1. Objective

This guide assumes familiarity with the previous lab; the concepts and algorithms previously developed should be used as guidelines for this lab. Refer to Lab 3 guide for prerequisite topics.

In this lab, you will:
- Use LabVIEW to simulate the control of the iRobot Create as it climbs a hill.
- Use LabVIEW Embedded to generate C code that targets the Luminary Micro.
- Program the iRobot Create to navigate and climb a hill.

2. Equipment

- Windows computer with Keil μVision and LabVIEW Embedded installed.
- iRobot Create, DB-25 connector, and charger.
- Luminary Micro LM3S8962 controller, USB cable, and Keil ULINK2 JTAG emulator.
- ADXL-322 accelerometer.
- BlueSMiRF Bluetooth serial modem (optional).

3. Overview

In this lab, you will first implement your controller and state machine design from the previous lab into a desktop simulator. Once your simulator is working properly, these VIs will then be translated, compiled, and programmed to the Luminary Micro.

The final goal of this lab is to program the iRobot to autonomously climb a hill. Your robot must determine the correct orientation to drive towards the top of the hill; avoid driving off the hill; and navigate around obstacles that may be along the way.

The lab has been divided into two parts, completed over two weeks:
1. Desktop simulation of the controller and iRobot Create.
2. Hill climbing iRobot.
4. Introduction – LabVIEW Embedded

National Instruments LabVIEW Embedded extends the LabVIEW programming environment to embedded platforms. This adds an additional level of abstraction to software development: code generation. Working alongside Keil µVision, your LabVIEW Virtual Instruments (VIs) are translated into C code, which is compiled, linked, and downloaded to the Luminary.

Code generation allows developers to more closely follow model-based design; however, it remains crucial to consider size and runtime of the final application. The addition of array manipulation VIs or floating point operations may significantly increase the number of machine instructions generated. Keep in mind the design methodology introduced in lecture.

5. Development Environment

1. Follow the step-by-step instructions in [5] beginning on page 14 to build an embedded application in LabVIEW. Please note that the ULINK2 JTAG emulators are not configured for ARM debugging, so debugging must not be enabled in your build specification. The instructions outline a simple application that uses two numeric controls on the front panel to toggle the status LED on the Luminary; because debugging is not available, you will not be able to change these controls dynamically from your desktop. Instead, change the values of the controls, set these values as the defaults for your VI, then build and flash the Luminary. The status light should change to the appropriate state after pressing the reset button.

2. Example programs should be installed in your computer (C:\Program Files\National Instruments\LabVIEW 8.6\examples\lvemb\ARM\LM3S8962). Try building and downloading several programs. A good starting point is the “OLED” project. Remember to disable debugging before running.

Note: Please do not modify the projects in the examples folder, as they are used by other labs. You may copy an example to your user (U:) drive if you wish to change it.
6. PART 1 – Hill Climb Simulation (week 1)

1. Download the template LabVIEW project [0] that has been created for this lab to your user (U:) drive. Open the project and review the code. Make sure you understand how the code drives the iRobot and reads sensor packets.

   You will notice that the VI hierarchy exists for both the desktop and embedded target. To run the desktop simulation, open HillClimb.vi from the “My Computer” target.

2. Run HillClimb.vi and observe its behavior. An iRobot simulator generates sensor packets every 15ms based on the values set on the robot sensors control on the front panel. You should also see state variables update based on these sensor packets. Changing the delay will affect the simulation speed. Accelerometer controls simulate the ADXL-322 [6].

3. In the iRobot Simulated Sensor control, enter values for distance and angle. You should see the control state accumulate these values; this simulates the distance and angle traveled by the iRobot. What happens when you press the advance button?

4. Try several values for the Vstep control and observe how the simulation changes. The value is fed into a speed stepper that smoothly ramps up speed instead of instantaneous changes. You may find this VI useful when implementing your controller.

5. There are a number of conditional disable diagrams throughout the code. Only one subdiagram of a conditional disable diagram will execute; which one depends on if you are using the desktop simulation or if you are targeting an embedded platform. Be sure to know which diagram executes in which mode.

6. This lab can be completed by modifying only the VIs in the root of the project hierarchy, though you are free to modify other VIs.

7. Build on the template project. Extend the wheel controller, state machine, and state evolution VIs to reflect your solution to the previous lab. Add additional states to the Control State Variables type definition. Use the iRobot sensors control to simulate bumps, cliffs, and wheel drops. The advance button should stop the program. Your control state and wheel speeds shown on the front panel should simulate the behavior of the iRobot.

   **Checkoff:** Demonstrate that your robot simulation avoids obstacles and adjusts wheel speeds according to your controller specifications.
7. **PART 2 – Targeting the Luminary (week 2)**

1. Open HillClimb.vi from within the EK-LM3S8962 hierarchy in the LabVIEW project. To download and flash to the Luminary, right click on the Release build specification and select “run”. Compiling and flashing will take a few moments.

2. If you have added floating point code, or made calls to VIs that are not supported by the Luminary, the compile process may generate errors. Do not attempt to debug the generated C code; even if you resolve the error, the code will be overwritten the next time you generate. Use the diagram disable structure to fence in the bug in LabVIEW.

3. Your solution **must not** make use of the inline C node

4. Build, download, and run your application on the Luminary. If you run into problems, don’t hesitate to return to the simulator to debug your code.

**Checkoff:** Demonstrate your fully autonomous robot, which climbs to the top of a hill, avoids falling off of a cliff, and navigates around obstacles encountered along the way.
8. LAB REPORT
Your lab report should include:

1. Descriptions of any design decisions that were made.
2. Brief outline of the structure of your code.
3. Printouts of any VIs that were modified from the project template.
4. Did you make any logical or structural changes to your hill climb solution from the previous lab? What prompted these changes?
5. Lists of bugs you encountered or unexpected challenges you overcame.
6. What worked well? What would you change if you were to start this lab from scratch?
9. REFERENCES

Your primary reference will be Getting Started with LabVIEW Embedded Module for ARM Microcontrollers [1].

[0] Lab 4 Files (LabVIEW project template)
http://chess.eecs.berkeley.edu/eecs149/sp09/lab/Lab4-files.zip

http://chess.eecs.berkeley.edu/eecs149/sp09/docs/CreateManual_Final.pdf

[2] iRobot Create Open Interface (OI)
http://chess.eecs.berkeley.edu/eecs149/sp09/docs/CreateOpenInterface_v2.pdf

http://chess.eecs.berkeley.edu/eecs149/sp09/docs/EK-LM3S8962_EvalBoard_UM.pdf

[4] Luminary Micro Datasheet

[5] Getting Started with the LabVIEW Embedded Module for ARM Microcontrollers
http://chess.eecs.berkeley.edu/eecs149/sp09/docs/374931b.pdf

[6] ADXL-322 Datasheet
http://chess.eecs.berkeley.edu/eecs149/sp09/docs/ADXL322.pdf

10. ADDITIONAL RESOURCES

[7] Introduction to ARM Microcontrollers
http://zone.ni.com/wv/app/doc/p/id/wv-583

[8] ARM Cortex-M3 Processor
http://chess.eecs.berkeley.edu/eecs149/sp09/docs/CortexM3_TRM.pdf