

March 01, 2008

Lab 7: Computer Control of Digital Output Using C++

Bernard M. Gordon Center for Subsurface Sensing and Imaging Systems (Gordon-CenSSIS)

Recommended Citation

Bernard M. Gordon Center for Subsurface Sensing and Imaging Systems (Gordon-CenSSIS), "Lab 7: Computer Control of Digital Output Using C++" (2008). *High Tech Tools & Toys Labwork*. Paper 7. <http://hdl.handle.net/2047/d20003900>

This work is available open access, hosted by Northeastern University.

GEU111 – High-Tech Tools and Toys Lab

Lab 7: Computer Control of Digital Output Using C++

Introduction:

In this lab, you will learn how to write a C++ computer program to control the digital output of a data acquisition card in a computer backplane. We will use the digital outputs to control a stepper motor to rotate through fixed angles and change directions.

Pre-lab Preparation:

1. For Part I of the lab you will need to learn about programming in C++ with Microsoft Visual C++ in the High-Tech Tools and Toys Lab. We will be using some special libraries with control programs for the National Instruments 6024E data acquisition card. This document will explain how to link to these libraries.
2. Print out and bring to lab with you the documentation for the National Instruments control functions that we will be using as explained in Part II.

Part I: Programming a Computer for Digital Control

This part of the lab will be done in a laboratory with computers that have National Instruments PCI-6024E multifunction input/output boards installed in the backplane. We will be using the digital output function of these boards and writing programs in C++ to allow us to control a stepper motor.

The National Instruments PCI-6024E boards have analog input, analog output, digital input, digital output, and clock/timer functions. A description of the boards and a link to the data sheet and specs can be found on the National Instruments web site at:

<http://sine.ni.com/apps/we/nioc.vp?cid=10969&lang=US>

For the base lab we will only be using the digital output functions of the board, and we will be using two C++ functions in the National Instruments Data Acquisition (NIDAQ) libraries to control the outputs:

DIG_Prt_Config sets the digital ports for input or output use

DIG_Out_Prt loads a bit pattern into the digital port to set the 8 digital outputs either high or low

To do the Extension Problem you will need the following NIDAQ functions to control the Analog Input/Output ports:

AI_Configure

AI_VRead

Documentation for these functions is on the Blackboard site under “Software Documentation”.
Print out the documentation for these functions and bring it to lab with you.

A. Linking to External Libraries and Header Functions

In addition to the source program that you write, all C++ programs include header files. The source program is a text file with a name such as `myprogram.cpp` that typically begins with a line such as `#include <stdio.h>` (we have been using `<iostream>` instead) and has an `int main(void)` or `void main(void)` function in it. The header files are plain text files such as `stdio.h` containing the function prototypes of any function that you call, defined constants, and other preprocessor instructions. After the your source program and included header files are *compiled*, the resulting *object file*, is *linked* to library files that contain the executable code for standard input/output (such as `printf`), math (such as `sin` or `abs`), other functions that you may have used, and instructions to the processor. The resulting *executable* file (for example, `myprogram.exe`) can be run by the operating system.

In the vocabulary of the Microsoft Visual C++ Studio programming environment that we will be using, a complete program, including all the header and linked library files is a “Project.” Your Project will include Source files (including the program you write), Header files (such as `stdio.h`), Library files, and perhaps other Resource files.

A major difference between this lab and other C++ applications that you may have written is that you will have to link your program to the external header files and libraries provided by National Instruments (NI) to control their hardware. All the C++ programs that you have written needed to be linked to header and library files to run, but in many cases this may have been done transparently so that you didn’t have to worry about where the files were. To use the NI software, we don’t have this luxury.

To run the digital and analog control functions above you will need to make sure that that your project includes the following header file:
`nidaq.h`

Your C++ source file must `#include` the file `nidaq.h` by adding the line `#include <nidaq.h>` before your main program. If you haven’t created a path to the NI-DAQ\Include folder you will have to use the entire path to `nidaq.h`:

```
#include <C:\Program Files\National Instruments\NI-DAQ\Include\nidaq.h>
```

You will not need any of the other header files in the NI-DAQ\Include folder to run the digital output and analog input NI-DAQ routines in this lab.

In addition, you will need to add the following library into your project:
`nidaq32.lib`

See the handout on “Creating a C++ Program in Microsoft Visual Studio” for detailed instructions on adding a library file to a project using the Project >> Add Existing Item menu.

On the computers in the High-Tech Tools and Toys Lab the library files are stored in the folder:
`C:\Program Files\National Instruments\NI-DAQ\Lib`

Directions for adding these header files and libraries to your Project are included in the document “Creating a C++ Program in Microsoft Visual Studio” which is on the course Blackboard site. Print out this file and bring it with you to lab as well.

Prelab Task 1: We will need to create a function to cause a time delay in some of the steps below. (We want to write a routine to make the program run slower!) Write and debug a function `void time_delay(int n)` that runs two nested loops `n` times. You don't have to put anything in the outer loop except the inner loop and the inner loop can be empty. Write a main program to read in an integer `n` and call `time_delay(n)`. Continue to read `n` and run `time_delay` until the user enters a negative number for `n`. See how large `n` should be on your computer to get a time delay of 1 second, 2 seconds, and 3 seconds. If you are using an empty loop you should probably start with `n=5000` and increase it in steps of 5000 until it takes a second or two to finish the loop and prompt for another input. Turn in a printout of the program and bring it to class on your memory stick.

Prelab Task 2: Write a C/C++ function `void Lab7_D2_xyz(void)` (where "xyz" are your initials) which uses the NI-DAQ functions `DIG_Prt_Config` and `DIG_Out_Prt` to configure the DIO port for output and write the values from 0 to 16 into the digital output port, waiting 1 second between each write, as in Experiment Part D2 below. Make sure you know what parameters you need to pass to each NI-DAQ program. You will not be able to link or debug this program at home because you do not have the NI-DAQ programs on your computer, but bring your draft program on a memory stick to class and turn in a printout of the file.

B. Getting Started

Experiment B1: To familiarize yourself with the Microsoft Visual Studio interface, open a new Windows Console Project called `Lab7_xyz`, where "xyz" are your initials. Add to the project a C++ source file called `Lab7_xyz.cpp`. Include `<iostream>` and write a main program that writes to your terminal the words, "GEU111 HTT&TL - Lab 6." Compile, link, and run this program to make sure that you know how to get a program to run in the MS Visual Studio environment in the HTT&TL. You will be doing a number of different tasks below in Sections C through F. Write each of these tasks in their own function and change your main program to call the appropriate function for each section.

C. Writing a Variable to a Register

In the "High-Tech Tools and Toys Lab" you will find a breakout box from the NI PCI-6024E card that had four labeled digital outputs: DIO0, DIO1, DIO2, and DIO3. Three of these outputs are BNC connectors and one is a banana output. `DIG_Out_Prt(Device, Port, Pattern)` writes the value of the variable/constant "Pattern" into the Digital Output (DIO) port. The DIO port is one byte – eight digital lines ("bits") – as illustrated below. In the lab we have brought the lowest four bits of the DIO port (DIO0-3) to the breakout box front panel.

Digital Output Port (Port 0)



Experiment C1: Take the output from DIO0 and connect it to the Digital Multimeter in the lab. (Make sure that you use the correct set of inputs to the multimeter.) Write a function to a) call the `DIG_Prt_Config` program to configure the digital input/output port for output, and then b) call `DIG_Out_Prt` to write the value "1" into the port. The "Device Number" of the PCI-

6024E card is “1”, the digital output register is Port 0, the Mode is 0 (no handshaking), and the Direction is 1 (output). The “pattern” is the value that you are writing to the register (either 0 or 1 here – you will be writing other integers in the following sections). You will have to include `nidaq.h` in your source code by adding the line:

```
#include "nidaq.h"
```

before your main function. You will also have to add to your Project the NI library `nidaq32.lib` as explained in the last section.

Compile and link your main program and function (hope for no errors!) and run it. What happens to the output voltage of DIO0? Change your function to write the value “0” into the port, recompile your project, and run it. What do you observe?

Experiment C2: Modify your function to use your `time_delay` function to add a 2 second delay between writing a “1” and writing a “0”. (You may have to adjust the value of `n` if the computers in the HTT&T Lab are faster or slower than the computer you wrote the `time_delay` program on.) Run your program again. Write a `for` loop to alternately write “0” and “1” ten times and wait 2 second after each write. Does the output of DIO0 measured on the multimeter do what you expect?

D. Using Light Emitting Diodes (LED’s) in an Electronic Proto-Board

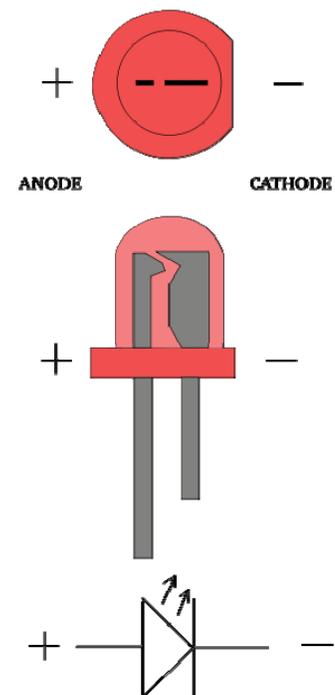
A light emitting diode (LED) is a semiconductor device that emits light when current is directed through it. A typical red LED as shown in the figure, along with its circuit symbol, is made from an alloy of the semiconductors Gallium Arsenide and Gallium Phosphide (GaAsP).

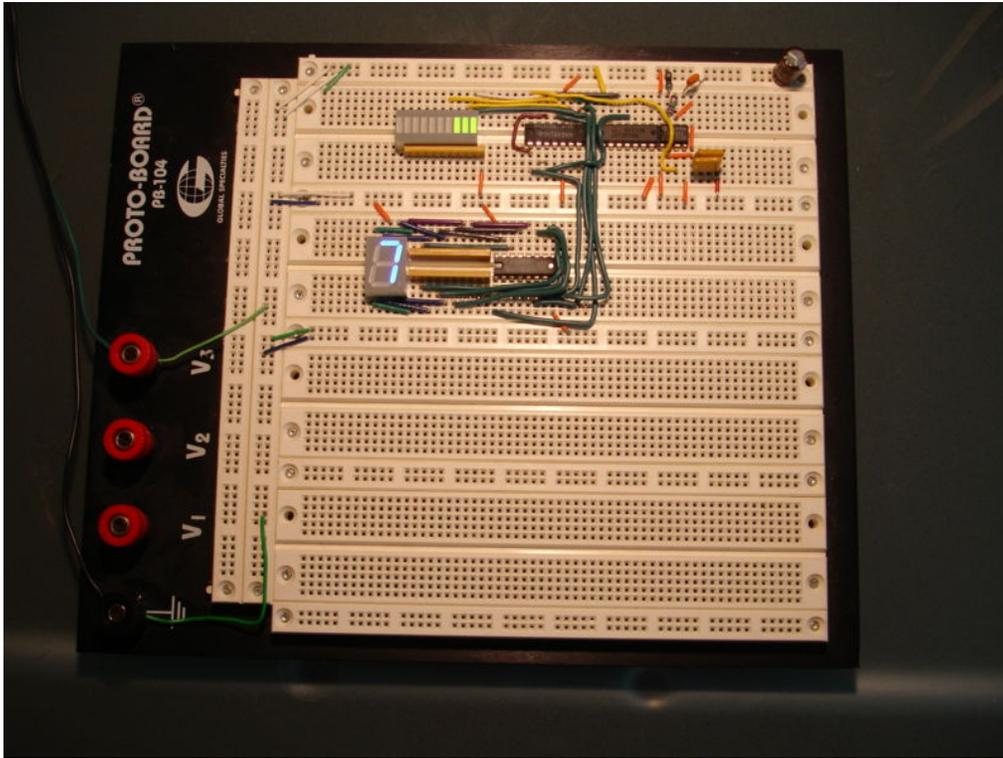
The LED will light up if it is *forward biased* if current is passed through it from the positive terminal (identified by the longer lead) to the negative terminal. If too much current is passed through the LED ($> \sim 60$ mA) it will burn out. For this reason if a LED is used with a battery or other voltage source it is always used with a resistor in series with it to limit the current. The size of the resistor is not critical – with a standard 5V logic level voltage 220 ohms is sufficient ($I=V/R=5V/220\Omega=9$ mA).

To make it easier to examine the output of a memory location, you are going to hook up LED’s to the lowest four bits of the Digital Output Port. To do this, you will use a Proto-Board electronic breadboard as illustrated below.

The Proto-Board is covered with dimples that electronic components and wires can be inserted into to make connections without soldering. The dimples are connected together by internal connections underneath the board in the pattern illustrated below.

The dual rows of five horizontal dimples that extend from one side of the board to another are internally connected horizontally, but the rows are not connected to each other. The densely dimpled sections are connected vertically in groups of five, but are not connected across the indentation in the center.

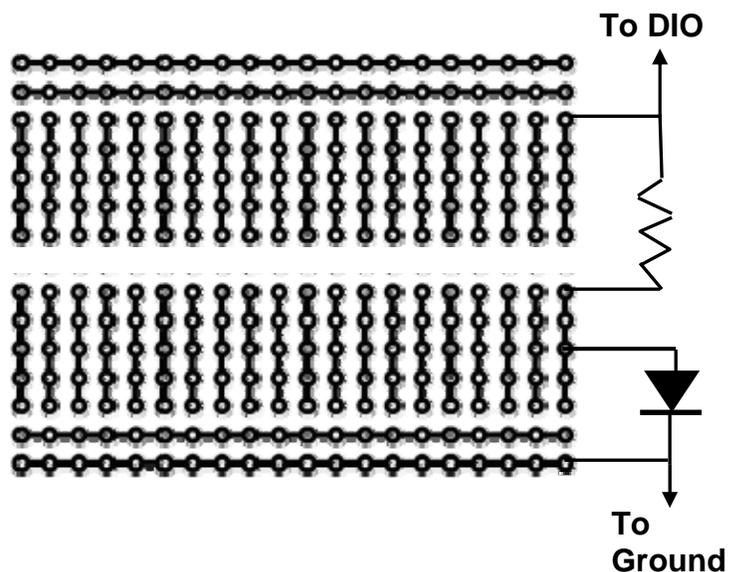




You can use one of the horizontal rows to connect to one of the black ground (0 volt) outputs on the A/D board breakout box. All the grounds of the outputs are the same, so you only have to have one connection. You can run a banana plug from the breakout box to the ground banana plug on the Proto-Board and then run a wire from the plug into one of the horizontal rows of dimples.

The negative leg of all four LED's can then go into the same grounded horizontal row. The other side of the LED can go into one of the five vertically connected dimples in the center of the figure. One leg of the 220Ω resistor can go into the same column and with its other end in a disconnected vertical group. (Resistors are unpolarized so it doesn't matter which end goes where.)

Use a different column for the next LED and its series resistor – you can use the same row for the ground connection. Once you connect up four LED's you should check your connection by connecting the resistors one at a time to the power supply set for +5 volts and the ground line to the power supply ground. If you have made your connections correctly one and only one LED will light up – the one in series with the resistor you have connected – and none of the LED's will burn out. Once you are sure of your connections, you can connect the resistor to the DIO connections and the ground line to the ground.



Experiment D1: Using your protoboard, wire up four LED's in series with 220-ohm resistors to ground. Connect one LED to each of the four digital output ports DIO0, DIO1, DIO2, and DIO3 in order. Write a new function to write the value "2" into the digital output register. Comment out the previous function call (for Part C) in your main program and call the function you just wrote. Recompile and run your project and describe what you observe from the LED's. Now change your function to write "3" into the digital output register. Change the function once again to write "5" into the digital output register. How do you explain your output?

Experiment D2: Write a new function with a `for` loop that writes the values: 0, 1, 2, 3, 4 and so on, up to 16, to the register, waiting 1 seconds between each write. Change the main program to call this function. (You can delete or "comment out" any previous function calls in your main file.) Run your project and get the instructor or TA to observe and sign off on your lab book.

E. Bitwise AND and OR

You may recall that the syntax for a logical AND operation between two variables in C++ is "&&" and a logical OR is "|". We are going to examine the *bitwise* AND and OR operations that are denoted by "&" and "|" .

Experiment E1: Write a function that prompts and read in two positive integer variables `i` and `j`. Define `k1=i & j` and `k2=i | j` . Write the value of `i`, then `j`, `k1` and `k2` sequentially into the digital output register with a 1 second wait between them. At the same time display `i`, `j`, `k1`, and `k2` on your terminal display. Record what you observe on the LED's and on the screen for each case below. Explain your results. What do the bitwise operators do?

- a. `i=8, j=15, k1=i & j, k2=i | j`
- b. `i=12, j=15, k1=i & j, k2=i | j`
- c. `i=12, j=9, k1=i & j, k2=i | j`
- d. `i=12, j=3, k1=i & j, k2=i | j`
- e. `i=1, j=12, k1=i & j, k2=i | j`
- f. `i=5, j=12, k1=i & j, k2=i | j`

F. Controlling a Stepper Motor from C++

For this part of the experiment you will use the *stepper motor* connected to the calibrated dial and the *stepper motor controller* in the aluminum box. The controller has two inputs, labeled "Clock" and "Direction." Connect up the Clock input to DIO0 and the Direction input to DIO1. The controller has circuitry to cause the stepper motor to make one step counterclockwise if it receives a single rising pulse in Clock and the Direction input is held at 5V. It will take one step clockwise if there is a rising pulse on Clock and the Direction input is held at 0V.

Experiment F1: Using a `for` loop, write a new function that it will cause the stepper motor to take 50 steps in a counterclockwise direction. For each pulse on the Clock input, you will need to cause DIO0 to go high, wait for about 0.2 seconds, and then go low again. All this time the Direction input will have to be high. How many degrees does the stepper motor move per step?

Experiment F2: Write a function using DIO0, DIO1, DIO2, and DIO3 that causes the following things to happen in order:

- a. A red LED lights for two seconds
- b. While the red LED stays on, the stepper motor turns one complete revolution (360°) clockwise.
- c. The red LED turns off and a green LED turns on for four second
- d. While the green LED stays on, the stepper motor rotates one complete revolution counter clockwise
- e. Both LED's turn off.

When this program works, have the instructor or TA observe it operating and sign off on a printout of your function.

Experiment F3: By varying the delay time in the Step input, experiment to see how quickly you can make the stepper motor complete the previous task. Stepper motors all have some maximum step rate, depending on the construction of the stepper motor, the circuitry of the controller (particularly the amount of current provided to the motor), and the torque applied by the motor (how large the rotational inertia of the object to be rotated is). Above this rate, the stepper motor will begin to skip steps. What would you estimate is the highest step rate that your motor can handle without skipping steps?

When you turn in your lab write up, include your main program and the functions that you wrote to do the tasks in Sections C-F.

Extension/Challenge Problems

1. You may have noticed that your `time_delay` program leaves something to be desired. Among other problems, the time delay depends on the speed of the computer you run it on. Write a new `time_delay` program using the C++ program `time` (contained in the `ctime` library) that returns the time in seconds since 0:00:00 on January 1, 1970, (“UNIX time”). You will have to include the `ctime` library, define two new variables of type “`time_t`”, get the current time by calling `time` with an argument which is a pointer to `time_now` as below.

```
#include <ctime>
```

```
time_t set_time, time_now;
time(&time_now);
```

This sets the value of `time_now` to the current UNIX time. You can then assign `set_time` to `time_now + your desired time delay` (in seconds – it can be a decimal number) and do a loop until `time_now` exceeds `set_time`.

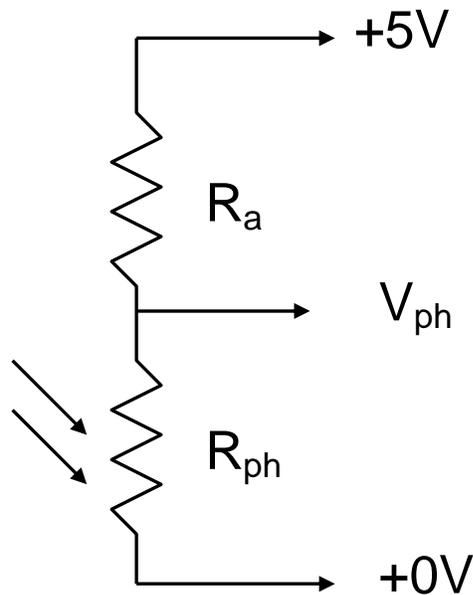
2. Read in the NIDAQ Reference manual (<http://www.ni.com/pdf/manuals/321645b.pdf>) about the Analog Input functions `AI_Configure` and `AI_VRead()`. Note that the PCI-6024E boards that we have in the HTT&TL computers are 12-bit “E-series” boards.

Use the Analog Input functions to do the two tasks below:

a. Connect the AI0 and AI1 channels to the two photocells on the stepper motor dial face and use `AI_VRead()` to read the voltage of the photocells. Use this to write a program in C++ that, starting from a random position, (i) moves clockwise to the first photocell, (ii) lights up a red LED, (iii) oscillates between the two photocells 3 times, (iv) lights up a green LED, (v) rotates to the 180° position on the dial, (vi) turns off both LED's, prints "Program over" to the console and ends.

b. Write a program that calculates the number of degrees per step by prompting the user for the initial angle, takes 1000 steps clockwise keeping track of the number of times that the flag passes the photocell at 0° by looking for a step in the photocell voltage, prompts the user for the final angle, and then divides the total angle covered by 1000.

Note: The photosensors on the stepper motor dial are *photoresistors* that change their resistance, not *photodiodes* that produce a voltage when they are illuminated. To get a voltage out you will need a power source to produce a voltage based the resistance of the photoresistor. The circuit diagram for this is below if you are curious, but all you really need to know is that the red banana plug from the stepper motor dial connects to the 5V output on the breakout box, the black plug goes into ground (0V), and the blue and white plugs measure the voltage across Sensor 1 and Sensor 2. They will go into the Analog Input plugs, AI0 and AI1.



College of Engineering, Northeastern University.
Last updated: 3/6/08.(S. W. McKnight)