Artem Kulakevich & Jonathan Law
Professors Brian Cruikshank & Ataur Patwary
TA Deepen Parmar
ECE-540 System on Chip
04 Nov 2020

**Rojobot Project**

**Theory of Operations:**

Below is a top-level view of our project design. The robot implementation is separated into 2 sections. The rojobot_interface and the rojobot_ctl file. The interface file holds the wishbone communication register and the 100Mhz to 75Mhz domain handshake register. The data from the interface is fed into the rojobot_ctl file which holds the Rojobot IP, map, icon colorizer, VGA map scalar, and final colorizer.

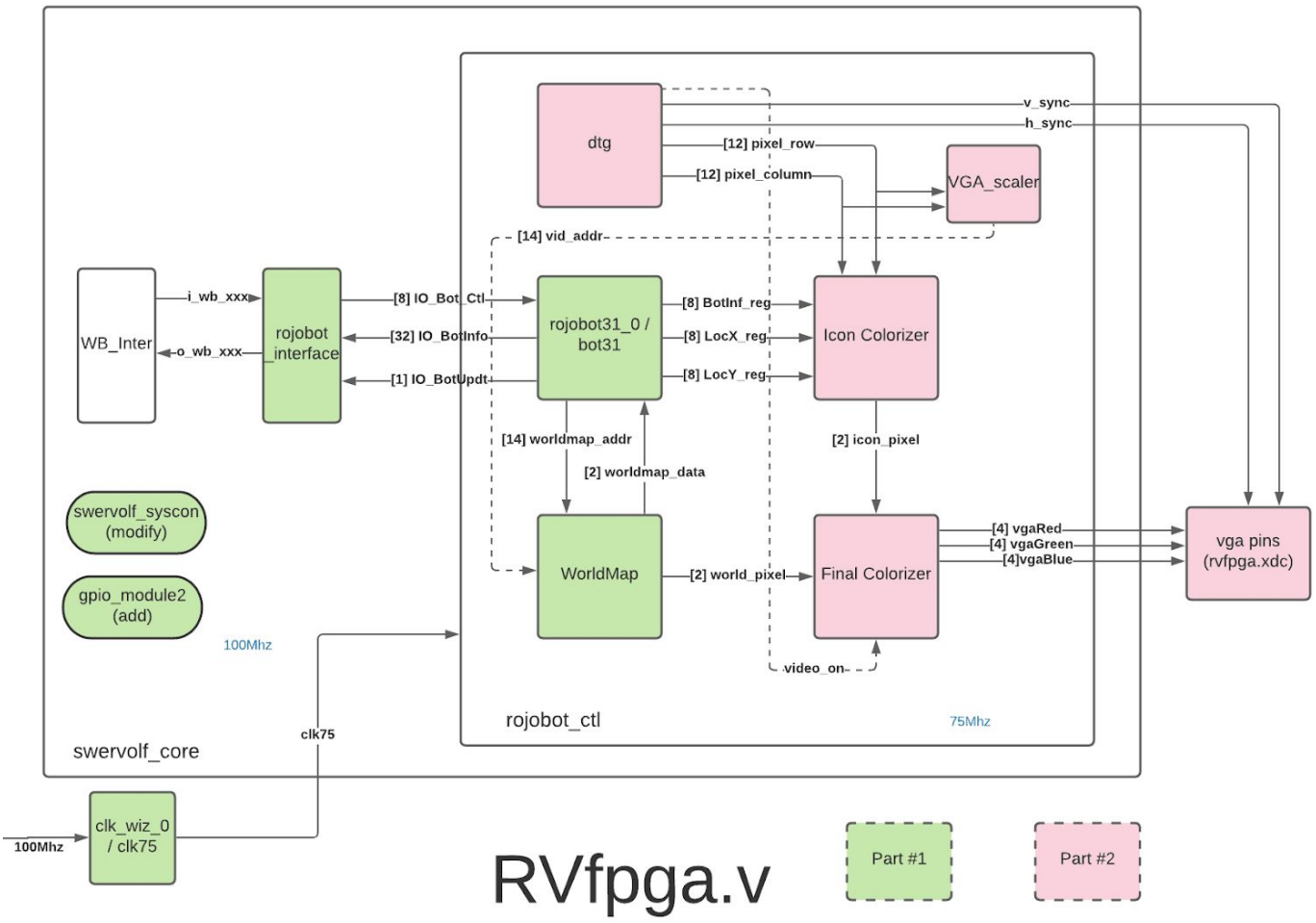


**Figure 1:** Rojobot block diagram.

This implementation is all stored inside the swervolf core and connects to the outside world from the final colorizer to the VGA pins.

**Preliminary Button / SevenSegment**

The buttons, 7segment display, and VGA outputs were all modified to allow for part 1 of the project to work. These modifications were copied directly from simplebot. Modified files include the wishbone interface to allow for button and robot connections.

**Final Colorizer**

Colorizer receives map and pixel data from the world_map module and icon module. It then compares the world_map pixel with the icon module pixel information. If the icon is a value greater than 0, then it overrides the world_map pixel data and displays a color based on the icon pixel. If the icon pixel is equivalent to 0 then, it allows the world map data to be displayed. The final colorizer then outputs a 12-bit color directly to the VGA pins defined in the XDC file.

**VGA Scaler**

VGA scaler uses the screen pixel from the provided dtg module as the reference point. It then scales the map from 128 to 512 and centers it in the image area 1024x768. Anything outside the 1024x768 turns the pixel off. It then creates the world map address needed by the world map module. We were also able to scale the map to 1024x768 without much trouble:

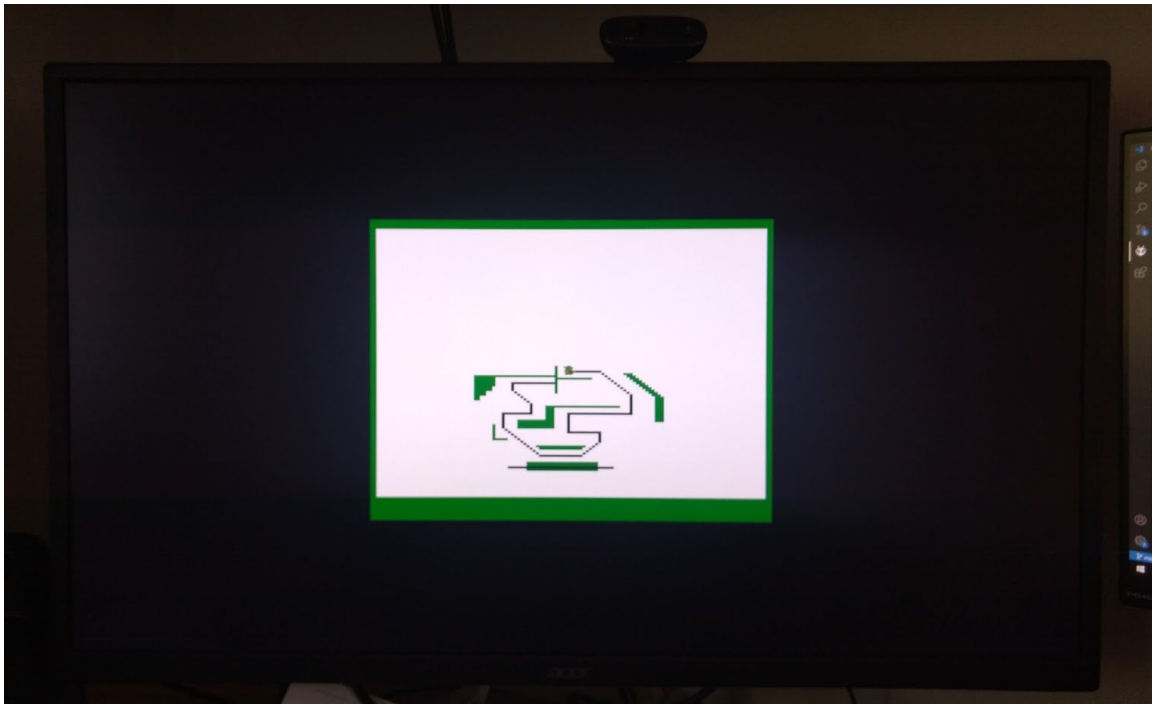

**Figure 2:** Rojobot 1024x768 image on a 2560x1440 monitor set to 1:1 ratio.

Our demo video was recorded with VGA set to the aspect ratio, this way the image was easy to see, but not stretched out by the monitor.

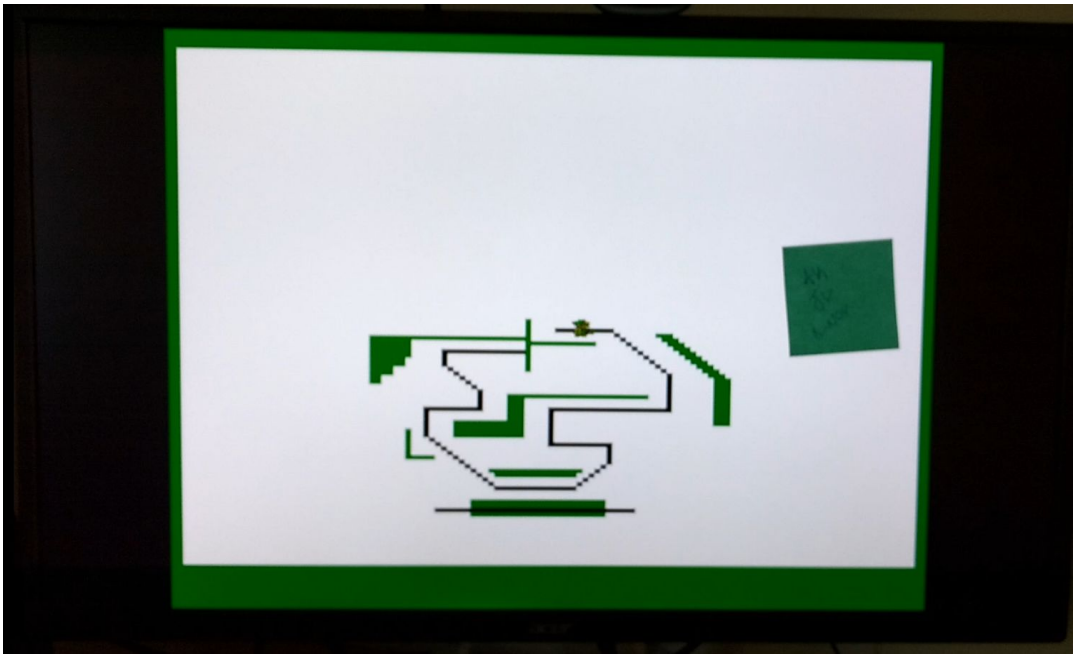

**Figure 3:** 1024x768 adjusted by monitor settings.

The scaling was done exactly as requested by the lab requirements.

**Icon Colorizer**

Icon colorizer takes the rojobot location and scales it up to match the scaled up world map. The icon space is then made relative to the world map. After the space is defined, the icon pixels are grabbed for display from a bitmap.

The icon bitmap is stored in a 16x16 space using strings to represent each pixel color. The icon looks animated by using multiple bitmaps as it moves along the line. The direction the rojobot is facing influences which bitmap is used and how it is rotated. Once the correct pixel color is determined. It translates the string color code to the 2bit color code needed by the final colorizer.

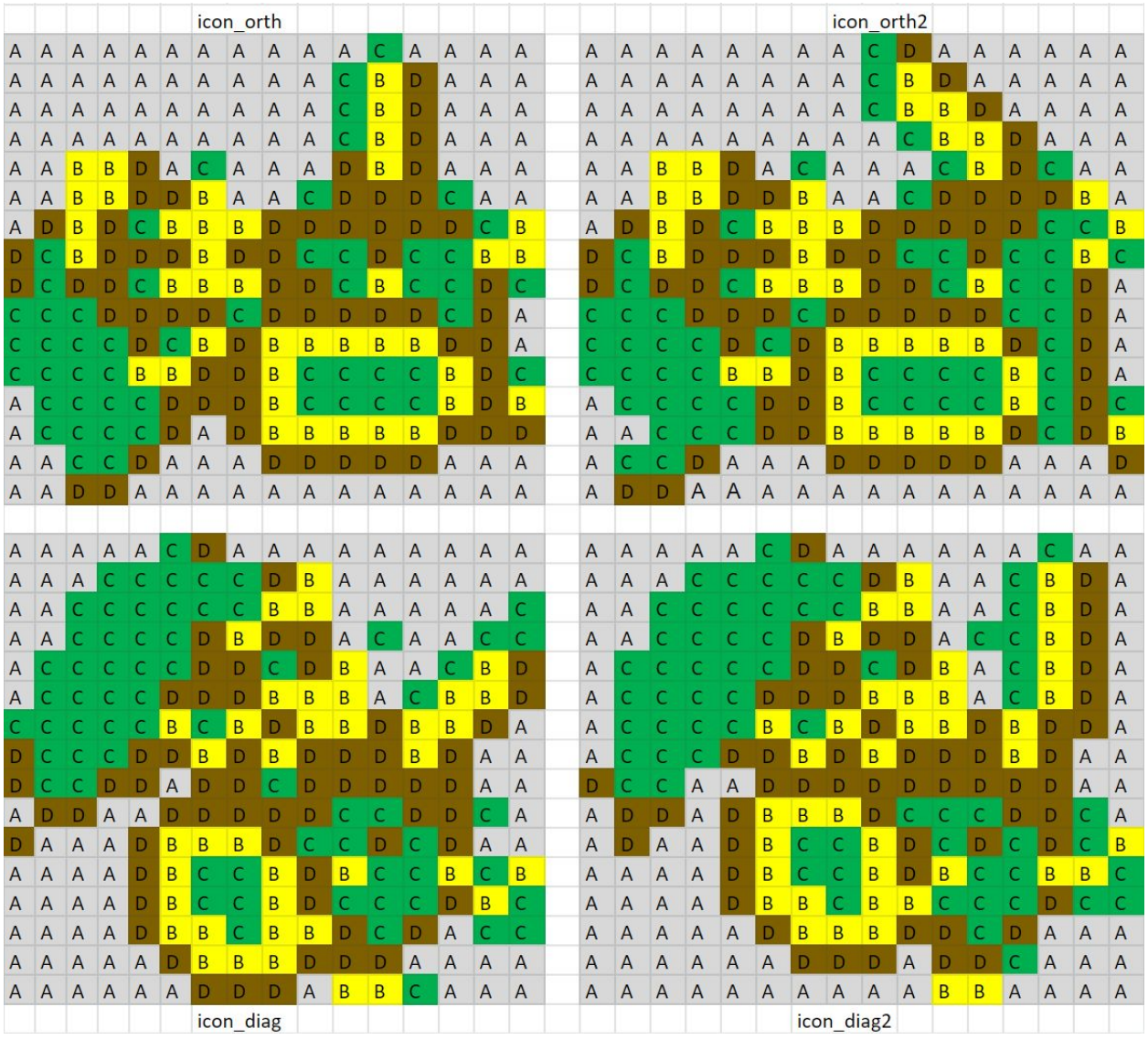


**Figure 4:** Icon drawing

**Robot Interface**

This is the module that handles the communication between Rojobot and Wishbone. It has IO_BotUdpt, IO_BotInfo, and IO_BotCtl as inputs from Rojobot, and the standard WB connections as inputs from wishbone. It does the 2 cycle WB read/writes that were recommended in the slides. It also has the recommended handshake register. The connections were modified to work with our wiring names, but functionally this code is equivalent to what was given.

**Assembly Code**

Here is a flow chart of the program that controls the rojobot.

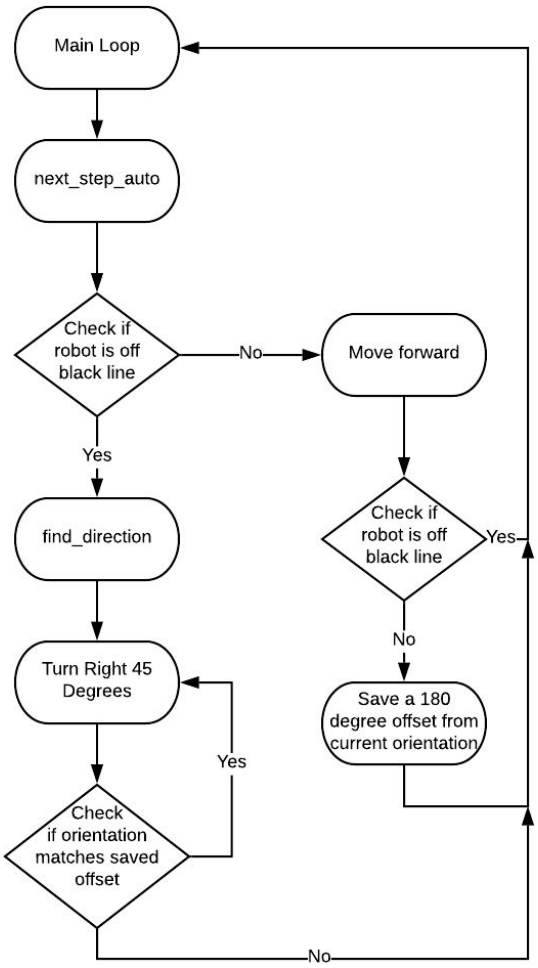


**Figure 5:** Black line flow chart

The place holder functions that were created in the provided for part 2 in the assembly file were mostly were ignored. The code is almost entirely implemented under the next_step_auto function. The assembly is very simple, it checks if the robot is off the black line,

if it is then it goes into the “find_direction” function. This function moves the robot in reverse 1 block using the “go_dir” function, and then it rotates the robot 45 degrees clockwise using “turn_45_right”. If the robot reaches a previously saved offset value, then it turns again.

The “go_dir” function is fed in the value from register “a1” which holds a motor direction value like SP_LFRF or SP_LRRR (Both motors forward or Both motors reverse). Both the “go_dir” function and the “turn_45_right” function read the BOTINFO port to verify that they have finished completing their action. Otherwise, the bot would have no time to turn before we moved onto a different instruction.

Another significant portation of the code is the part that saves the offset direction used for left turns. This offset is only saved after rojobot is still on the blackline after a forward move command. This way we only record good values that are in the direction of the black line. If the robot moves forward, and it is not on the black line after the move forward, then it’s possible that rojobot in the process of turning. We want to avoid saving those values because rojobot will then replace it’s offset value every time it turns.

It is necessary to get a 180 degree offset, so we know what turn has to be skipped. Given the value from BotInfo, 4 is added to offset it 180 degrees, and the value 8 subtracted if the saved value has overflowed above 7. For example, the south direction would return an 04. The function would then add 4, resulting in 08. This is overflow, so we subtract 8, resulting in the value 00. That represents the north direction, so we offset from south to north, and saved that direction in a register.

A couple of functions were also modified for this project. For example “check_off_black_line” originally reads data from a memory location that it populates at a different time. I modified this function to read the data directly from the rojobot port so that my implementation would function properly.

| The main functions added / modified: |
| --- |
| ● next_step_auto<br>● find_direction<br>● check_off_black_line<br>● turn_45_right<br>● go_dir |

**Conclusion**

In conclusion, we were able to hit all of the requirements. We have an image that had a visible orientation at 0, 45, 90, 135, 270, and 315 degrees. We displayed out to a VGA port with proper scaling. Our bot was also able to go the full left/right track without backtracking on the blackline. Our code is also well documented and described using flow charts and diagrams. This project was a great introduction to all the various aspects of the FPGA demo board and a great success.