Project 2 Theory of Operation

Introduction

This document shows the theory of operation of our second project: RoJoBOT world. Our implementations satisfies all of the requested functionalities from part a and b of project specification [1], as well as giving the following additional features:

- Utilizes a custom map, created using BOTWorld Development Kit
- Uses an 8-bit color icon (allows 256 unique colors for icon as opposed to 4)
- Uses BOTSim 3.1, which allows for bot wheel thresholds to be adjusted on-the-fly.
- Includes a UART embedded in robot control program for debug purposes.

More information about project specifications is described in the provided theories of operation and references [1],[2], and [3] at the end of this document.

Hardware Design

Top module: nexys3fpga.v

Below is the top module integrating both parts of the project:

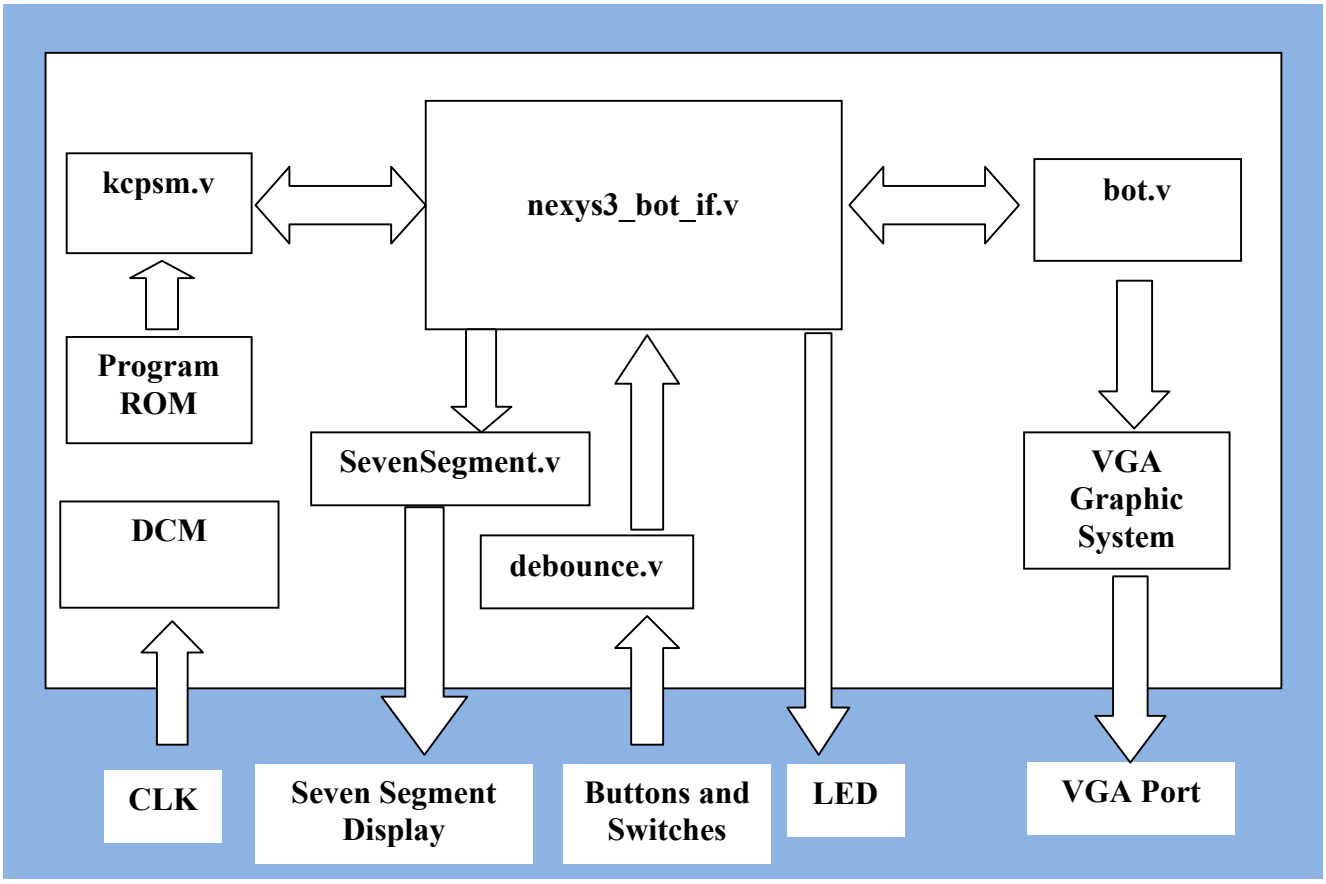

Figure 1 – nexys3fpga.v top module

- Picoblaze core (kcpsm.v): Controls robot behavior. The core is attached to a dual port 2K program memory, with JTAG enabled for rapid firmware deployment.
- BOTSim core: Simulates robot behavior in its world
- Interface module (nexys3_bot_if.v): relays information between robot control module, simulator and peripherals
- DCM : digital clock manager module; provides clock signals for the entire system (including 25MHz clock for the VGA subsystem)
- VGA subsystem: graphic logic that drives VGA display with RoJoBOT world map and robot icon.
- SevenSegment.v: 7-segment display controller
- debounce.v: debouncer for button and switch signals

BOTSim Hardware Interfacing

Below is the reference design (described in project specification) for module interfacing. Provided modules are: kcpsm6.v, SevenSegment.v, debounce.v, bot.v

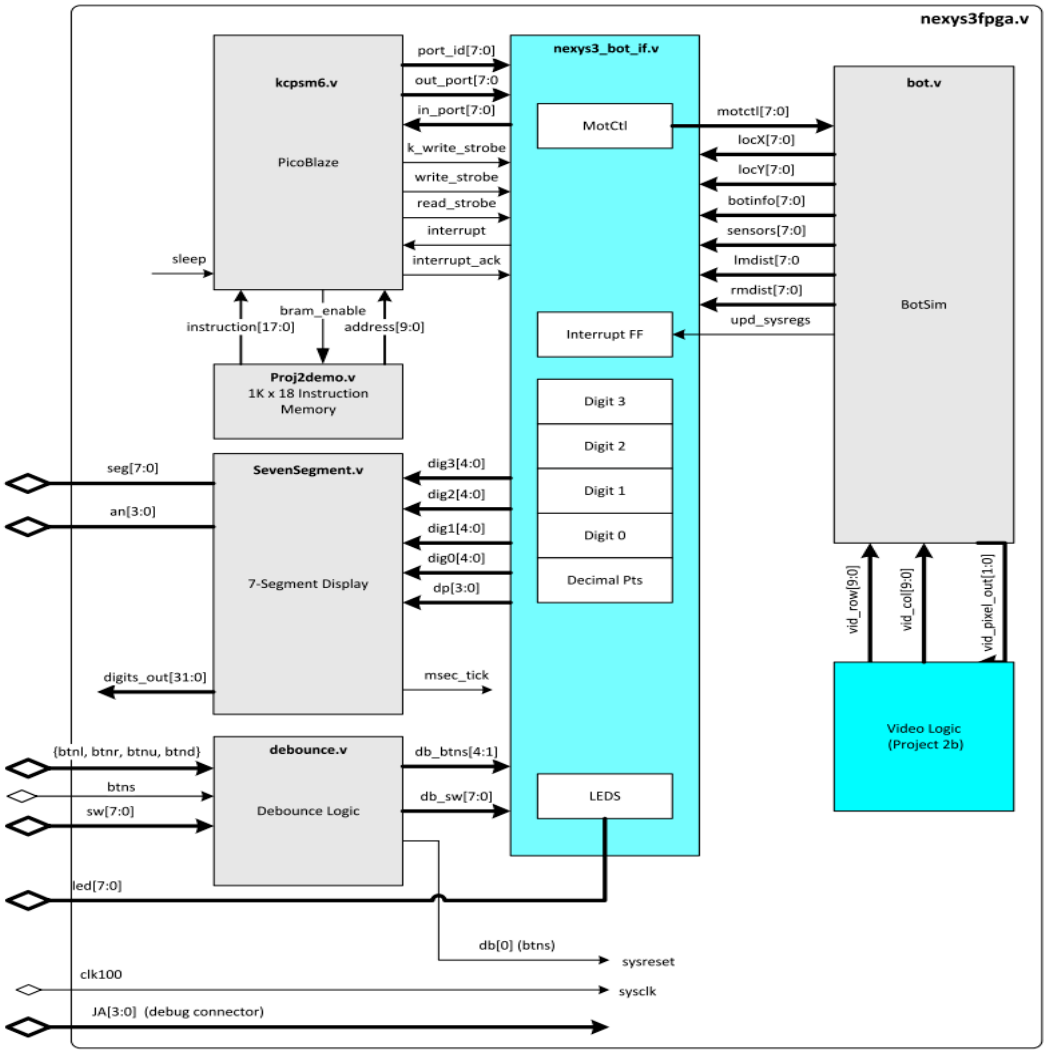

Figure 2 - nexys3fpga.v reference design

Interface module: nexys3_bot_if.v

This module implements the reference design below, providing an interface between the BOTSim simulator core, the PicoBlaze core controling robot behavior, and the Nexys3 board peripherals.

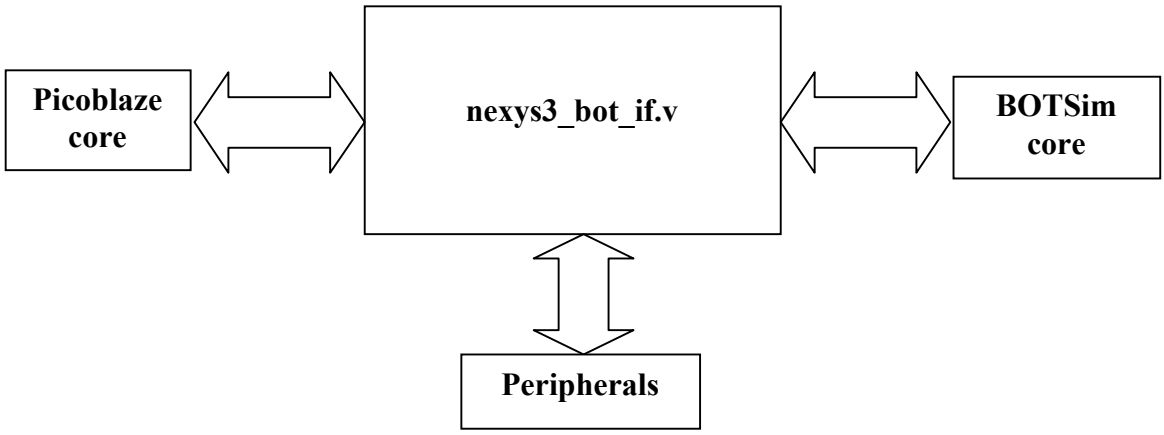

Figure 3 – interface module nexys3_bot_if.v

Module logic involves the following:

- A simple interrupt controller provides an interrupt signal to the PicoBlaze core when upd_sysregs is asserted. The interrupt is cleared when interrupt_ack is received in response from the PicoBlaze core.
- A UART controller is embedded and connected to the PicoBlaze core for debugging purposes.
- Signals are relayed between the PicoBlaze core and other modules using the following methods:

Input logic

```
always @(posedge clk)
      if(write_strobe == 1'b1)
         case (port_id[3:0])
         // LEDs output at address 02 hex
         4'h02: LEDS <= out_port;
        // Digit 3 output at address 03 hex
         4'h03: dig3 <= out_port[4:0];
```

Similarly for output logic:

```
always @ (posedge clk)
    case (port_id[3:0])
        // Read 4 pushbuttons status at port address 00
        4'h00 : in_port <= { 4'b0000, db_btns };
        // Read slide switches status at port address 01
        4'h01 : in_port <= db_sw;
        // Read X coordinate at port address 0A
        4'h0A : in_port <= locX;
        ...
```

Video Graphic System

This module "draws" a picture of the world with the robot icon following the black line and displays the picture on a VGA monitor/projector. The 128 x 128 map is rendered as a 512 x 512 image along with the robot icon, a 16 x 16 image. In the reference design below, the video graphic system includes:

- Display Timing Generator (provided): issues VGA timing signal using 25MHz clock.
- bot.v (provided): issues appropriate world pixel at current drawing row/column
- Icon: issues appropriate robot pixel at current drawing row/column
- Colorizer: combines world pixel and robot pixel to output VGA RGB signal

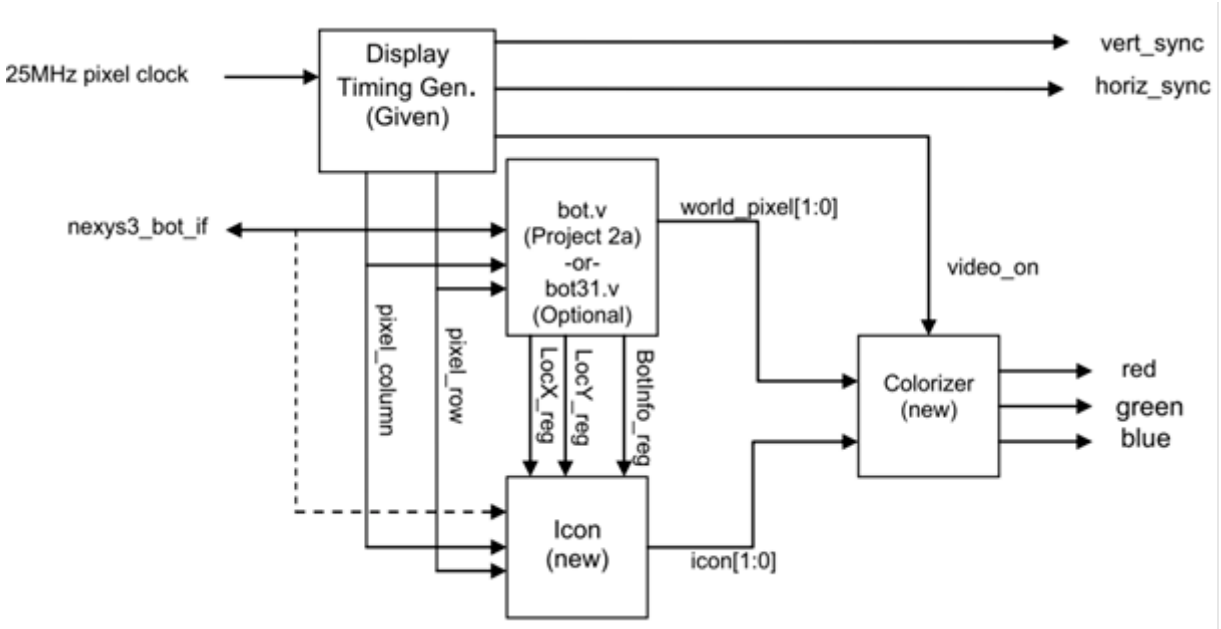

Figure 4 – Video graphic System (reference design)

Note: In our design, icon module outputs 8-bit color to the colorizer, unlike the reference design, which outputs 2-bit color code to the colorizer, which then selects an 8-bit color.

Icon module: Icon.v

This module includes a Single Port ROM module which stores the robot icon as well as logic blocks for remapping icon pixel into the world map.

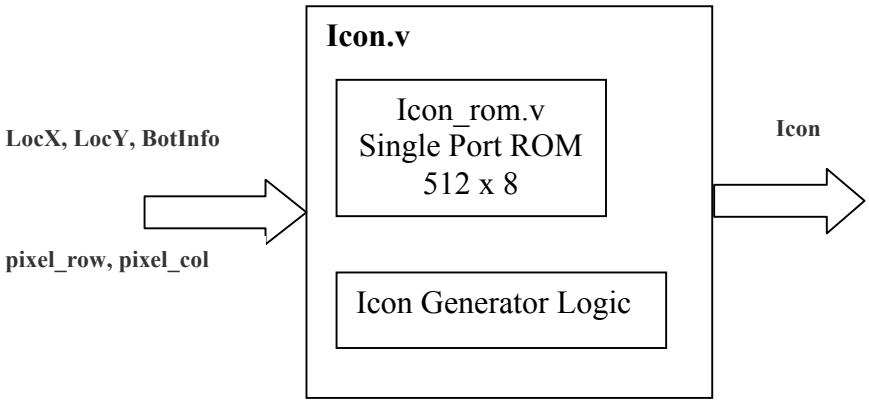

Figure 5 – icon module

Single Port ROM module (icon_rom.v): Stores two 16 x 16 icons (normal orientation and 45 degree orientation). The ROM size is 512 x 8. This module was generated using Xilinx Core Generator tool (Block Memory Generator). Parameters are:

- Interface type: Native
- Memory type: Single Port ROM
- Read Width: 8
- Read Depth: 256
- Always Enable
- Initialize memory file: <ROM_IMG>.coe (see appendix)

Icon Generator Logic: generate the appropriate pixel using current drawing row and column, the logic involved perform the following:

- Remap icon pixel to 512x512 map. This is done by calculating the start_X and start_Y pixel value, which is the top left corner pixel of icon, using the current LocX and LocY. Remap also considers aligning robot icon so that LocX and LocY is at the center of icon, as well as clipping in case robot move to display edge. Subtracting start_X and start_Y from current column and row's VGA index gives the current ROM index.

- Determine appropriate icon orientation. Using two basic orientations of icon: 0 degree (north) and 45 degree (northwest), we can obtain all eight orientations of robot in its world. This is done by:
    1. Selecting the basic icon for rotating
        - first icon: lower 256 bytes, for orientation of 0, 90, 180, 170
        - second icon: upper 256 bytes, for orientation of 45, 135, 225, 315

2. Rotating the icon by mean of recalculate the indexes of the icon in ROM. Assuming pixel of the original orientation is icon[y][x], the rotated pixel is:
   - 90 degree clockwise: Icon[original height - x][y]
   - 90 degree anticlockwise: Icon[x][original width - y]
   - 180 degree : Icon[original height - y][original width - x]

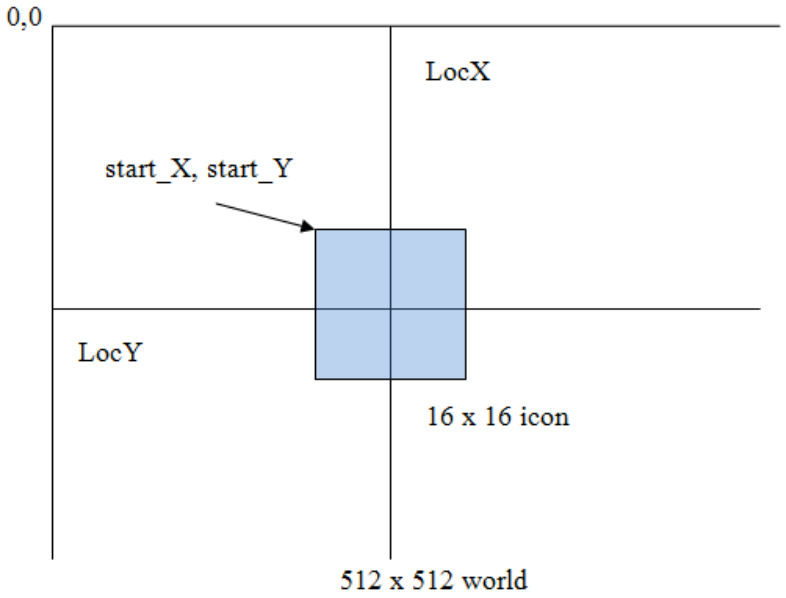

Figure 6 – icon pixel remap

Colorizer module: (colorizer.v)

The colorizer receives inputs from bot.v and icon.v and determines what is actually drawn on screen. The colorizer maps the 2-bit input from bot.v to four unique 8-bit colors and sends the result to the red/green/blue signals used for VGA. Since an 8-bit color icon was used, the colorizer simply passes the icon bits through with no processing. Priority is given to the icon; if it is present (any color other than black, 0x00) it will be the output. Otherwise, the map will be displayed. Hence, the robot icon appears "on top of" the world map.

Software Design

Two major changes were made to the reference psm design. All references to/functions for pushbuttons were removed, and the next_step function was rewritten to control the robot.

The next_step function now implements a state machine, described in figure 7 below. The robot always starts in NORMAL MODE. Because of how the reference design was written, it was necessary to return all the way to main after issuing instructions of any kind to the RoJoBOT, hence the state of the bot is stored in memory, then fetched each time next_step is executed. The first thing next_step does when called is fetch MODE from memory, and perform a case statement on it. Next_step then calls the matching subroutine. All subroutines and transitions are shown in the figure 7 below.

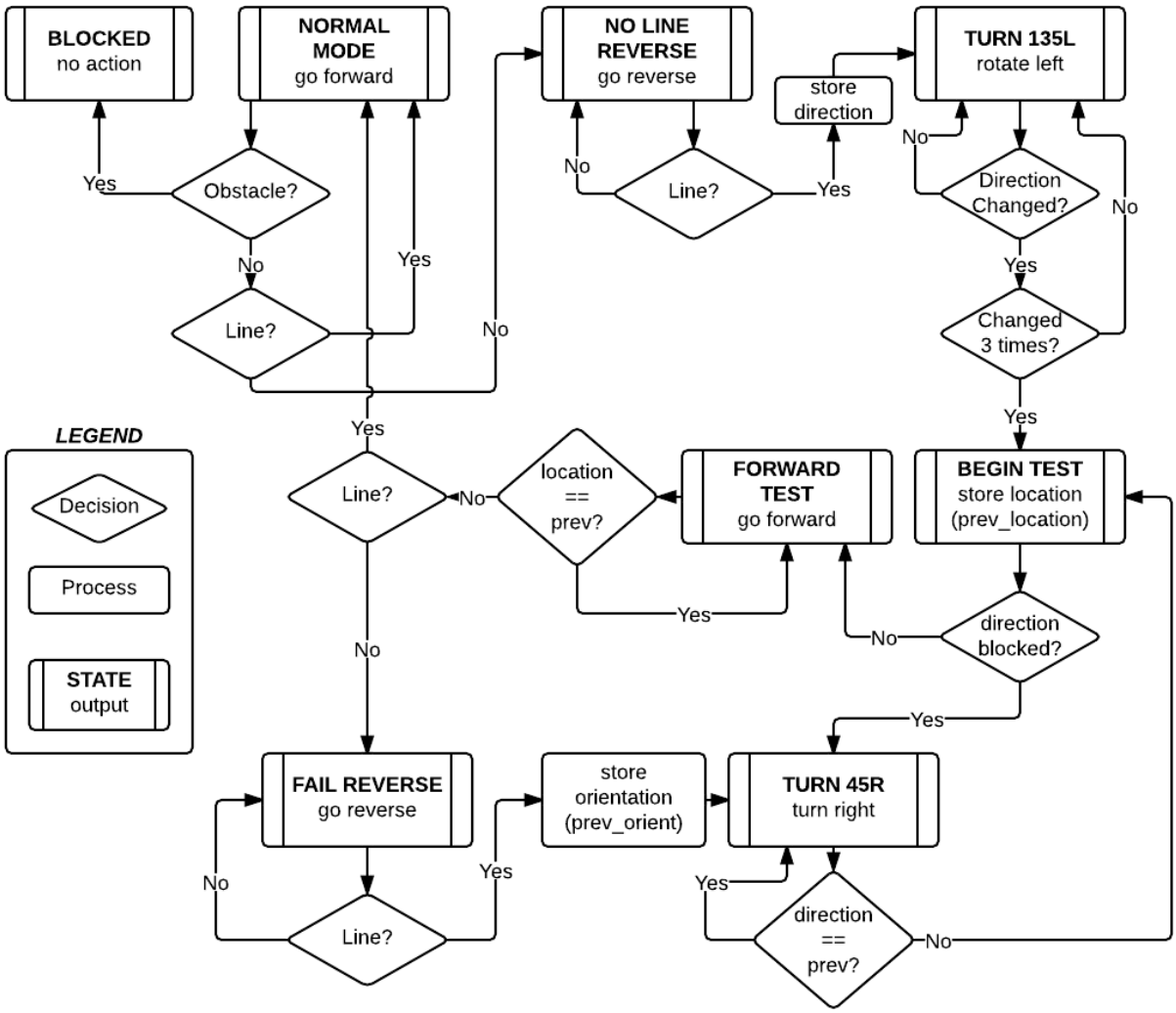

Figure 7 – Robot Action Flow Chart

As illustrated above, the bot navigates both right and left turns using a modified right-turn-only algorithm. Upon losing the line, the bot backs onto the line, first does a 1-time left turn of 135 degrees, and then makes right turns only until it finds the line. A dead end with no obstacle to stop the bot would cause it to eventually turn 180 degrees and go back the way it came.

Project I/O Specifications

The project receives user input from the switches and pushbuttons on the Nexys3 development board, as follows:

Switch[0] Display Mode: When switch[0] is **'off'** the program displays the location of the RoJoBOT in its "world" on the 4 digits of the 7-segment displays. When switch[0] is **'on'** the program uses the 7-segment digits to display movement (Stopped, Forward, Turning, etc.) and heading in degrees.

Switch[1] Enable movement: When switch[1] is **off**, the RoJoBOT will not make any movements until switch[1] is turned on again. Interrupts continue to run, and all other interface still function normally When switch[1] is turned **on**, RoJoBOT will resume whatever action was underway when the switch was turned off.

Switch[7:3] Wheel Threshold: Switches [7:3] define a 5-bit binary number which directly sets the wheel movement threshold for the wheel motors. Values lower than 4 are interpreted as 4.

Center Button: Software reset.

The project provides output through the 4 7-seg displays and 8 LEDs on the Nexys3 board, as well as through the VGA output.

7-Seg When switch[0] is off, the robots location is displayed in hex as follows:
({*digit 3, digit 2*} **,** {*digit 1, digit 0*}**)**
When switch[0] is on, digit 3 indicates movement, and digits 2-0 display heading in degrees.

Dec. Pts During normal operations, the decimal points chase right to left. Upon hitting an obstacle, only decimal point [0] flashes.

VGA Displays the map of the BOTWorld and animates the Bot's actions in it.

All other inputs/outputs/ports are unused.

Task Summary

Task	Implemented by	Note
Nexys3fpga.v	**Dung Le**	
Nexys3_if_bot.v	**Dung Le**	
Icon.v, Icon Builder	**Dung Le**	
Bot_control.psm	**Eric Krause**	
Colorizer.v	**Eric Krause**	
Motor Threshold Control	**Eric Krause**	**Handled by HW, added to nexys3fpga.v**

Appendix

1. Deploying picoblaze source using JTAG loader
    a. Enable JTAG LOADER in program memory module

```
bot_control #(
    .C_FAMILY            ("S6"),
    .C_RAM_SIZE_KWORDS (2),
    .C_JTAG_LOADER_ENABLE       (1))
```

    b. Configure the FPGA board with the bit file
    c. Edit the psm source
    d. Make sure kcpsm6.exe, JTAG_Loader.exe, msvcr100.dll and ROM_form.v module is in the build folder
    e. Run **loader.bat** script to compile and upload new firmware

2. Making Icon
   Icon builder**:** Constructs 16 x 16 pixel icons, in both orientations and packs them into a single .**coe** file (used by Core Generator). This is done with a python script, which:
    - Converts an 16 x 16 image input to 3byte–RGB image
    - Compresses the 3byte–RGB image to array of 1byte–RGB pixels using the most significant bits of the three colors, specifically, 3 bits RED – 3 bits GREEN – 2 bits BLUE. This format can be output directly to the VGA RGB signal lines.

   Usage:
    a. Find or make a 16 x 16 image for icon
    b. Install python and python image lib http://effbot.org/downloads/PIL-1.1.7.win32-py2.7.exe
    c. Modify the IconBuilder.py with image name
    d. Run the script to generate the .coe file.
    e. Use this .coe file during generating ROM module (with Core Generator)

Reference

[1] R. ECE540 Project 2a, 2b (RoJoBOT world) Specification, Fall 2012.
[2] BOTSim 2.0 Theory of Operation, Rev 2.0 (Digilent Nexys3 Release), Fall 2012.
[3] BOTSim 2.0 Functional Specification, Rev 2.0, Fall 2012.