Project 2: RojoBot World and Black Line Following Software Theory of Operation

By Michael Bourquin and Jean Shirimpaka

I. Introduction

In this project, we use the Diligent Nexys4 DDR development board to model a Rojobot icon in a virtual world (displayed via VGA). The Rojobot, a "virtual" robot, is a two wheeled mobile platform with proximity and IR sensors(for black line sensing and following). We emulate the mobile platform and its world in an IP (intellectual property) block, Rojobot31. Rojobot31, an SoC design containing a Xilinx Picoblaze (firmware in program memory and logic to manage its interface) is treated as a "black box", which is monitored and controlled from the MIPSfpga system. In addition, the Rojobot icon is set to transverse the hardwares virtual world by successfully making turns and following a black line until stopped by an obstacle.

II. Method

Design of this project is consist of integrating both hardware and software, respectively described in section II.1 and section II.2 below. Hardware development consists of implementing an icon-based video controller that connects to a VGA monitor through the VGA connector on the Nexys4 DDR. This required an addition of a top level module (nexys4_ddr.v) that includes the Rojobot31 IP(previously described + MIPS slave drivers), a clock generator that generates both 75 MHz and 50 MHz clocks, the world map, some synchronization logic to handle signals that cross between the two clock domains, and VGA logic to map a 16X16 icon and 128X128 world map to the screen. Software development consisted of implementing an application (based of the provided Project2Demo program) that causes the Rojobot to traverse a virtual black line in its virtual world until it is stopped by a virtual wall. This application reads the Rojobot registers and drives the motor control inputs to direct the Rojobot to follow the black line.

II.1 Hardware Development

The hardware architecture consists of a modified top level module (*mfp_nexys4_ddr.v*), containing the following new and modified instantiations which will be explained in more detail under the following sections:

- 50MHZ and 75MHZ PLL (75MHZ new)
- MIPS System(GPIO slave modified from Project 1 to drive the Rojobot (black box).
- World Map (new)
- Handshake Flip Flop (new)
- Rojobot (provided IP)
- Icon (new)
- Delay World Pixel (new)
- Video Scale (new)
- DTG (new)
- Colorizer(new)
- Rojobot(new)

For an overview look at how all these modules are instantiated at the top level see Source/Hardware/Design/rojobotVGAiconDesign.png

50HZ and 75HZ PLL:

The Vivado clock generator was used to generate a clock module that outputs both a 75MHZ and 50MHZ clock from the FPGA's 100MHZ clock pin. The rojobot and display modules operate at 75MHZ and so are driven by the new 75MHZ clock. The following synchronous modules operate in the video domain(75MHZ):

- World Map
- DTG
- Delay World Pixel
- Icon
- Rojobot

MIPS System:
This module contains both the top level processor hardware and the bus / slave system to drive the connected peripherals. From the previous project, the following slave peripherals were already included:

- 7 Segment Display Driver
- GPIO Driver (including read interfacing to the switches and to the button inputs)
    - Read Interfacing
        - Switches (15 bits)
        - Buttons (5 bits including UPRL, and C)
    - Write Interfacing: LEDS

In order to drive the rojobot, the GPIO slave peripheral was modified so software can read from; and write (control) the robots current movement. The following was added to the GPIO driver (for reference see *mhb_ahb_gpio.v* under Source/Hardware/Design):

- Read Interfacing
    - BotInfo Bus (32 bits including sensor information, robot x/y location, orientation)
        - Mapping: Physical Addr -> h1f80000C, virtual addr -> hbf80000C
        - LocX: bits [31:24], LocY: bits [23:16]
        - Sensors: bits [15:8]
        - Bot Info: bits [7:0]
    - Rojobot update bit (set to 1 when the rojobot emits an update pulse unless ACK is 1
        - Mapping:
            - Physical Address: h1f800010 -> Virtual Address: hbf800010
        - Bits [0]
- Write Interfacing:
    - Bot Control (controls the movement and speed of the rojobot)
        - Mapping
            - Physical Address: h1f800014 -> Virtual Address: hbf800014
        - Bits [7:0]
    - Acknowledge Signal (acknowledges an update was received and resets the update signal to 0 until the next update pulse)
        - Mapping
            - Physical Address: h1f800018 -> Virtual Address: hbf800018
        - Bits [0]

The above address mapping information can all be found in *mfp_ahb_const.vh* under Source/Hardware/Design. All signals are directly connected to the rojobot module besides the update signal which is synced with the 50HZ MIPS clock before driving the slave. Also these four newly added signals are routed to the top level module for the required connections to be made.

World Map:

This module was provided and contains an instantiation of the 128X128 world map contained in dual port RAM. Both the Rojobot and the Colorizer use the data outputs (a [1:0] bus) of this module concurrently.

The address selects for A and B are generated by the Rojobot and Video Scale modules.

Handshake Flip Flop:

This module synchronizes the bot update signal with the MIPS clock domain and outputs a synchronous update signal. The update signal can be reset with the ACK signal from a MIPS write.

Rojobot:

This is the provided module for the Rojobot31 which is instantiated under the top level module. The main outputs for the purposes of software control included are as follows:

- Bot Information
    - Sensors [7:0]
    - Bot Info [7:0]
    - LocX [7:0], LocY[7:0]
- Update Pulse (1 bit)

The rojobot pulls location information from the world map ram via an address output.

The main input is the control bus [7:0] which is directly connected to the MIPS SYS module.

Icon:

The main purpose of this module is to map the current Rojobot location (128X128 space) to the current value of the display pixels (768X1024 space). This mapping determines the display pixel offset of the icon image. If the display pixels are currently overlapping the rojobots location (which happens to be a 24X32 display space), then the display pixels are mapped to an address in the 16X16 icon space determined by the offset + (24X32). Every icon pixel also happens to map to a 1.5X2 pixel space on the display: (24X32)/16.

Instead of using dividers case statements were used to determine the icon indices. In the case of finding the row index, a range pattern was used to approximate the 1.5 pixel mapping. The following patterns of ranges were used: 112, 122, 112, 122… -> end is whatever range <= 2 leftover. For example index 0 would be any display pixel between offset and offset + 1. Index 2 would be any display pixel between offset + 2 and offset + 4.

To simplify, subtraction can be used to take the difference if icon is overlapping as follows:

- Currentpixelrow – offset = range(0-24). Then similar case statements are used to find the icon index.

For the purposes of this module eight (each a different icon orientation) static read only 256 address (16X16) memory arrays are initialized from input image files. Once the address has been determined from the above, the orientation input defines which icon memory should be read from.

The icon 2 bit output is synched to the 75MHZ clock similar to the world pixel output being synched via its generated RAM block. Icon outputs to the colorizer module and is always transparent (0) when the current display pixels do not overlap with the rojobots location on the screen.
In order to solve found timing issues some of the case statement logic was pipelined and the full pipeline takes 2 clock cycles until the currently received logic is valid when changed.

Simulation:
Because it was difficult to debug errors with the above logic, a testbench was developed for the icon module by looping through overlapping display pixels and making sure the correct icon bits were output. The simulation file used can be viewed under Source/Hardware/Simulation

Reference Files/Folders:

- Source/Hardware/Design/icon.v
- Source/Hardware/Design/iconmem

Delay World Pixel:

This module delays the 2 world pixel outputs from the world map RAM block by 1 clock cycle to stay in sync with the icon pixel (takes 2 clock cycles for changing address to be valid)

Video Scale:

This module maps the current display pixel output from the DTG module to the 128X128 world map space by using range case statements. Every pixel in the 128X128 space is 6X8 pixels on the display. The output is an address which generates an output from the world map. The world map output connects to the colorizer.

DTG:

Provides the horizontal output display signals for the VGA pins + the video on signal. The current display indices are also output and connect directly to both the video scale and icon modules. The display indices are continuously looped through at a 60HZ refresh rate.

Colorizer:

A purely combinational logic module with the following inputs from world map RAM, icon, and the DTG modules:

- VideoOn
- Icon Bus
- World Pixel Bus

This module provides the output RGB video PIN logic for a VGA display. The default output is black. When icon is 0(transparent), the world pixel input is always used. The color mapping is below:

- Icon: 0 (green), 1 (blue), 2(yellow)
- World Pixel: 0(white), 1(black), 2(red)

For reference see *colorizer.v* under Source/Hardware/Design

II.2 Software Development

In the software part of this project, a MIPS Assembly application is implemented to read and control the Rojobot direction based on the blackline detection and display its orientation angle in degrees(0, 45, 90, 135,180, 225, 270, and 315) on a seven-segment display. As illustrated in the below state machine of Fig. 1, our algorithm is to move forward until the Rojobot is no longer over the black line and reverse until it finds the back line again. After the Rojobot finds the black line it makes a right turn, moves forward again until it loses the black line, and so on. The algorithm also assures that the Rojobot avoids backtracking along the path the Rojobot has already followed (avoids doing a 180º turn from the original orientation).

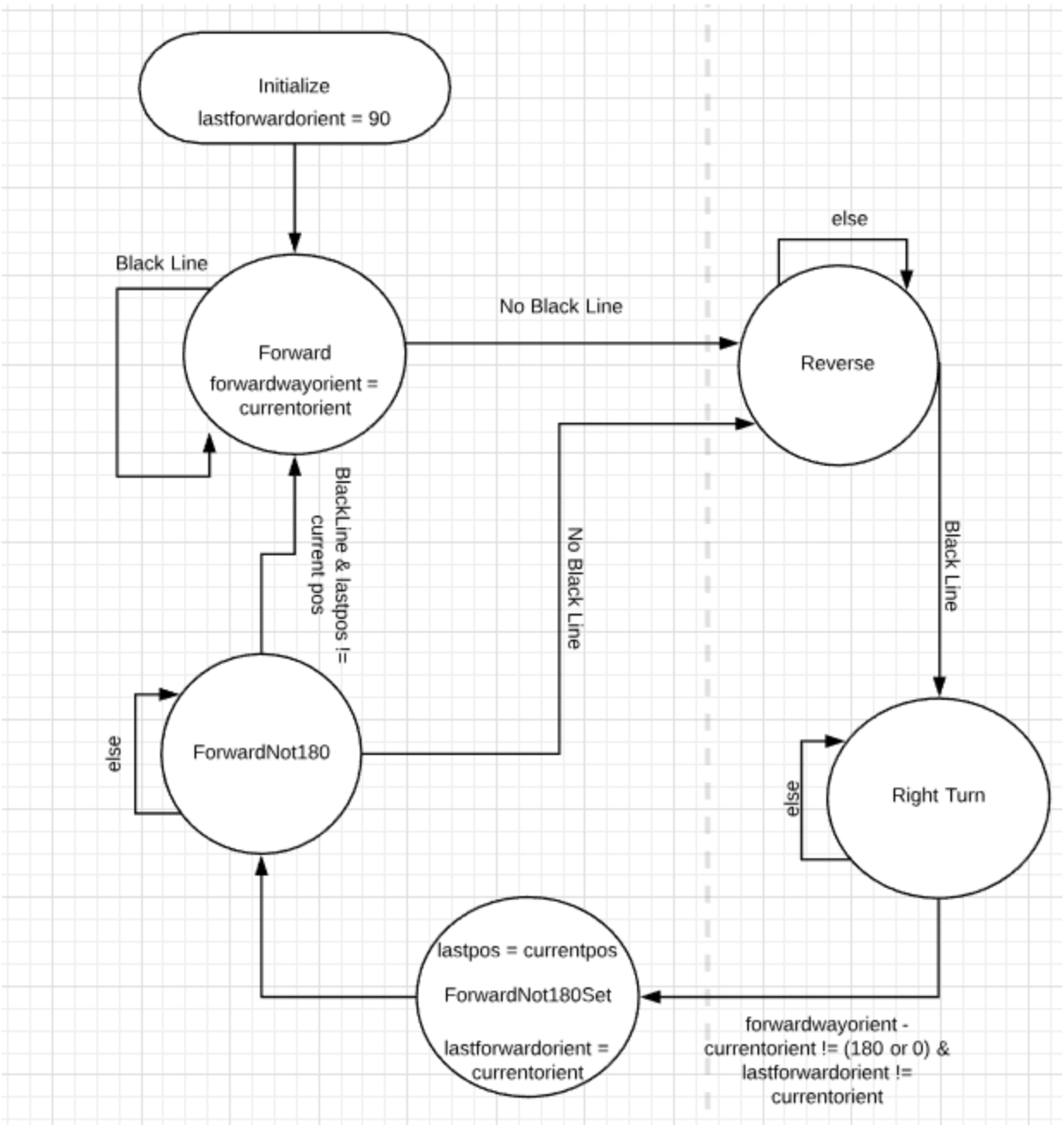

Fig.1: Transition diagram for Rojobot motion and orientation control.

Detailed Code Design Analysis:

The software design follows the below flow diagram(Fig.2) in which the program first initializes and immediately starts in the State Machine Output logic where depending on the state, a command is sent to the rojobot. Also a reference position and/or orientation may need to be saved future comparisons. Immediately after, the program continues to the update loop where until the Rojobot sends an update status (hardware registers have been modified), the program remains in a waiting state. When an update is

received, an acknowledge signal is sent to the hardware, where then the program continues onto the next state logic. From there, the output command is again updated and the acknowledge signal is reset, and the program again returns to the update loop (wait state); waiting for the next update. This is the main loop of the program.

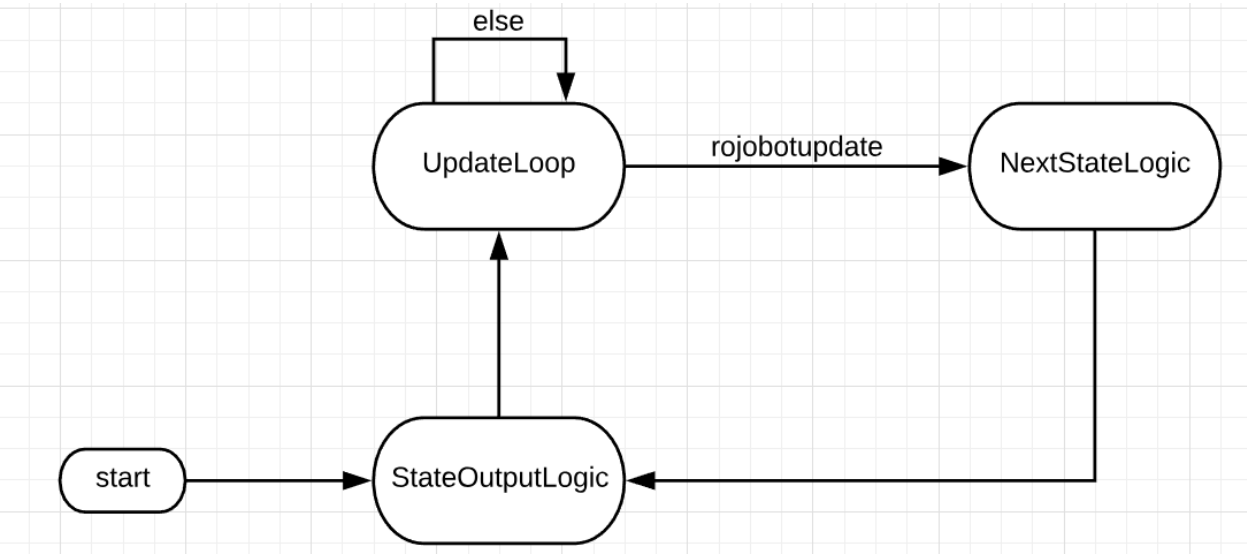

Fig.2: Transition diagram for the Rojobot's software flow.

The following is a description of the project's assembly program. Line numbers are referred to from the "main.S" file under Source/Software

Lines: 72 - 100
The main section of the software initializes all the required address registers, reads and displays the orientation of the Rojobot on the seven-segment display, and continues to the state machine out logic.

Lines 103 - 120
The "updatelooplogic function" section of the program, writes an acknowledgement signal to the acknowledge port($12) and continues to the nextstatelogic(implementation of the transition diagram in Fig.1 above) whenever there are any updates on the Rojobot's state registers.

Lines 123 - 130
The "nextstatelogic" statements check and determine which state(forward, reverse, rightturn, etc)code section to jump to in order to add the correct output (next orientation) to the rojobot's motion per the following output table. From each state to the next state, the rojobot needs to detect the black line by reading the black line value from the bot info register(through masking bot info with the black line mask value) and logically shifting the resulting data by 8 bits to the right as in the following lines of code: andi $22, $23, BLACKLINE_MSK

srl $22, $22, 8

Also, in the forward180 next state logic, the current position is checked against the last forward position. Once the next state is determined the program continues to the output state logic, of which the output commands are shown in table 1.

State	Output
0	FORWARD
1	RIGHT
2	REVERSE
3	FORWARD
4	FORWARD

Table 1: Output State Machine

Lines 132-312

This section of the code is the implementation of different states logic and their outputs to be written out to the bots control port($9). Also some states require saving current bot information into memory for future reference. Once complete, the acknowledge signal is reset (so that another update signal can be received from the rojobot) and the program continues on to the update loop once again. When another update is received, the entire process starts over again.

Line 316-428

This function outputs different orientation values based on the rojobot's inputs. The orientation values are also displayed on the seven segment display in this section of the code. Whenever there is an update from the rojobot this function is called.