Learning Objectives:

Project 2

- 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 assembly language for the MIPSfpga softcore CPU
- Learn SoC simulation (optional) and debug techniques

Project 2 demonstrations will be on Wed Nov 2nd and Thu Nov 3rd.
Project 2 deliverables are due on D2L by Sun Nov 6th @ 10:00 PM.

You will work in teams of two on this project

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* is a SoC design in its own right; containing a Xilinx Picoblaze, firmware in program memory and logic to manage its interface. This functionality has been wrapped in a Vivado IP block that can be added to your design without having to understand the implementation. You can treat *Rojobot31* as a "black box" that can be monitored and controlled from an application CPU (in this case a MIPSfpga system).

The Rojobot31 receives instructions that control 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 (Proj2Demo) 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…most notably a memory mapped I/O interface to the Rojobot registers. The Rojobot registers could be added to the existing `mpg_ahb_gpio` module or you could implement an additional AHB-Lite slave for the Rojobot registers much like you did for the 7-segment display for Project #1. There are advantages and disadvantages to both approaches, but the design decision is yours to make. You will also be provided with an `ECE540_IP` repository that contains the Rojobot31 IP. This IP block can be added to your design in a manner similar to the way you added the clock generator in Project #1 using the Vivado IP Integrator. You will create a new top level module (`nexys4_ddr.v`) that includes the Rojobot31 IP, a clock generator that generates both 75 MHz and 50 MHz clocks, the world map, and some simple synchronization logic to handle signals that cross between the two clock domains.

Once you have the Proj2Demo design example running you can start on your Project 2 application firmware and hardware. From a hardware perspective you will implement an icon-based video controller that connects to a VGA monitor through the VGA connector on the Nexys4 DDR. From a firmware perspective you will add code to the Proj2Demo application that causes the Rojobot to traverse a virtual black line in its virtual world until it is stopped by a virtual wall.

Your application will read the Rojobot registers and drive the motor control inputs to direct the Rojobot to follow a black line in its virtual world. There are several algorithms to 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 so you will have to extend the algorithm to properly also handle left turns without backtracking along the path the Rojobot has already followed (you're your algorithm needs to avoid doing 180º turn from the original orientation) .

Deliverables

- Demonstration of your project to the instructor or the T/A. In the demo we will expect your Rojobot icon to successfully traverse a black line with left and right turns. We will also expect the icon 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.
- Source code for the Verilog module(s) you write. You do not need to include any simulation test benches or simulation results (simulation is optional).
- Source code for the MIPS assembly language that you successfully demonstrate with. You do not need to include earlier versions of code.

Grading

There will be a single grade for this project based on the following rubric:

- (50 pts) A successful demonstration of correct operation on the Digilent Nexys 4 DDR.
- (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 Verilog source code. Please comment your code to help us understand how it works. The better we understand it, the better grade it can receive.
- (15 pts) The quality of your program expressed in well documented Assembly language code. 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. You will be able to re-form teams for subsequent projects if you desire.

Project Tasks

1. Download the Project 2 release package

Download the Project 2 release from the course website. The project 2 release .zip file contains the ECE 540 IP repository and world_map, MIPSfpga assembly language code for Proj2Demo, and documentation. A description of the files is provided in *ECE 540 Project 2 List of Files*.

2. Create a new project in Vivado

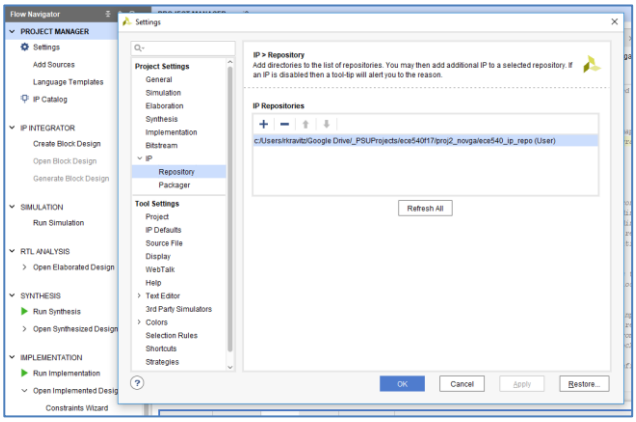
Since the 7-segment interface from Project #1 is also used in Project #2, you can open Vivado and create a new project from your Project #1 source code. Then add the files from the following directory:

- `hdl_part1/world_maps/ world_maps_part1 (for Proj2Demo )`

`world_map.v` provides an instantiation template for all of the maps (we provide 3). `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.

3. 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 `Flow Manager/Project Manager/Settings/Project Settings/IP/Repository` and adding the folder location of `ece540_ip_repo`. In my Vivado installation it looks like this:



4. Instantiate and connect a clock generator, rojobot31 and world_map.

1) Use the Clocking Wizard to create a clock generator with both a 50 MHz and 75 MHz output clock. The MIPSfpga system will be clocked at 50 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. 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.

3) Modify `mfp_sys.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 MIPSfpga through a GPIO peripheral like the switches and pushbuttons. You could add the registers to the existing GPIO or you implement a separate AHB-Lite peripheral to connect to the Rojobot. There are advantages and disadvantages to either approach. 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).

4) Follow the block diagram from "Proj2Demo Example design description" to create connections between the Rojobot, MIPSfpga system and the world_map.

5) Add the handshaking flip-flop logic to your design. This flip-flop is described in Proj2Demo Design Description.docx. The `upd_sysregs` signal from the Rojobot clock domain (75 MHz) needs to be synchronized to the 50 MHz MIPSfpga clock domain. The flip-flop is controlled by the Proj2Demo application program.

5. Synthesize the demo system

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

HINT: I (Roy) have found it useful to elaborate my design (`Project Manager/RTL Analysis/Open Elaborated Design/Schematic`) and look through the schematics because adding an I/O port involves changes to at least 4 levels of hierarchy and `mfp_ahb_const_vh`. It only takes a few minutes to Elaborate your design (vs. the 20 minutes it takes on my laptop 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. I know that doing so saved me hours of work.

6. Bring the Proj2Demo code into Codescape and build the image

Create a new project in C4E 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.

7. Download the demo system to the Nexys4 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*.

8. Write and debug your new program

Make a copy of, rename, and modify, `proj2Demo/main.S` to implement the black line following algorithm. Test your black line following algorithm by simulating sensor values with and without the black lines and with and without obstructions. Be sure to "take ownership" of the code; this means updating the header, adding more comments, etc. In this course neatness and clarity in your source code are important. The easier the code is to follow the happier we will be. 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 works.

9. Synthesize and implement the completed system using Vivado

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 (nexys4_ddr.xdc) matches the ports at your top level (nexys4_ddr.v). For example, you will need to uncomment the VGA signals once you have added the VGA controller to your design and its ports to your top level.

10. Download the system to the Nexys4 and test it

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.

Congratulations! You have completed the first part of the project.

11. Design, implement, and debug the Rojobot world video controller

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

NOTE: PLEASE RETURN THE JUMPER SETTINGS ON ANY OF THE NEXYS4 DDR BOARDS IN THE LABS TO THEIR DEFAULT SETTING (JTAG DOWNLOAD, SO THAT THE SELF-TEST STARTS UP WHEN POWER IS APPLIED)

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*

<finish>