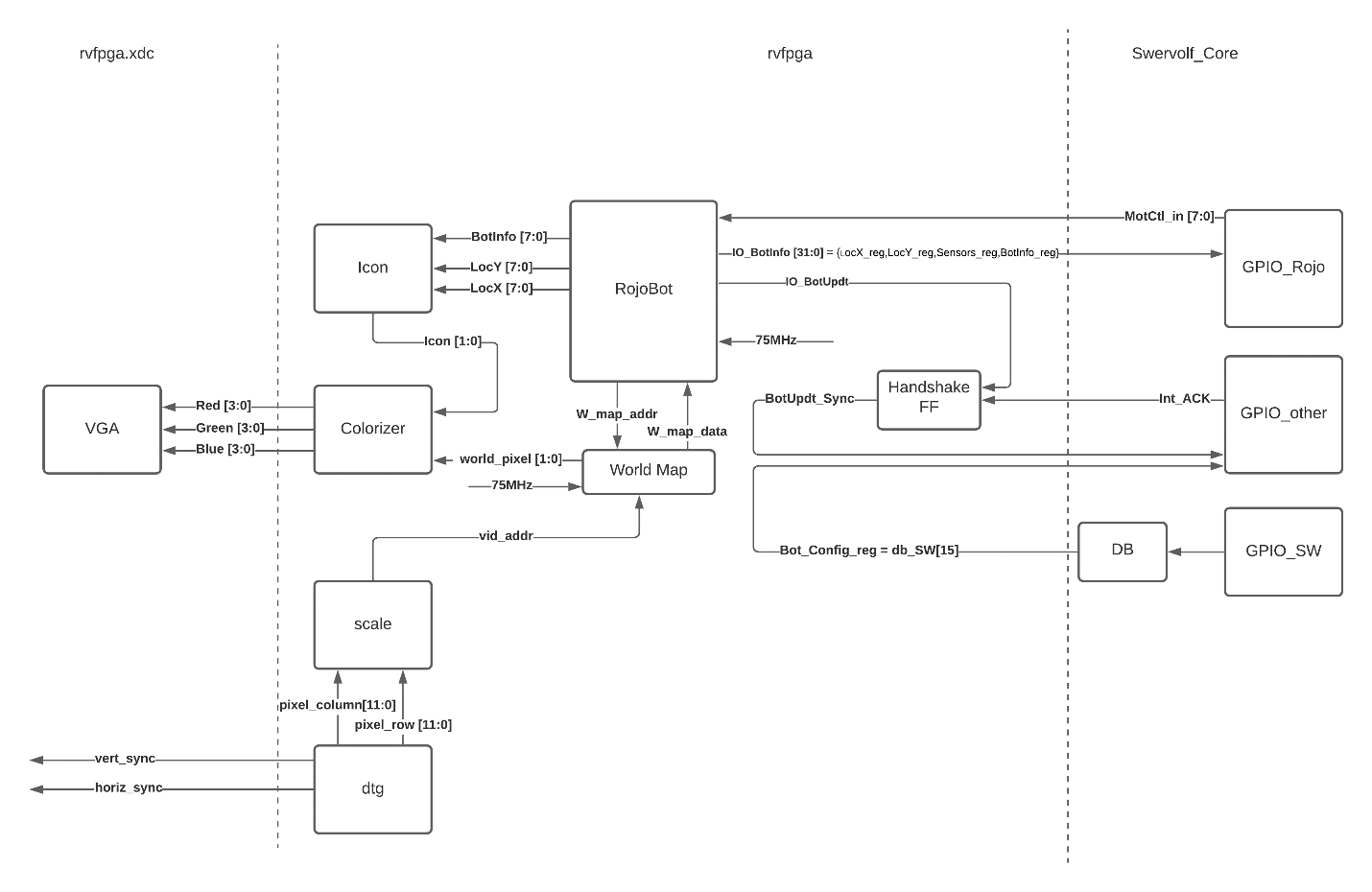
ECE 540

Project 2: Rojobot World Video Controller

By Katie Loseke, and Sonali Fernando

03/02/2021

In the Rojobot World project we used hardware and software components to be designed, implemented, and integrated into a working system. Figure 1 provides a top-level design view of our hardware components. Our project was divided into two parts. The first part consists of getting the basic components of the Rojobot World connected to verify the basic functionality of the Rojobot is correct.



**Figure 1: Rojobot Hardware implementation**

For part A, the Rojobot emulation file was given to us as supporting material added to our design in the Vivado IP block. The Rojobot emulator is connected to a handshaking FF. This handshaking FF ensures to update the state of the Rojobot. The handshake FF was instantiated in the rvfpga.v file. In rvfpga we instantiate the world map, and Rojobot. The map addresses are provided by the Rojobot to indicate the movement of the Rojobot and based on the address the world map provides the stored world map information for the given address. The clock frequency for the Rojobot and the world map is 75MHz. This is generated from the Vivado’s clock generating wizard, where the default 100MHz clock is converted to a 75MHz clock using a clock divider.

In the swervolf\_core.v file three GPIO (General Purpose Input Output) ports were added to accommodate the 8-bit motion control signal, the 32-bit motion controller signal consisting of X and Y location signal, the sensors signal and Botinfo signal, 8 bits each. Additionally, the acknowledgement signal and the bot update sync signals are also connected as an output and an input respectively to the GPIO.

**Buttons and 7-segment display**

The implementation of the buttons is the same as project 1. The implementation of the 7-segment display was done where the extended bit is selected as high to enabled 0 to f in hex digits, and extended digit is selected as low to select a single segment or to select a specific letter (H, L, R, l, r) to display on the 7-segment display digit.

**Colorizer module**

In the colorizer module, it checks to see if the video is on or off. If the video is on then, check to see if the icon pixel signal is out. If yes, then check to see if the world map pixel signal is out. If yes, then set the background to green, black line to black, and obstructions red, the base icon color to teal, icon eyes to blue, and icon second color to white. If the video is off then, set all red, green, and blue VGA signals to 0 and that leads to the display to turn off.

**Scale module**

In the scale module we are taking in display pixel rows and columns from the DTG (Display Timing Generator) and dividing the display pixel column by 8 and pixel row by 6 to match the icon ratio to the pixel ratio and concatenate the calculated value to be the video address value.

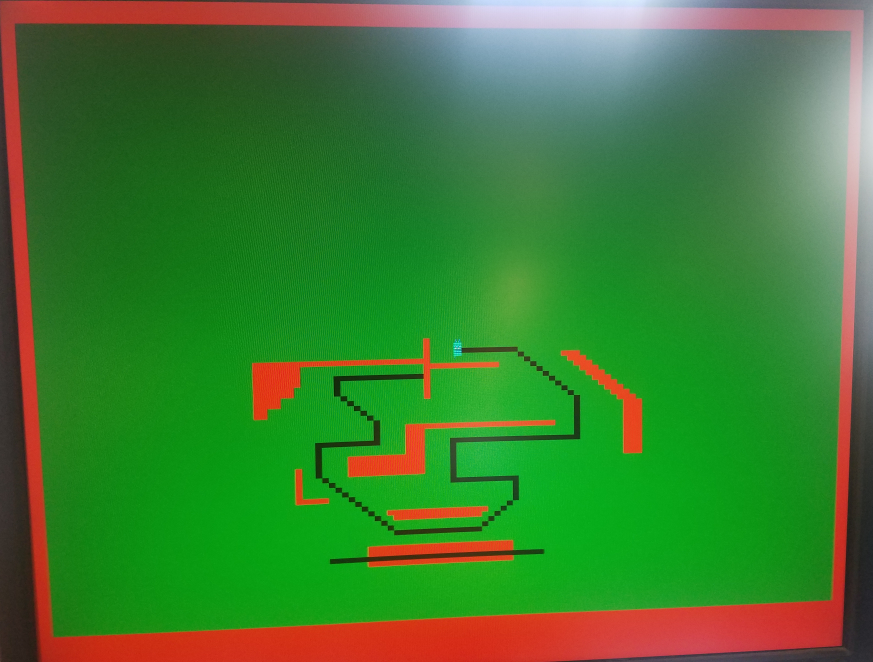


Figure 2: The world map and the bot displayed on the 1024x768 monitor.

**Icon module**

In the icon module you check the actual location of the Rojobot and using a case statement we are checking to see if the orientation changed. If the orientation of the bot has changed then, we redraw the Rojobot icon’s pixels to match the direction it is facing. If we are, check orientation (case) to see which pixel to write to.

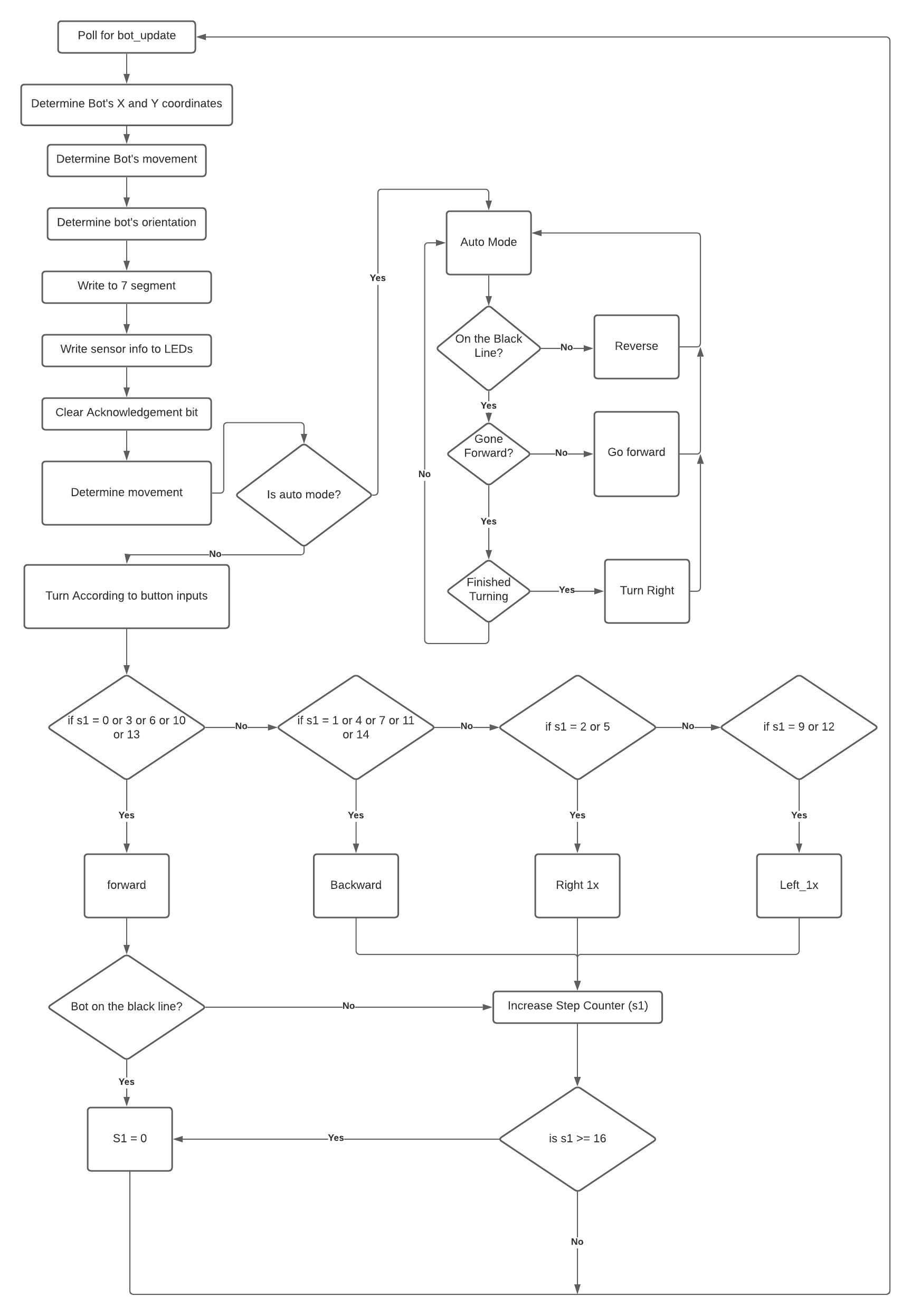


Figure 2: the flowchart of the Rojobot Assembly Code

**Assembly Code**

First, we poll for bot to be updated. Then we determine the x and y coordinates. Next, we determine the Rojobot’s movements, orientation, and write this information to the 7-segment display. We write the sensor information to the LED’s. Next, we clear the acknowledgement bits. We determine the next movement and decide on the Rojobot’s turn according to the button press.

Based on the state of the Rojobot we implement forward, backward, right 1x or left 1x. When moving forward we check to see if the Rojobot is on the black line. If yes, we reset S1 and poll for bot updates. If no, we increment the step counter S1. If the step counter S1 reaches 16 then the cycle is complete, and we will reset the step counter and poll for bot updates. If the step counter did not reach 16 then we check for bot updates without resetting the step counter.

In Auto mode we decided to check to see if we are on the black line. If we are not on the black line then we reverse to the black line. If we are on the black line we ensure the robot is moving forward and ensure finished turning after going forward.

**Conclusion**

This project had many issues in getting it to work. Mainly getting it to follow the black line and getting the icon to appear in the correct location. For the black line, the bot would often get off of it and wander around, this was fixed by adding a check to see if the bot changed orientation before we issued another turn command and following a right turn only algorithm. For the VGA controller, we thought that the world map was getting shrunk by two horizontally, but it turned out that the robot was actually getting doubled horizontally. This was fixed by doubling the input pixel address for the bot.