

ENG H192: Hands on Labs

Lab 7: Controlling a Stoplight

Engineering Disciplines Explored: Electrical, Computer Science, Civil

Introduction

Purpose

The purpose of this lab is to create a program for the Handy Board to control a simulated stoplight. Additionally, this will be implemented in LabVIEW. Another program will be implemented in LabVIEW also.

Basic Principles

This lab write-up will cover the following basic principles of programming in LabVIEW:

- Linearity
- Local variables
- Timing and Timed Loops
- Event Structures

Lab Experience

The lab experience will encompass:

1. Using the Handy Board and Interactive C,
2. Writing and converting source files from C to IC, and Compiling source files.
3. Programming in LabVIEW

Fundamentals

Background

All quarter we have been programming in C, and we have used LabVIEW for data acquisition purposes, only. Next quarter the robot will be programmed in Interactive C (IC). We will look at programming in LabVIEW and IC, compared to C. The programs you write in LabVIEW will give the same results as similar C-programs we have written, but with slightly different methods, and a graphical user interface (GUI). We will be programming assignment A08 in LabVIEW, along with the stop-light lab.

Linearity in Programming:



LabVIEW executes code in parallel. Recall wiring up Boolean expressions for the digital electronics lab. You could put a number of Boolean expressions into one while loop, and they would all execute in parallel. In LabVIEW, there are a number of ways to force a program to operate linearly. Most VIs come with “error in” and “error out” terminals (the pink ones, usually at the bottom). Wiring these error terminals in series will force the program to only execute a VI once it gets an error in from the previous VI. For the stop-light lab, this would require creating a proprietary VI for your operations (similar to the functions you wrote in C). You don’t have to do this, though you can for extra points.


variable, it is also input to the structure which it represents (in this case the LED indicators).

You can create multiple local variables for any of your indicators. The indicator value will be reassigned as each part of your program is executed. You will need to create a number of local variables for the stop light lab.

Timing and Timed Loops

Figure 1 also shows a timed loop. This works like a while loop, but takes timing information too. It is found under “Programming >> Structures >> Timed Structures >> Timed Loop.” Right clicking on the timing loop and choosing “Configure Input Node” will bring up some options that will allow you to adjust how the loop runs. (Specifically, you might need to adjust the “period” value.)

Additionally, the frames in the flat sequence have wait specifications () in them. The wait VI waits a certain number of milliseconds before moving on. It can be found under “Programming>> Timing>> Wait.” In this case, the amount of wait time (in ms) for each frame of the flat-frame sequence is specified by wiring the “wait” to a numeric constant (), found under “Programming >> Numeric >> Numeric Constant.”

In the bottom left-hand corner of the timed loop is the iteration count variable (). This counts the number of times the loop has executed since the program was started. In Figure 1, this counter is being used to set the stop conditions for the timed loop (it will stop executing after 3 iterations).

Event Structures

In a sense, event structures are used to get user input. An event structure waits for an action, then executes code based on that action. In A08, you wrote a program that waited for a user to type in a shape. In your LabView program, the shapes will be buttons. The program waits until an action, such as “mouse down” on button, occurs, and then executes the code that matches that case. Your event cases will be performing math on the input dimension, which the user will input in a numeric control on the Front Panel.

The event structure is found under “Programming >> Structures >> Event structure.” Right-clicking on the event structure allows you to edit the current event, or create a new event. The events are mutually exclusive. For your purposes, you do not need to worry about any of the options listed in the structure (e.g. Source, Type, Time).

Figure 2 shows the event structure for the “quit” case. Notice that the numerical indicator “Area” does not have a value for this case, but the Boolean does. In your event cases, the area MUST be wired up for each shape case. If you wish, you can wire the Boolean to “False” to assure that the program won’t stop. Additionally, you can clear the area in the quit case by wiring a constant to it. Notice all this happens within a while loop.

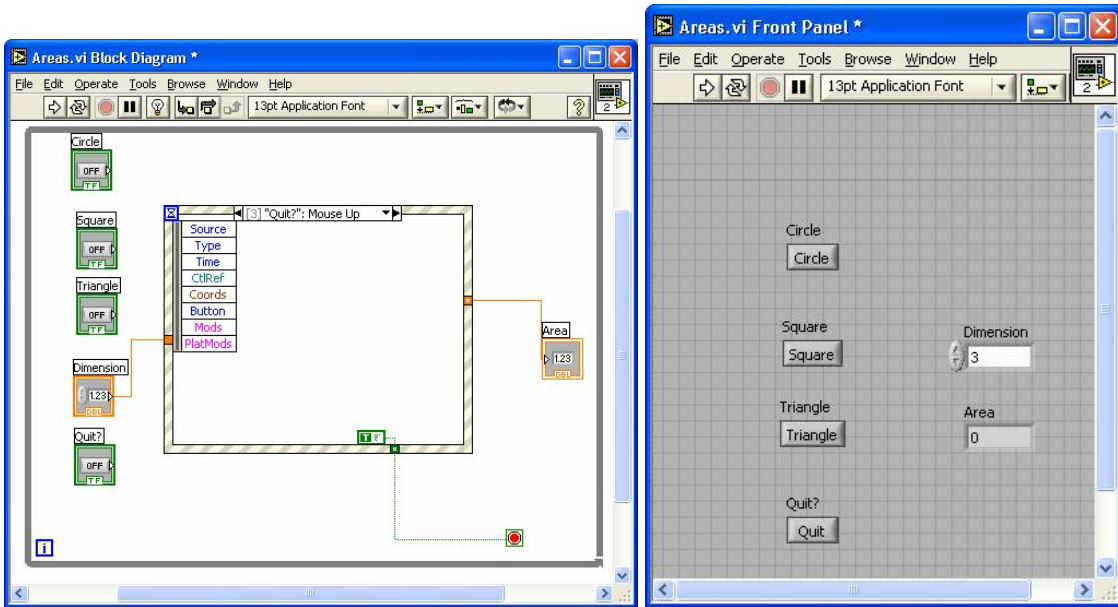


Figure 2. Event Structure and Front Panel for A08

Lab Experience

Using the C program you wrote for Hand Out #1, you are to create a program for the Handy Board to control a simulated stoplight. You will need to edit your program to make the necessary changes so that it will compile with the Interactive C compiler (IC) for the Handy Board. You may choose to either use a Windows editor (Notepad, Wordpad, or other) or you may use the editor built into IC (suggested). You will need to connect the stoplight wires to the motor ports on the Handy Board. Refer to the diagram of the Handy Board for location of motor ports. Recall that one motor port will be used to control one of the colors on the stoplight, and the polarity of the motor output will control the which of the two lights (i.e., which direction) will be turned on. Thus, three motor ports will be used, one for the "red", one for the "yellow", and one for the "green" lights.

Procedure

Each team member should attempt the following procedure:

1. Start IC and "Download P-Code" to Handy Board. (Click on "Board" and select "Reload Pcode", follow directions, choose "Handy_Board_1.2.icd" if given choice).
2. Open your source file in IC. (Click on "File" and select "Open").
3. Edit source file as may be necessary.
4. Download (compile) source file to Handy Board. (Click on "Load" and select "Download Window". If not error free, go to step 3).
5. Reset (Turn off, then back on) the Handy Board to start program. If stoplight does not perform as desired, check hookup and/or program. Return to step 3 as necessary.

If stoplight performs as desired, have the Lab Instructor or TA verify operation and initial your printed copy of the Handy Board source code. (You will have to open your source code in a Windows editor to print it.)

Post-Lab Requirements : Group Memos (in pairs)

Lab Memo style will be used. Please refer to the FEH website for guidelines.

No separate results section is required for this lab

This memo should be 1-2 pages.

Introduction

Discussion

- Brief discussion (make a list) of steps required to transport and convert UNIX source code to Handy Board source.
- Brief discussion of changes you expected to make to your source code.
- Brief discussion of changes you actually had to make to your source code.
- If you were not able to get your own source code converted and running, explain what the difficulties were.
- Compare the IC, C and LabVIEW versions of your stop light program
- What were the main differences? Similarities?
- Which one made the most sense to you as a programmer? As a user? Why?
- Compare A08 in C and LabVIEW
- What were the main differences? Similarities?
- Which one made the most sense to you as a programmer? As a user? Why?
- Why did you have to use an event structure for A08?

Conclusions:

- What are the benefits to programming in C? in LabVIEW?

Attachments:

- Copy of source code that runs on UNIX system.
- Printed copy of messages produced when program was run on UNIX system.
- Copy of Handy Board source code as initialed by Instructor or TA. (If you are not able to get your own version running on the Handy Board, you must include a copy of your team's working program).
- Include block diagrams for your 2 LabVIEW programs
- You need not include A08, but if you have your C version, feel free to do so.

NOTES:

-All figures from the FEH site MUST be appropriately cited.

-All figures and tables must have numbers and titles

-Memos have attachments not appendices (similar to an email message).