**Project #1: SimpleBot**

**Due date: October 21, 2020**

**Learning Objectives:**

Project

1

* Practice coding SystemVerilog for synthesis
* Build on the RVfpga Getting Started projects
* Practice assembly language coding RVfpga Processor
* Introduce the Digilent Nexys A7 development board
* Gain experience debugging an SoC embedded system
* (optional) Learn how to use GitHub and GitHub classroom

**Project:SimpleBot**This project models a very simple robot using the Nexys A7 board. The “virtual” robot is a platform with two wheels, each driven by an independent motor. A third free rotating wheel or Teflon skid serves to stabilize the platform.

![A picture containing graphical user interface

Description automatically generated]()

For this project, you will not use a physical robot; instead, we will model the robot’s operation using the Digilent Nexys A7 development board. In this first project we will use the pushbuttons, switches and seven-segment display to control your virtual robot’s wheel motors and show information about the robot’s motion.

***Your task for this project is divided into the Hardware and Software development:***

***Hardware Development:***

**Pushbutton:** The code provided to you for Previous Assignment doesn’t contain the pushbutton functionality. Thus, you need to implement that functionality into your HDL code. You can refer to the GPIO section of the **swerolf\_syscon.v** for LEDs and switches and learn how to design and instantiate your new pushbutton functionality in the same place.

**Adding debounce. v file**: You also need to add the debounce.v to get rid of the debouncing problem for the pushbuttons. This file will be added in **swerolf\_core.v.** After figuring out how are you going to implement the pushbutton functionality in swerolf\_syscon. v those signals must be provided as input to the debounce file and the output of the debounce files needs to be brought further in the hierarchy. If you want to learn more about the debouncing problem, you can refer to the pdf which is attached with this guide.

**Seven Segment Controller:** For this project you also need to edit the functionality of the seven-segment controller where there must be addition of few more bits to it so that it can display more functionality. You need to add the functionality to display the individual segment this changes will be done in swerolf\_syscon.v where the seven segment controller module is defined.

The 8-digit 7-segment display controller used in this course has been custom-designed for RVfpga. It includes two registers, called *Enables\_Reg* and *Digits\_Reg*, that are mapped to addresses 0x80001038 and 0x8000103C respectively (note that these addresses are unused addresses within the address range reserved for the System Controller).

The current 7-segment digit register is only 32 bit and uses 4 bits per segment. A recommend method of changes is to change it to 8 bits per segment and occupying two 32 bit addresses to include more functionality to your controller.

Memory-mapped addresses of SweRVolf SoC peripherals

|  |  |
| --- | --- |
| **System** | **Address** |
| Boot ROM | 0x80000000 - 0x80000FFF |
| **System Controller** | **0x80001000 - 0x8000103F** |
| SPI1 | 0x80001040 - 0x8000107F |
| SPI2 | 0x80001100 - 0x8000113F |
| Timer | 0x80001200 - 0x8000123F |
| GPIO | 0x80001400 - 0x8000143F |
| UART | 0x80002000 - 0x80002FFF |

***Software Development:***

Design and implement two motion indicators for the SimpleBot: The first indicator (Digit4) indicates whether the SimpleBot is stopped, moving forward, moving in reverse or doing a right turn or left turn. The second is a directional indicator, (Digit3-Digit0), which indicates the compass heading of the SimpleBot (0°= North, 90°= East. 180°= South, and 270°= West and every point in between). The compass range is 0° to 359° (e.g. there is no compass heading 360°, the compass wraps around).

***Seven-segment Display:***

The seven-segment display on the Nexys A7 is a multiplexed display with 8 digits (See Section 9.1 of the Nexys A7 Reference Manual for information on how a multiplexed display works). An application can write to the digits of display by writing to 32-bit locations in the RISC address space. This is an example of memory mapped I/O.

The *Enables\_Reg* is an 8-bit signal where each bit determines if the corresponding digit is *ON* (0) or *OFF* (1). The *Digits\_Reg* is a 32-bit signal where each 4-bit group represents the hexadecimal value to show in the corresponding digit. For example, to show *71* on the two right-most digits, the programmer would assign the following values to the registers:

* *Enables\_Reg = 0xFC* (two right-most digits enabled)
* *Digits\_Reg = 0x00000071* (value = 71)

***SimpleBot Functional Specification***

The two independently controlled wheels on the SimpleBot enable the SimpleBot to move forward, backward, turn left or turn right, as shown in Table 3.

**Table 3: Robot Motions**

|  |  |  |
| --- | --- | --- |
| **Left Motor** | **Right Motor** | **Robot motion mode** |
| Stop | Stop | Stop |
| Forward | Stop | Turn Right 1X Speed |
| Stop | Reverse | Turn Right 1X Speed |
| Forward | Reverse | Turn Right 2X Speed |
| Stop | Forward | Turn Left 1X Speed |
| Reverse | Stop | Turn Left 1X Speed |
| Reverse | Forward | Turn Left 2X Speed |
| Forward | Forward | Forward |
| Reverse | Reverse | Reverse |

Note that the robot can turn at two speeds in either direction depending on whether both or only one wheel is moving during the turn.

**PUSHBUTTON MOTOR CONTROL**

The four pushbuttons on the Nexys A7 control the two-wheel motors as specified in Table 4:

|  |  |
| --- | --- |
| **Pushbutton** | **Motor Function** |
| BTN\_LEFT | Left Motor Forward |
| BTN\_UP | Left Motor Reverse |
| BTN\_RIGHT | Right Motor Forward |
| BTN\_DOWN | Right Motor Reverse |

A motor is stopped if neither of the two buttons that control the motor are pressed. If *both* buttons are pressed that motors are stopped. Both motors are stopped if all four buttons are pushed at the same time.

You will implement the functionality of the two motion indicators in a RISC

Assembly Language program that you write and debug. You will use four of the eight digits in the 7-segment display for these indicators.

The digits of the seven-segment display are to be configured as follows:

![Table

Description automatically generated]()

The decimal point to the right of Digit 3 should be lit to separate the Motion Indicator from the Compass. All other decimal points are unassigned. You may, for example, want to use one of the decimal points to blink to show that your program is running (this is up to you).

These functionality of the two indicators is described below:

**MOTION INDICATOR**

This indictor uses a single seven-segment digit to provide an animated display of the robot’s current motion mode.

![A close up of a device

Description automatically generated]()

|  |  |
| --- | --- |
| **Robot motion mode** | **Display Action** |
| Stop | Segment g on steady |
| Turn Right 1X Speed | Chase clockwise 5Hz |
| Turn Right 2X Speed | Chase clockwise 10Hz |
| Turn Left 1X Speed | Chase counterclockwise 5HZ |
| Turn Left 2X Speed | Chase counterclockwise 10HZ |
| Forward | Segment a blinks |
| Reverse | Segment d blinks |

“Chase” means that one lit segment moves around the perimeter of the display digit. Only one segment is lit at a time. The lit segment advances one position at the rate of 5Hz or 10Hz depending on the turning speed. For a clockwise chase, the segments are lit in the following sequence: a, b, c, d, e, f, a, ... For a counterclockwise chase, the sequence is reversed.

**COMPASS**

This indictor uses the three rightmost digits on the seven-segment display indicate the robot’s current heading from 0 to 359 degrees. The compass heading is computed by dead reckoning, that is, by accumulating the robot’s motions over time. When the robot is stopped or moving in a straight line (either forward or reverse), the compass heading does not change. The compass heading only changes when the robot is turning. The rate of compass change will, of course, depend on the turning rate. The compass can be implemented with an up/down counter. Table defines the actions of the compass counter as a function of the motion mode.

|  |  |
| --- | --- |
| **Robot motion mode** | **Compass Action** |
| Stop | Hold |
| Forward | Hold |
| Reverse | Hold |
| Turn Right 1X speed | Increment at 5Hz |
| Turn Right 2X speed | Increment at 10Hz |
| Turn Left 1X Speed | Decrement at 5Hz |
| Turn Left 2X Speed | Decrement at 10Hz |

**SIMPLEBOT PROJECT DESIGN NOTES**

Both the Motion Indicator and Compass modules use the SimpleBot’s motion mode as inputs. The motion mode can be decoded from the pushbutton inputs by reading the value of the buttons at physical address 0x1F80\_0008. The motion indicator (MI) can be implemented with a finite state machine (FSM). The MI output can use special codes that cause each of the individual segments of a digit to be displayed. Refer to Table 2 for the codes needed to drive the individual segments.

The compass can be implemented with a 0-359 BCD (decimal, not hexadecimal) up/down counter. There is, of course, additional control logic not described in detail here; the implementation is left for you to complete. Figure 2 provides the architecture and a simplified flowchart for your application.

![Diagram

Description automatically generated]()

RVfpga

***Write a Theory of Operations document for your implementation***

It is important that you document your design so that we can correctly grade your project. Your *Theory of Operations* should include a short summary of the design and then technical documentation describing your implementation. Make use of the flowcharts and state transition diagrams you created when you did your design to explain your code. Select snippets (short pieces of code, typically less than 20 lines) that you feel deserve further explanation and include them, along with a concise description of how they work. The goal for your design report is to provide insight into how your design works.

While you are at it, make sure your code is clearly documented. Include headers for all of your source code with your name, project, and a short description of what the module or assembly language file does. **Comment your code liberally.** This is especially true with assembly language code which is difficult to follow even if it’s well-written. You should have one useful comment for each line (or every few lines) is appropriate. You will be docked points if you don’t do this!

**Deliverables**

1. When your design is working correctly, organize the deliverable files which would contain Verilog files which you edited and assembly code file and Vivado XPR file along with your *Theory of Operations* and e-submit them to your Project Dropbox on D2L or push them on your GitHub repository. The files should be delivered in a single .zip file while submitting them on d2l.
2. You will also need to create a video showing the implementation of all the functionalities and upload it on d2l.

***Related Documents***

**[1]** *Nexys A7 Reference Manual*, Digilent Inc. (https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual)

**[2]** *A Guide to Debouncing*, Jack Grassle, June 2008 (good reference)