:127]                iconWinner[0:31];


    parameter
        NORTH        = 4'b000,
        NORTHEAST    = 4'b001,
        EAST         = 4'b010,
        SOUTHEAST    = 4'b011,
        SOUTH        = 4'b100,
        SOUTHWEST    = 4'b101,
        WEST         = 4'b110,
        NORTHWEST    = 4'b111,
        WINNER       = 4'b1xxx,
```



```
    scale       = 4; // offset value for indexing proper number of bits from icon memory


// text files for icons read below
initial begin
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_32bitEast.txt", iconmemEast);
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_32bitWest.txt", iconmemWest);
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_32bitNorth.txt", iconmemNorth);
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_32bitSouth.txt", iconmemSouth);
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_32bitSouthEast.txt", iconmemSouthEast);
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_32bitSouthWest.txt", iconmemSouthWest);
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_32bitNorthEast.txt", iconmemNorthEast);
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_32bitNorthWest.txt", iconmemNorthWest);

    // Winner flag memory
    $readmemb("C:/Users/fabuf/OneDrive/Documents/School/ECE_540/project_final/rojobridge/icons/tortoise_flag.txt", iconWinner);

end

// Combinational logic for translating DTG counters to rojobot location-relative counters
// Math done here to match DTG to worldmap
assign hiresX = pixel_column - LocX_reg * 8 + 16;
assign hiresY = pixel_row    - LocY_reg * 6 + 16;

// Math done here to programmatically define vector bit-widths
assign hiresNX = hiresX * scale;
//    assign hiresoffset = hiresNX + 1;


always @(posedge clock) begin
    // Make icon transparent on reset
    if (~reset_n) begin
        icon <= 4'b0;
    end

    // Comparator to determine when DTG is scanning the rojobot's location
    if (hiresY >=0 && hiresY < 32) begin
        if (hiresX >= 0 && hiresX < 32) begin
            casex({winner,BotInfo_reg[2:0]})
                NORTH:      icon <= iconmemNorth[hiresY][hiresNX +: scale];
                NORTHEAST:  icon <= iconmemNorthEast[hiresY][hiresNX +: scale];
                EAST:       icon <= iconmemEast[hiresY][hiresNX +: scale];
                SOUTHEAST:  icon <= iconmemSouthEast[hiresY][hiresNX +: scale];

                SOUTH:      icon <= iconmemSouth[hiresY][hiresNX +: scale];
                SOUTHWEST:  icon <= iconmemSouthWest[hiresY][hiresNX +: scale];
                WEST:       icon <= iconmemWest[hiresY][hiresNX +: scale];
                NORTHWEST:  icon <= iconmemNorthWest[hiresY][hiresNX +: scale];

                WINNER:     icon <= iconWinner[hiresY][hiresNX +: scale];
                default:    icon <= 4'b0;
            endcase
        end
        else
            icon <= 4'b0;
    end

    // icon is transparent for the majority of the world map grid
    else begin
        icon <= 4'b0;
    end
end

endmodule
```

Following are the created icons for our project.

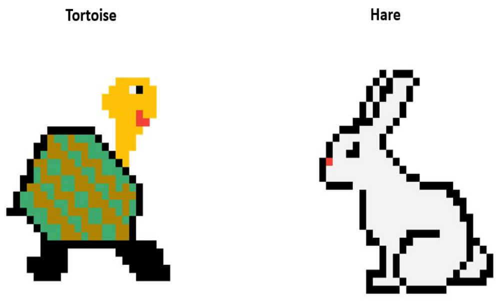


Figure 3: Two Rojobot Icons

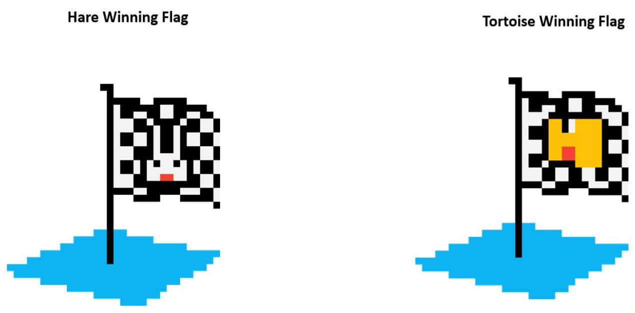


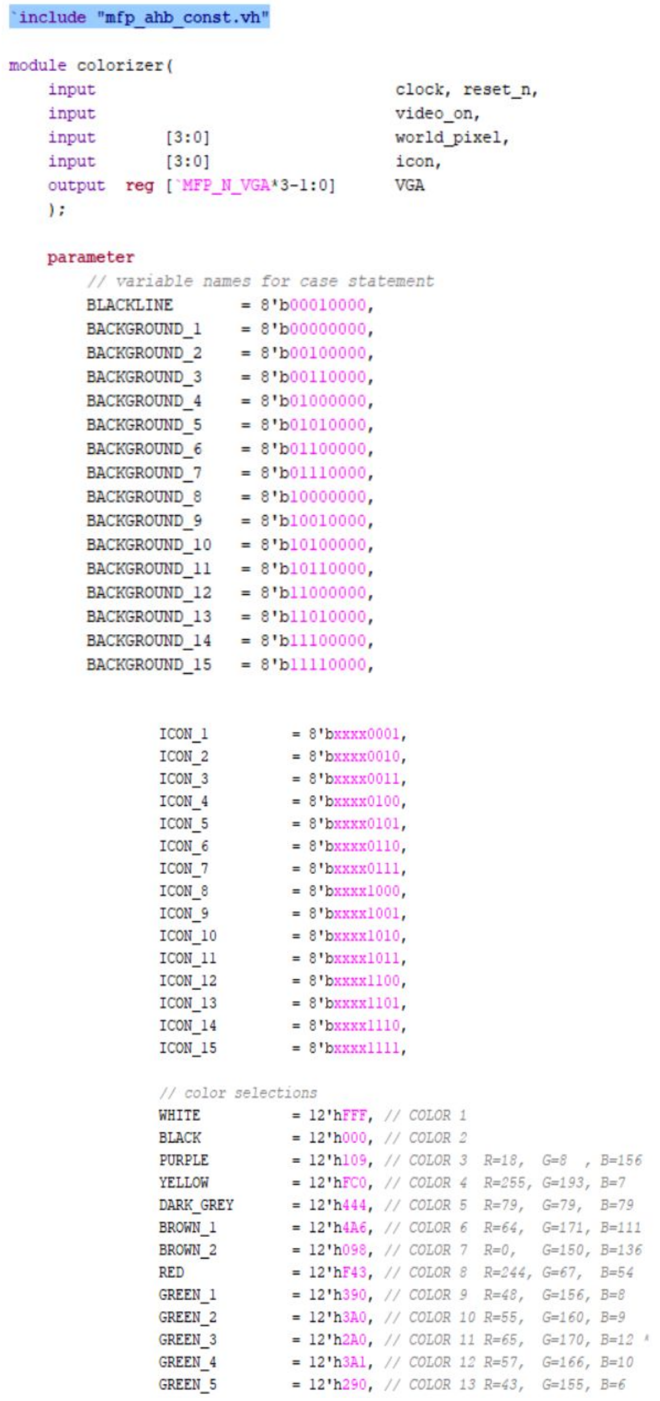
Figure 4: Winning Flag Icons

Colorizer Module:

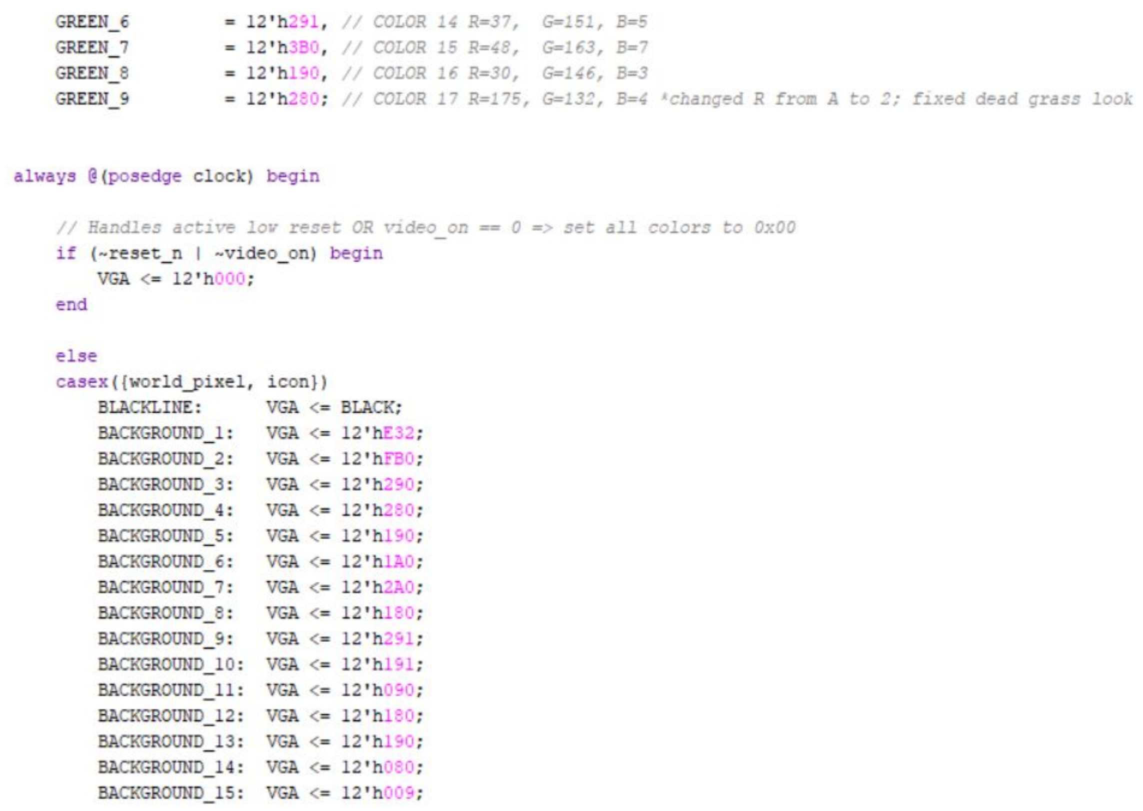
The colorizer module works at 75MHz clock. It assumes 4-bit color for the worldmap as well as 4-bit color for the player icons. This allows a total of 32 distinct colors, 16 of which are used for the worldmap palette, and 16 which are shared by the player icons.

- video_on signal from the display time generator module acts an enable signal for colorizer module. World_pixel and icon_pixel values are evaluated when video_on signal is HIGH.
- Based on the world_pixel and icon_pixel values, the colorizer module provides 12 bit RGB color to be displayed on the monitor screen via VGA port.
- world_pixel provides the bot world (background) on the VGA display monitor which is provided by bot.v module.
- icon_pixel represents the icon (foreground) which is provided by icon module.
- The Red, Green and Blue color output from the colorizer module is used to represent the world map (the black line traverse path and obstruction) on the VGA display monitor.

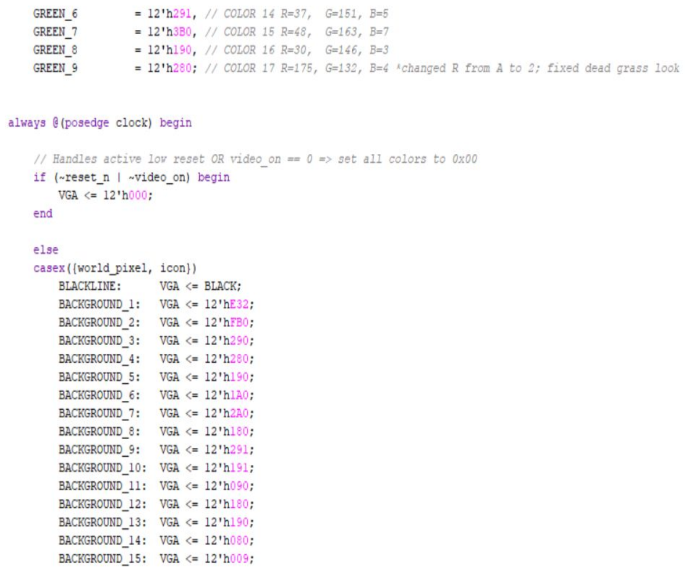
Following is the code snippet of colorizer module.



```
`include "mfp_ahb_const.vh"

module colorizer(
    input                               clock, reset_n,
    input                               video_on,
    input       [3:0]                   world_pixel,
    input       [3:0]                   icon,
    output  reg [`MFP_N_VGA*3-1:0]      VGA
    );

    parameter
        // variable names for case statement
        BLACKLINE       = 8'b00010000,
        BACKGROUND_1    = 8'b00000000,
        BACKGROUND_2    = 8'b00100000,
        BACKGROUND_3    = 8'b00110000,
        BACKGROUND_4    = 8'b01000000,
        BACKGROUND_5    = 8'b01010000,
        BACKGROUND_6    = 8'b01100000,
        BACKGROUND_7    = 8'b01110000,
        BACKGROUND_8    = 8'b10000000,
        BACKGROUND_9    = 8'b10010000,
        BACKGROUND_10   = 8'b10100000,
        BACKGROUND_11   = 8'b10110000,
        BACKGROUND_12   = 8'b11000000,
        BACKGROUND_13   = 8'b11010000,
        BACKGROUND_14   = 8'b11100000,
        BACKGROUND_15   = 8'b11110000,


            ICON_1          = 8'bxxxx0001,
            ICON_2          = 8'bxxxx0010,
            ICON_3          = 8'bxxxx0011,
            ICON_4          = 8'bxxxx0100,
            ICON_5          = 8'bxxxx0101,
            ICON_6          = 8'bxxxx0110,
            ICON_7          = 8'bxxxx0111,
            ICON_8          = 8'bxxxx1000,
            ICON_9          = 8'bxxxx1001,
            ICON_10         = 8'bxxxx1010,
            ICON_11         = 8'bxxxx1011,
            ICON_12         = 8'bxxxx1100,
            ICON_13         = 8'bxxxx1101,
            ICON_14         = 8'bxxxx1110,
            ICON_15         = 8'bxxxx1111,

            // color selections
            WHITE           = 12'hFFF, // COLOR 1
            BLACK           = 12'h000, // COLOR 2
            PURPLE          = 12'h109, // COLOR 3  R=18,  G=8  , B=156
            YELLOW          = 12'hFC0, // COLOR 4  R=255, G=193, B=7
            DARK_GREY       = 12'h444, // COLOR 5  R=79,  G=79,  B=79
            BROWN_1         = 12'h4A6, // COLOR 6  R=64,  G=171, B=111
            BROWN_2         = 12'h098, // COLOR 7  R=0,   G=150, B=136
            RED             = 12'hF43, // COLOR 8  R=244, G=67,  B=54
            GREEN_1         = 12'h390, // COLOR 9  R=48,  G=156, B=8
            GREEN_2         = 12'h3A0, // COLOR 10 R=55,  G=160, B=9
            GREEN_3         = 12'h2A0, // COLOR 11 R=65,  G=170, B=12 *
            GREEN_4         = 12'h3A1, // COLOR 12 R=57,  G=166, B=10
            GREEN_5         = 12'h290, // COLOR 13 R=43,  G=155, B=6
```



```
    GREEN_6         = 12'h291, // COLOR 14 R=37,  G=151, B=5
    GREEN_7         = 12'h3B0, // COLOR 15 R=48,  G=163, B=7
    GREEN_8         = 12'h190, // COLOR 16 R=30,  G=146, B=3
    GREEN_9         = 12'h280; // COLOR 17 R=175, G=132, B=4 *changed R from A to 2; fixed dead grass look


always @(posedge clock) begin

    // Handles active low reset OR video_on == 0 => set all colors to 0x00
    if (~reset_n | ~video_on) begin
        VGA <= 12'h000;
    end

    else
    casex({world_pixel, icon})
        BLACKLINE:      VGA <= BLACK;
        BACKGROUND_1:   VGA <= 12'hE32;
        BACKGROUND_2:   VGA <= 12'hFB0;
        BACKGROUND_3:   VGA <= 12'h290;
        BACKGROUND_4:   VGA <= 12'h280;
        BACKGROUND_5:   VGA <= 12'h190;
        BACKGROUND_6:   VGA <= 12'h1A0;
        BACKGROUND_7:   VGA <= 12'h2A0;
        BACKGROUND_8:   VGA <= 12'h180;
        BACKGROUND_9:   VGA <= 12'h291;
        BACKGROUND_10:  VGA <= 12'h191;
        BACKGROUND_11:  VGA <= 12'h090;
        BACKGROUND_12:  VGA <= 12'h180;
        BACKGROUND_13:  VGA <= 12'h190;
        BACKGROUND_14:  VGA <= 12'h080;
        BACKGROUND_15:  VGA <= 12'h009;
```

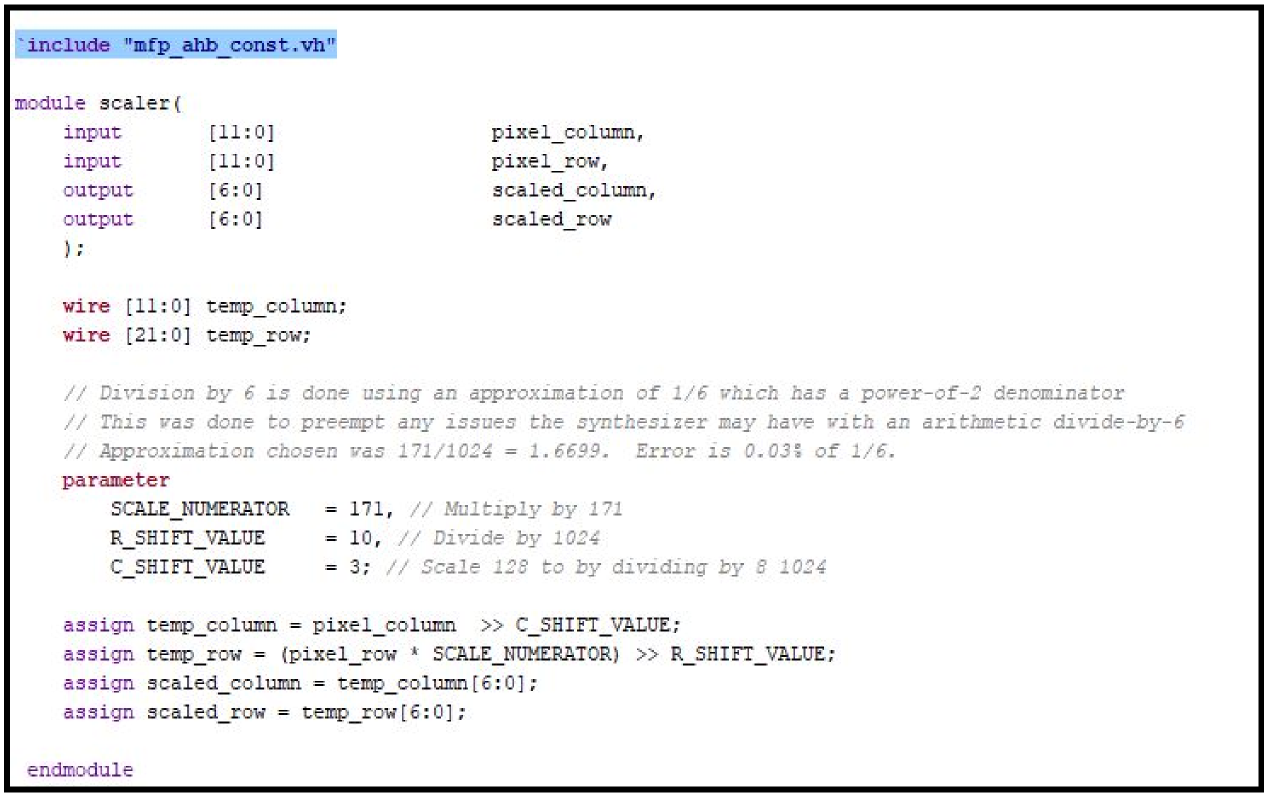


```
    GREEN_6         = 12'h291, // COLOR 14 R=37,  G=151, B=5
    GREEN_7         = 12'h3B0, // COLOR 15 R=48,  G=163, B=7
    GREEN_8         = 12'h190, // COLOR 16 R=30,  G=146, B=3
    GREEN_9         = 12'h280; // COLOR 17 R=175, G=132, B=4 *changed R from A to 2; fixed dead grass look


always @(posedge clock) begin

    // Handles active low reset OR video_on == 0 => set all colors to 0x00
    if (~reset_n | ~video_on) begin
        VGA <= 12'h000;
    end

    else
    casex({world_pixel, icon})
        BLACKLINE:      VGA <= BLACK;
        BACKGROUND_1:   VGA <= 12'hE32;
        BACKGROUND_2:   VGA <= 12'hFB0;
        BACKGROUND_3:   VGA <= 12'h290;
        BACKGROUND_4:   VGA <= 12'h280;
        BACKGROUND_5:   VGA <= 12'h190;
        BACKGROUND_6:   VGA <= 12'h1A0;
        BACKGROUND_7:   VGA <= 12'h2A0;
        BACKGROUND_8:   VGA <= 12'h180;
        BACKGROUND_9:   VGA <= 12'h291;
        BACKGROUND_10:  VGA <= 12'h191;
        BACKGROUND_11:  VGA <= 12'h090;
        BACKGROUND_12:  VGA <= 12'h180;
        BACKGROUND_13:  VGA <= 12'h190;
        BACKGROUND_14:  VGA <= 12'h080;
        BACKGROUND_15:  VGA <= 12'h009;
```

Scaler Module:

The 128x128 world is displayed as a 1024 x 768 image on the VGA monitor so implementation is scaled to represent each world location as an 8 x 6-pixel block on the display.

Following is the code snippet of scaler module.

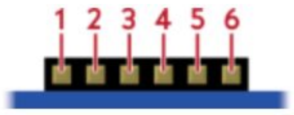


```
`include "mfp_ahb_const.vh"

module scaler(
    input       [11:0]                  pixel_column,
    input       [11:0]                  pixel_row,
    output      [6:0]                   scaled_column,
    output      [6:0]                   scaled_row
    );

    wire [11:0] temp_column;
    wire [21:0] temp_row;

    // Division by 6 is done using an approximation of 1/6 which has a power-of-2 denominator
    // This was done to preempt any issues the synthesizer may have with an arithmetic divide-by-6
    // Approximation chosen was 171/1024 = 1.6699.  Error is 0.03% of 1/6.
    parameter
        SCALE_NUMERATOR   = 171, // Multiply by 171
        R_SHIFT_VALUE     = 10, // Divide by 1024
        C_SHIFT_VALUE     = 3; // Scale 128 to by dividing by 8 1024

    assign temp_column = pixel_column  >> C_SHIFT_VALUE;
    assign temp_row = (pixel_row * SCALE_NUMERATOR) >> R_SHIFT_VALUE;
    assign scaled_column = temp_column[6:0];
    assign scaled_row = temp_row[6:0];

 endmodule
```

PMOD JSTK by DIgilent:

To operate movement of bridge we added two joysticks to the system. The PMOD JSTK are two-axis resistive joystick with 6-pin Pmod connector with SPI interface.

Pin 1: CS
Pin 2: MOSI
Pin 3: MISO
Pin 4: SCLK
Pin 5: GND
Pin 6: VCC



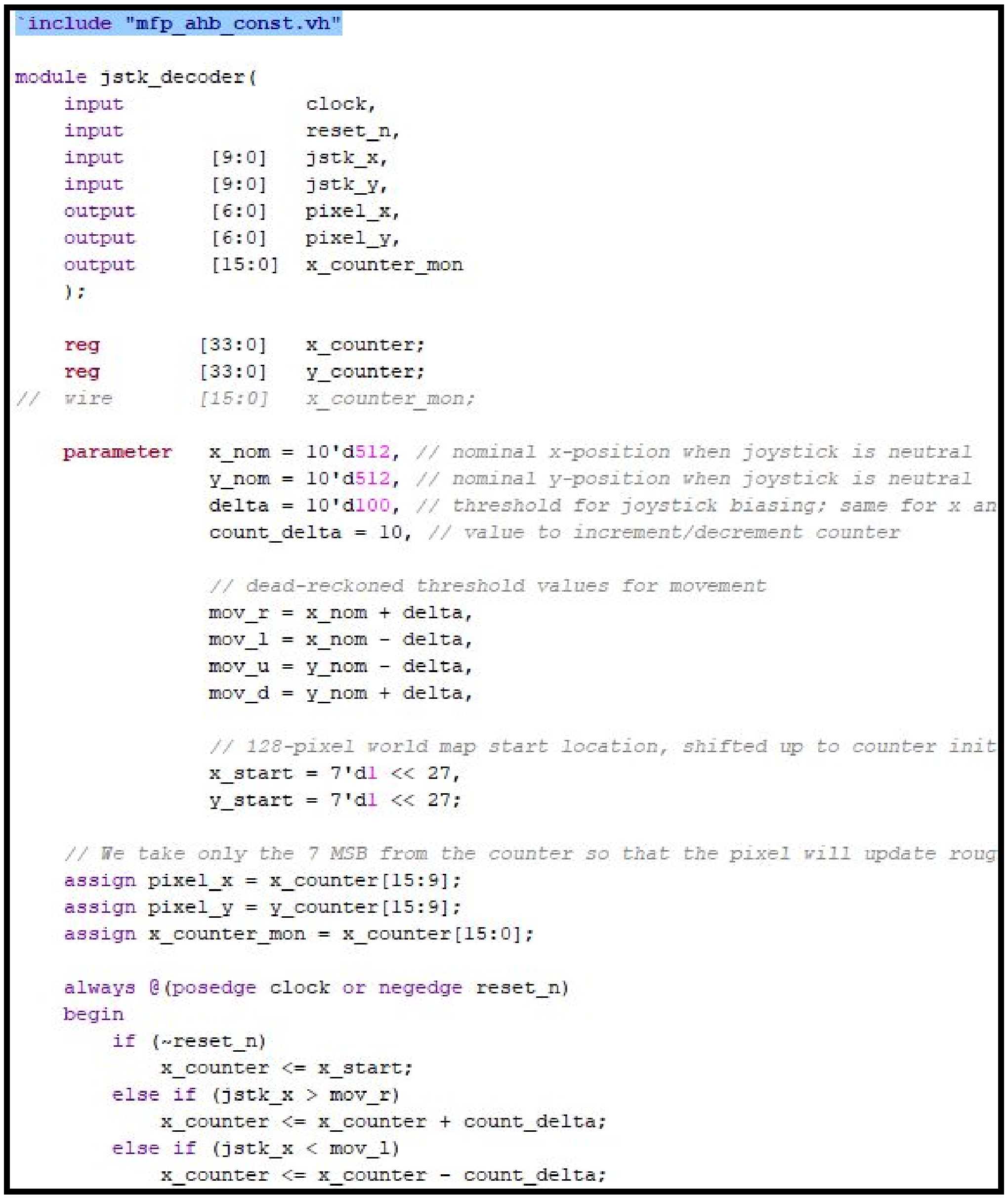
These joysticks were connected to the Pmod connector JA and JC of Nexys4 FPGA board.

JoyStick PMOD Interface:

The PMOD JoyStick consists of three components: 66.67 KHz serial clock, SPI Interface and SPI controller. The SPI Interface components is responsible for sending and receiving a byte of data to and from the PMOD JoyStick when request is made. SPI Controller component manages all data transfer requests, and manages the data byte being sent to the PMOD JoyStick. The 100 MHz input clock signal is converted to 66.67 KHz with 66.67 KHz serial clock module.

Joystick data decoder:

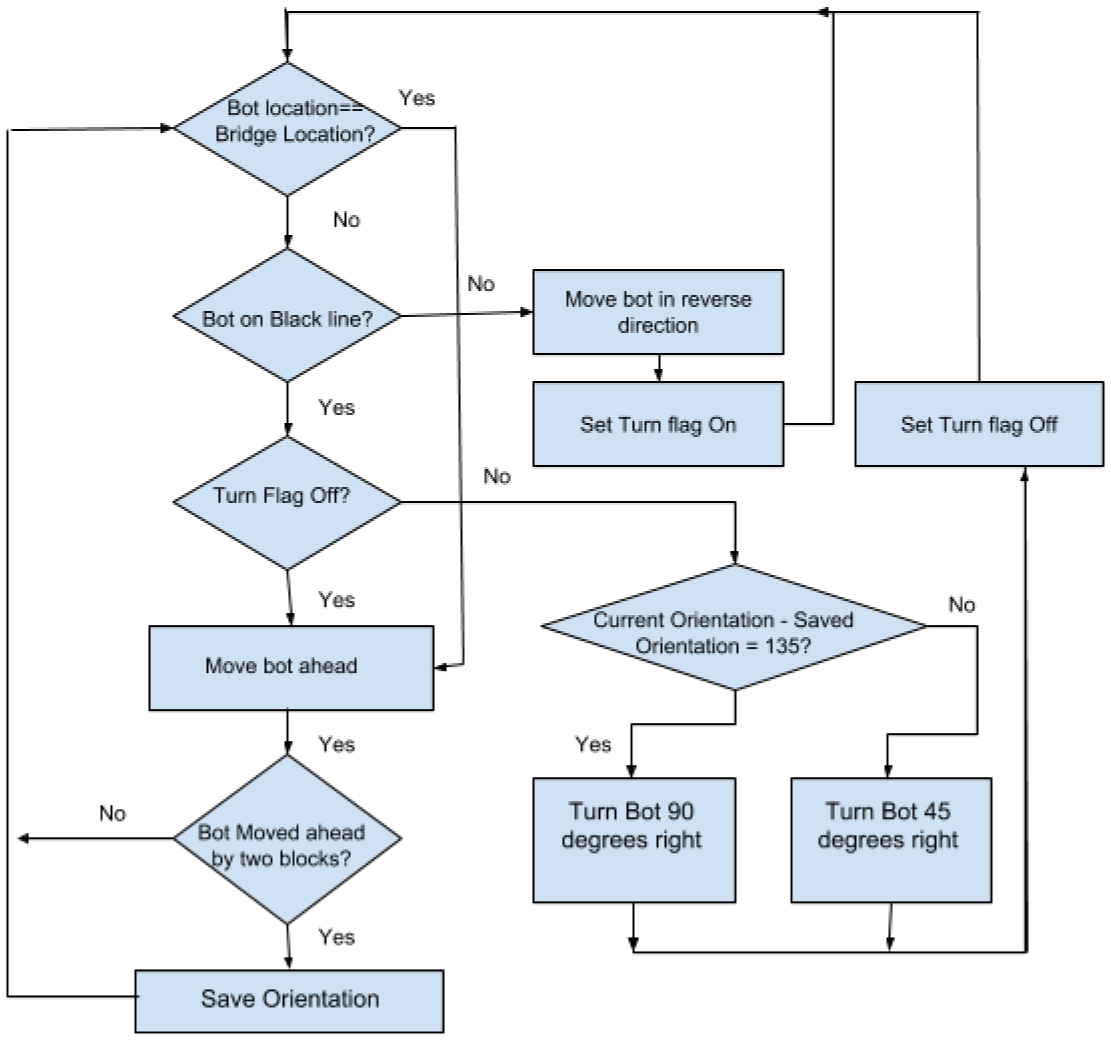
The joystick decoder takes the 10-bit value of each joystick dimension (X and Y) and uses them to change a 34-bit counter designated for the respective dimension. 7 bits from the X and Y counter are used to determine the player's bridge pixel location on the 128 x 128 pixel worldmap.The 7 bits are selected appropriately from the 34-bit counter so that the pixel moves quickly, but also consistently. This was initially calculated using the period of the 50MHz system clock and the desired update rate of the pixel, but was eventually chosen heuristically as different pixel behaviours were observed.Following is the code snippet of joystick data decoder module.



```
`include "mfp_ahb_const.vh"

module jstk_decoder(
    input                   clock,
    input                   reset_n,
    input           [9:0]   jstk_x,
    input           [9:0]   jstk_y,
    output          [6:0]   pixel_x,
    output          [6:0]   pixel_y,
    output          [15:0]  x_counter_mon
    );

    reg            [33:0]   x_counter;
    reg            [33:0]   y_counter;
//  wire           [15:0]   x_counter_mon;

    parameter    x_nom = 10'd512, // nominal x-position when joystick is neutral
                 y_nom = 10'd512, // nominal y-position when joystick is neutral
                 delta = 10'd100, // threshold for joystick biasing; same for x an
                 count_delta = 10, // value to increment/decrement counter

                 // dead-reckoned threshold values for movement
                 mov_r = x_nom + delta,
                 mov_l = x_nom - delta,
                 mov_u = y_nom - delta,
                 mov_d = y_nom + delta,

                 // 128-pixel world map start location, shifted up to counter init
                 x_start = 7'd1 << 27,
                 y_start = 7'd1 << 27;

    // We take only the 7 MSB from the counter so that the pixel will update roug
    assign pixel_x = x_counter[15:9];
    assign pixel_y = y_counter[15:9];
    assign x_counter_mon = x_counter[15:0];

    always @(posedge clock or negedge reset_n)
    begin
        if (~reset_n)
            x_counter <= x_start;
        else if (jstk_x > mov_r)
            x_counter <= x_counter + count_delta;
        else if (jstk_x < mov_l)
            x_counter <= x_counter - count_delta;
```

Software development

The Project requires the hare and the tortoise icons to traverse on the black line on the world map. This involves turning bot as the black line turns left or right. Also the icons stop at a line break until player places bridge pixel to fill in the gap in the track. The algorithm used to implement this is as below:


Bot location== Bridge Location?
Yes
No
Bot on Black line?
No
Move bot in reverse direction
Set Turn flag On
Set Turn flag Off
Yes
Turn Flag Off?
No
Yes
Move bot ahead
Current Orientation - Saved Orientation = 135?
No
Yes
Yes
Turn Bot 90 degrees right
Turn Bot 45 degrees right
No
Bot Moved ahead by two blocks?
Yes
Save Orientation

This algorithm is called inside main loop for both the Rojobots.

Implementation Challenges:

1. Getting the Rojobot initial position to the start of the line in worldmap.
2. Integration of two Memory Mapped I/O in hardware for two Rojobots.
3. Pixel control with Joysticks to fit in the gaps in the line properly.

Stretch Goals Achieved for Demo:

1. Increased the icon resolution from 16 x 16 pixels to 32 x 32 pixels.
2. Increased the worldmap color resolution from 2-bits to 4-bits.

Future Work:

1. Creating an animated icons.
2. Adding Obstacles in the path and making the Rojobots jump over or use an alternate path.
3. Adding ‘Start’ and ‘Game Over’ screens.
4. Changing the default speed of the Hare and Tortoise so that Hare is faster than Tortoise.

Conclusion/ Results:

The project was successfully completed and demonstrated. The two Rojobots icon [Hare and Tortoise] were able to traverse on their respective black line paths with both left and right turns. User is able to route bridge icon to gaps in line. Rojobots walk over line breaks filled with bridge icons. A winning flag is with the winner’s face is displayed when player reaches destination.

Contribution:

Noah Brummer	Sanika Chaoji	Poonam Ganoskar	Rakhee Bhojakar
Verilog System Integration Graphic Design	Software Design using MIPS Assembly language and debug; Graphic Design	PMOD joystick components for Rojobots, Demo Presentation, Report	Verilog Design for second Rojobot instantiation and Flag icon idea

References:

[1] Rojobot31 Functional Specification, by Roy Kravitz
[2] Rojobot Theory of Operation, by Roy Kravitz
[3] Pmod JSTK2 Reference Manual
[4] Daniel J. Ellard, ’MIPS Assembly Language Programming, CS50 Discussion and Project Book’, September, 1994