Learning Objectives:

- Gain experience designing an SoC with an embedded CPU, 3rd party IP, and custom hardware
- Gain experience implementing an icon-based VGA video controller
- Gain experience writing an embedded system application

Note 1: Project #2 includes many hardware and software components that have to be designed, implemented, and integrated into a working system. This has proven to be a formidable task to undertake in one "chunk." To that end we have decided to break Project #2 into two smaller projects this term, each with their own set of deliverables. You will extend your Simplebot hardware from the first project and bring up the Proj2 demo system in Part A. You will also implement a black-line-following (only right turns) embedded system application with output to the 7-segment display and LEDs to complete this part of the assignment.

In Part B you will add an icon-based VGA controller to your design and extend your black-line-following application to handle both left and right turns. Even though you will submit twice to D2L (once for Part A and once for Part B) we will only grade your final submission from Part B.

Note 2: This document and the other documents, source code, etc. for Project #2 treats the word "Verilog" that same as the word "SystemVerilog." In actuality, there is no more standalone Verilog – what we used to think of as "Verilog" has been subsumed into the SystemVerilog language which retains backwards compatibility with the last approved Verilog specification – Verilog 2005. You do not need to write your code using SystemVerilog constructs such as always_ff, typedef, unique case, etc. although I (Roy) believes doing so will be to your advantage – SystemVerilog, imho, is a better HDL language for synthesis.

Project: RojoBot World

This project builds on the SimpleBot concepts introduced in Project 1 by placing a virtual two-wheeled mobile platform with proximity and IR sensors (for black line following) into a virtual world. The mobile platform and its world are emulated in an IP (intellectual-property) block called *Rojobot31*. *Rojobot31* contains a Xilinx Picoblaze, the Rojobot emulation firmware and logic to connect it to a register-based I/O interface peripheral. This functionality has been wrapped in a Vivado IP block that can be added to your design without having to understand the details of the implementation. You can treat *Rojobot31* as a black box that can be monitored and controlled from an RVfpga system.

The Rojobot31 receives instructions that control emulated motors connected to the wheels. The motors are controlled by writing a value to the *Motor Control Input* register. 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. There is also a register that your application can write to change some aspects of the emulation (the Rojobot's emulated speed, for example). The details are described in the *Rojobot31 Functional Specification …*a must read for anybody who is planning to control our intrepid little mobile platform.

You will be provided with small sections of the hardware design, a demo program written in RISC-V assembly language (firmware_part1) to test your implementation and plenty of documentation. The system you built for Project #1 is the jumping off point for the hardware for this project but you will need to add new circuitry, primarily a memory mapped I/O interface to the Rojobot registers. The Rojobot registers needs to be implemented as an additional slave much on the Wishbone bus. You will also be provided with an ECE540_IP(`proj2_release/ece540_ip_repo`) repository that contains the Rojobot31 IP.

You will need to instantiate:

- A clock generator that generates both 75MHz and 100MHz clocks in `rvfpga.sv`
- The Rojobot31and the World map logic in `swervolf_core.v`
- Some simple synchronization logic to handle signals that cross between the 100MHz and 75Mhz clock domains

You may need to:

- Modify your 7-segment controller functionality from Project #1. The Simplebot project only required the lower 4 digits of the 7-segment display controller. The Proj2Demo program uses all 8 of the digits of the display and the decimal points. Review your implementation carefully (specifically in swervolf_syscon.v) and make any changes required to support all 8-digits and the decimal point.
- Modify the Base Address constants in Proj2Demo.S to support your specific I/O register map.

Your application will read the Rojobot registers and drive the motor control inputs to direct the Rojobot to follow the black line. There are several algorithms that follow a black line. One of the simplest 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 again it should make a 45 degree turn to the right, move forward again until it loses the black line, and so on. This algorithm only works for a maze consisting of right turns and is the goal for Part A.

You will have to extend the algorithm to properly also handle left turns without backtracking along the path the Rojobot has already followed (your algorithm needs to avoid doing a 180º turn from the original orientation) for your final deliverables.

You may write your application in either RISC-V assembly language like the `Proj2.S` demo code or in C.

Deliverables - Part-A:

- Source code for the SystemVerilog module(s) you write. You do not need to include any simulation test benches or simulation results (simulation is optional).
- Source code for an application that uses the LEDs and 7-segment display to demonstrate that our application correctly traverses a right-turn only map.

Deliverables – Part B:

- Source code for the SystemVerilog module(s) you write. You do not need to include any simulation test benches or simulation results (simulation is optional). Please submit all of the SystemVerilog code you write even if it hasn't changed since you submitted Part A.
- Source code for your final application. This application should use your VGA and icon logic to demonstrate that your Rojobot can traverse both left and right turns
- Demonstration video of your project. Your demo video should show your Rojobot icon successfully traversing a black line with left and right turns. We will also expect the icon to visibly show its orientation (0°, 45°, 90°, 135°, 270° and 315°). Display will be on a VGA display.
- A *Theory of Operation* document (less than 10 pages). Your report should include a flowchart, state diagram or pseudo-code, and text explaining your black line following algorithm.
- All Vivado project files. You will zip the whole Vivado project file containing the xpr file. Please test between partners to make sure your partner can recreate your project on their computer

by only clicking the xpr file.

Grading

There will be a single grade for your final Project #2 deliverables on the following rubric:

- (50 pts) A successful demonstration video of correct operation on the Digilent Nexys A7.
- (20 pts) The quality of your *Theory of Operation* report where you explain your design.
- (15 pts) The quality of your design expressed in your SystemVerilog source code. Please comment your code to help us understand how it works. The better we understand it, the better grade it will receive. Lack of good documentation will lead to significant point losses. Industry requires good documentation, here is a good place to start building that habit.

- (15 pts) The quality of your program expressed in well documented and nicely structured. The more readable your code, the higher grade it can receive.

IMPORTANT: Both team members will receive the same grade on the project regardless of the amount of, or the quality of, the work completed by each.

Project Tasks – Part A

1a. Create a new project in Vivado

Since the 7-segment interface from Project #1 is also used in Project #2, you can open Vivado and create a new project from your Project #1 source code. Then add the files from the following directory:

- `hdl_part1/world_maps_part1` (for Proj2Demo)

`world_map.v` provides an instantiation template for all of the maps (we provide 3 maps). `world_map.ngc` is produced by the Vivado Memory generator and contains (among other things) the bits in the BRAMs that describe the map locations. To change maps all you need to do is remove the world_map.ngc file from your project and add one of the others and synthesize and implement the project again. All of the maps we provide are named *world_map* to simplify the process.

2a. Add the Rojobot31 IP to the Vivado IP Catalog

Add the `ece540_ip_repo` folder to the list of IP repositories for the IP Integrator to search. You can do this by clicking on `FlowNavigator/ProjectManager/Settings/Project Settings/IP/Repository`and adding the folder location of `ece540_ip_repo`.

3a. Instantiate and connect a clock generator, `rojobot31` and `world_map`.

1. Use the Clocking Wizard to create a clock generator with both 100 MHz and 75 MHz output clocks. The RVfpga system will be clocked at 100 MHz and the Rojobot, world map, and VGA controller will be clocked at 75 MHz.
2. In the IP Catalog search for *rojobot*. If you added the `ECE 540 IP` repository successfully you should see it in the `UserIP` folder. Double click on the IP to add it to your project. Let Vivado generate the Out-of-Context modules. Once the generation is complete you should be able to see your `rojobot31` instance in the IP sources tab in the Project Manager.
3. Expand the IP block and open the `Instantiation/rojobot31_1.veo` file. This file provides the code to instantiate the rojobot31 in your design. Do a cut/paste to add the instance to your design.
4. Modify `swervolf.v` and carry the changes through your design hierarchy to add four I/O ports, `IO_BotCtrl, IO_BotInfo, IO_INT_ACK and IO_BotUpdt_Sync`. These four new ports will be connecting to RVfpga. You could implement a newWishbone peripheral to connect to the Rojobot or add another GPIO instance to your design and connect the registers (each of the Rojobot registers is 8-bits wide. The implementation details are left to you.
5. Read the Project2Demo assembly code for the physical addresses you will need to assign to the ports (or use your own port assignments and change the Proj2Demo code to match).
6. Follow the block diagram from Proj2Demo Example design description´ to create connections between the Rojobot, RVfpga system and the world_map.
7. Add the handshaking flip-flop logic to your design. This flip-flop is described in *Proj2Demo Design Description*. The `upd_sysregs` signal from the Rojobot clock domain (75 MHz) needs to be synchronized to the 100 MHz RVfpga clock domain. The flip-flop is controlled by the Proj2Demo application program.

4a. Synthesize the demo system

Synthesize and implement your Vivado project. Check the warnings after synthesis for the usual suspects such as mismatched port widths, signals that are not driven, signals that are tied to 0, signals driven by more than one source, etc.

***HINT**: It is useful to elaborate the design (`Project Manager/RTL Analysis/Open Elaborated Design/Schematic`) and look through the schematics, since adding an I/O port involves changes to multiple levels of hierarchy. It only takes a few minutes to Elaborate your design with respect to the time it takes to synthesize it. Look at the interconnect between the blocks - if any inputs are tied to ground there is likely something wrong in your design or in the way you passed signals through the hierarchy. With over*

200 warnings to examine after synthesis, doing this checking before synthesis could save you time. If you forgot to connect something properly synthesise will optimised out of the design and you will have no clue where you are going wrong.

5a. Bring the Proj2Demo code into Platformio and build the image

Create a new project in Platformio and import the files from the firmware directory. Make changes to the port assignments and code as needed to get the demo program working on your hardware. You need to make your Rojobot automatically follow the black line and complete the specific map properly. Understand the code given by us and edit the specific section to make it running manually. I have added comment from where you need to edit your code.

6a. Download the demo system to the Nexys A7 board and run/debug the demo

If your Verilog additions and modifications are correct the programmed FPGA will respond to the pushbuttons and switches, illuminate the LEDs and display the Rojobot activity on the 7-segment display as described in the *Proj2Demo Example Design Description*.

7a. Download the demo system to the Nexys A7 board and run/debug the demo

Either make a copy of, rename, and modify, *`proj2Demo/main.S`* to implement the black line following algorithm or create a new application using C instead of assembler. Test your black line following algorithm by simulating sensor values with and without the black lines and with and without obstructions. Be sure to ³take ownership´ of any code you borrow; this means updating the header, adding more comments, etc.

Neatness and clarity in your source code is important in this course, especially if you are writing Assembling language. The easier the code is to follow the happier we will be.

8a. Download the .bit file to the Nexys A7 and use PlatformIO to test your application

Verify that the display operates as specified in the functional specification. Verify that your Rojobot does, indeed, follow the black line to the ending wall by checking the performance of your Rojobot against the expected results.

9a Submit your Part A deliverables to your D2L dropbox

Submit clean copies of any SystemVerilog files you wrote or modified and your line following program for Part A. You do not need to submit a report or a demo. The Part A deliverables serves as a checkpoint to your team and to us that you are on track towards completing the project. Part A will not be graded, other than to verify that you have made your submission.

Project Tasks – Part B

1b. Design, implement, and debug the Rojobot world video controller

The Project 2 deliverable includes a video interface to a VGA-compatible display. You will be able to see your Rojobot move around the world map, which is very nice! Instead of trying to follow the seven segment display to check that your Bot is navigating the black line The write-up for this part of the project is in *Rojobot World Video Controller* document.

2b. Synthesize, implement, and generate a .bit file for the new system

Study the logs for synthesis errors and warnings. There may be some that could keep your design from operating correctly. Pay special attention to any critical errors. Make sure that your constraints file (`.xdc`) matches the ports at your top level (`rvfpga.v`). For example, you will need to add the VGA signals to the .xdc file after you have added the VGA controller to your design and carried its ports to the top level of the hierarchy.

3b. Refactor/rewrite your Part A application to traverse both left and right turns

In Part A the application was only required to traverse a maze using a right-turn only algorithm. Refactor your application to traverse mazes containing both left and right turn. The trick for the traversal algorithm to handle this is to eliminate the backtrack case. That is, consider a right-only turn algorithm where the Rojobot follows the line until it loses it and then backs up, turns to the right, and tries again. There are left turn cases where the Rojobot will make consecutive right turns until it finds the black line and then proceeds to move forward. If the Rojobot has turned 180 degrees it will most certainly find the black line it just came from and dutifully retrace its path back to START. One way to eliminate this case is to do enough checking to skip over the backtrack case.

4b. Download the .bit file to the Nexys A7 and use PlatformIO to test your refactored application

Verify that the display operates as specified in the functional specification. Verify that your Rojobot does, indeed, follow the black line to the ending wall by checking the performance of your Rojobot against the expected results.

5b. Submit your Part B deliverables to your D2L dropbox

- Submit clean copies of any SystemVerilog files you wrote or modified even if they haven’t changed from Part A
- Submit a copy of the source code of your application for Part B.
- Write a Design report for the full Project. Your report should include a flowchart, state diagram or pseudo-code, and text explaining your black line following algorithm. Please include a section on challenges you encountered along the way and how you overcame them. We’d also like to “hear” your thoughts about how the project could be improved. Project #2 is as it is today largely because of feedback we received from students that had completed the project. Also, please include a work breakdown describing what each team member was responsible for. Keep in mind that the main goal of your Theory of Operation and your source code organization and comments is to provide those of us grading your project with insight into how your implementation
- Produce and submit a video of your demo showing that the Rojobot correctly traverses the LR track and that your icon follows the path and shows the orientation of the Rojobot.

References

[1] Xilinx Vivado Software Manuals
[2] *Rojobot31 Functional Specification*, by Roy Kravitz
[3] *Rojobot Theory of Operation*, by Roy Kravitz
[4] *Proj2Demo Example Design Description*, by Sarvesh Kulkarni and Roy Kravitz
[5] *RojoBot World Video Controller*, by Roy Kravitz
[6] ECE 540 Project 2 List of Files