DAQ 1: Introduction to Data Acquisition: Digital I/O

1.1 Introduction

In order to expand the usefulness of computational methods to real-world devices and show some of the ways MATLAB might be used towards solving aspects of the Grand Challenges, it is important to learn how MATLAB can generate electrical signals. In the DAQ laboratories, you will use MATLAB to analyze and control a series of devices. To accomplish this, we will first start with the basics of controlling digital outputs using MATLAB and the National Instruments data acquisition cards. We will either be setting an electrical potential - or voltage - on up to four digital output lines. Because of the nature of electricity and the complex and delicate nature of the data acquisition cards and the human body, there are several safety considerations for lab.

1.2 Safety Notice

Running a safe lab is the top priority for this exercise. Failure to abide by these rules will result in your being removed from the lab and receiving a zero on the assignment. Intentionally creating a dangerous environment for yourself or others will also result in judicial proceedings. The rules and their reasons are as follows:

(1) *No food or drink in the lab. Period.* This week, this rule must be strictly maintained. Because of the conductive nature of food and drink, the possibility of damaging the DAQ cards, and the expense of those DAQ cards, no food or drink is permitted in lab this week.

(2) *Remove jewelry on or near your hands.* Rings, bracelets, and watches are generally good conductors, and those generally cause a problem with electricity should any energized stray wires come into contact with the metal.

(3) *Do not put anything unnecessary on the lab bench.* This includes bookbags, calculators, etc. The only things on the lab bench should be the equipment you were provided, the lab handout, and a pencil or pen. The fewer things on the bench the easier to keep track of them.

(4) *Wire things properly.* Make especially sure that all wires connected to the CB-68LP are in the correct lines. Failure to do so could send a high-power signal to a low-power lead and ruin the DAQ board. The digital I/O leads are especially susceptible to voltage overloads.

(5) *In case of a big emergency, get Dr. G.* If something starts to smoke or spark or in any other way act unnatural, call Dr. G over and back away from the lab bench. Do not attempt to "fix" any electrical problems. When in doubt, shout, and get out (of the way). Lab stations, while expensive and precious commodities over which you should exercise diligent stewardship, are imminently replaceable. Until cloning is perfected, you are not.

1.3 Resources

The additional resources required for this assignment include:

- Books: None
- Pratt Pundit Pages: `EGR_53/DAQ_1` (linked on `EGR 53` page), `MATLAB:CB-68LP Pinout`, `Resistor Color Codes`
- Lab Manual Appendices: None

1.4 Getting Started

1. Each lab group should have one person sit at the PC that has the patch of tape (blue, green, or red) on it. This is **not** the same as the strip with the computer's name on it - it will literally be a small patch of tape all by itself.
2. Set the machine to "Log on to" `...(this computer)` and use the "User name" `mrglocal` and the "Password" `p1p2dell` (that is p-ONE-p-TWO-d-e-l-l). The other lab partner can log into the other computer either using the `mrglocal` account or any other valid NET ID (as log as the "Log on to" is set to `Kerberos`). You do *not* need to run X-Win or PuTTY at all on either computer.
3. Start MATLAB on the PC with the tape on it. The first time MATLAB is run, it may take some time to start up.
4. Once MATLAB starts, make sure the `Current Directory` listed at the top of the MATLAB window is

   ```
   C:\Documents and Settings\labuser\My Documents\MATLAB
   ```
5. In the `Current Directory` window on the left, select and delete any files in the `MATLAB` directory.

1.5 Retrieve Necessary Parts

For this week, you will need to retrieve some parts and tools from the front shelf of the room. This includes:

- Four long red wires and one long black wire from the buckets
- A plastic parts box containing four resistors (should all have the same markings), four LEDs (two red, one green, one yellow), a screwdriver, and a breadboard

1.6 Equipment Description

The DAQ lab this week requires several different pieces of equipment. Note that not every piece will be used with every section of the lab.

- *MATLAB on the PC:* This week, you and your partner will actually be using MATLAB on the PC in the lab rather than connecting remotely to a machine in Teer. Each lab bench has two computers on it; only one of them will be used this week. Figure out which of the two computers has the data acquisition equipment attached by first locating the green card between the two computers then following the ribbon cable. Whichever machine has the ribbon cable attached is the machine you need to use. One of the partners should log in to the PC, but you do not need to connect to any Teer or Hudson machines.
- *DAQ (Data Acquisition) Board:* This allows the computer and MATLAB to interact with real-world signals by either measuring or generating voltages (electrical potentials). The board itself is installed in the computer chassis, so there is also a connection block connected via a ribbon cable. All electrical connections will be made on the connection block rather than the DAQ board itself. The pinouts for the CB-68LP connection block are given on the Pratt Pundit page `MATLAB:CB-68LP Pinout`.
- *Breadboard:* This piece of equipment allows multiple wires to share the same voltage value. This is useful for quickly creating an electric circuit and testing it without having to solder anything. The breadboard is divided into 48 different groupings. First, all sockets in the top row, marked **X**, are connected to each other. Also, all sockets in the bottom row, marked **Y**, are connected to each other. These are generally used for power and ground (reference voltage), respectively, since there may be many connections to each of these signals.

  The other 46 groupings are made up of half-columns of five sockets each. For example, column 10 rows A-E are electrically connected to each other. If one wire is plugged into A10 and another is plugged into D10, it is as if those two wires are connected to each other. If one wire has one end plugged into

A10 and another plugged into H20, then it is as if B10, C10, D10, E10, F20, G20, I20, and J20 are all connected. Breadboards are excellent for prototyping circuits before using some more permanent way of building them, such as soldering the wires together or creating a printed circuit board.

- *LEDs:* An LED, or **L**ight **E**mitting **D**iode, functions as an electrical dam, allowing current to flow in one direction (if the electrical potential difference is high enough) but not the other. This is much like a dam, in that a dam will allow water to flow over it in one direction as long as the water level is high enough.

  Furthermore, if current is flowing (and there is enough of it), the LED will light up as a visual indicator that current is, in fact, flowing. Generally, the more current is flowing, the brighter the light will be up to some current maximum. Beyond that, the light may remain bright or possibly burn out. Note, however, that the LED cannot handle *too* much of a potential difference in the other direction - applying the voltage to the LED backwards can easily damage or destroy the LED, so always make sure you wire them in the proper direction. For the LEDs we are using, in general, the longer of the two leads should be wired with the higher voltage. Also, the plastic cap of the LED may have a small flat section along the cap - that corresponds to the side of the LED that should be on the side with a lower potential.

- *Resistors:* Resistors are electrical devices for which there is a particular ratio of voltage (or potential difference) across the device to current (or flow rate of charge) through the device. Resistors convert energy into heat for an electronic circuit the way dampers convert energy to heat for a mechanical system. The resistance value can be read from the side of the resistor by translating the colored bands properly. Another band gives the tolerance of the resistance - typically either 1%, 5%, or 10%. To continue with the water analogy, resistors are like rocks in the stream, taking energy out of the flowing water. The values for the particular colors are given on the Pratt Pundit page `Resistor Color Codes`.

1.7 Basic Digital Output

The DAQ cards used in this lab have eight digital input/output (DIO) channels. This section will use these as output channels, meaning MATLAB can tell the card to turn a particular channel on or off. In this case, "on" means a voltage somewhere between 4.25 and 5 V, and "off" means a voltage somewhere between 0 V and 0.4 V. This range is due to the nature of the digital outputs.

1.7.1 Hardware

To see how this works, you will wire the first three DIO channels (DIO0, DIO1, and DIO2) such that each controls an LED. For this part of the lab, you will need the following in addition to the CB-68LP and the breadboard:

(1) Three LEDs (one green, one yellow, and one small red)

(2) Three 470 Ω or 680 Ω resistors

(3) Three red wires and one black wire

Once you have the items above, make the connections in Table 1.1 on page DAQ 1 – 9 to make the circuit shown in Figure 1.1. Note that anything called a "Pin" or referred to as the letter A-J, X, or Y followed by a number indicates a socket on the breadboard – the letter is the row and the number is the column. Anything called a line "Line" or referred to as the letter L followed by a number indicates a gate on the CB-68LP connection block. Note that RED# and BLK# (and later, GRN#) in the table just refer to different wires.

To attach a wire to the breadboard, make sure that the end of the wire is straight and push it straight down into the connector. Push hard enough that you can feel the wire seat snugly, but do not push so hard that you pierce the bottom of the breadboard. For the LEDs, hold the leads of the LED in your hand and push them in that way - do not push the plastic cap as that generally ends up causing the leads to bend. For the resistors, to bend the leads, pinch the wire between your thumb and forefinger at a location close to, but not directly at the end of, the resistor with the canister itself in the fleshy part of your thumb and forefinger. Then bend the wire over your fingernail - this reduces the stress on the more delicate interface between the wire and the canister.

To attach a wire to the CB-68LP, you will need a screwdriver. First, open up the “gate” of the connection by turning the screw all the way counter-clockwise. You will see there is a thin metal slider with a window in it that moves as you turn the screw. Next, put the bare end of the wire in and turn the screw clockwise until it firmly holds the wire in place. Loose connections will result in either erratic or floating voltages being measured, which in turn will result in your having to perform the experiments over. You should also make sure that the **bare ends of the wires do not touch other wires or other gates or anything they are not supposed to**.

1.7.2 Software

Once you have made sure the circuit is connected correctly, you can begin writing the program to control the three lights. For this part, the red LED is connected to `DIO0`, the yellow LED is connected to `DIO1`, and the green LED is connected to `DIO2`. We will now go through the code required to access the data acquisition card and two different methods of determining which lights should turn on.

Open a new script file that you will save as `BasicDOutput.m`. First, you will need to write code that sets up the card and prepares it to act as an output device. The following code will clear the workspace, stop and clear any connections MATLAB currently has to the DAQ card, and creates a variable called `DIO` (the capital letters dee, eye, and oh) that we can use to talk with the card:

```
% Prepare workspace
clear; format short e

% Clear out DAQ object
delete(daqfind)

% Create Digital I/O Object
DIO = digitalio('nidaq', 1)
```

If you save and run this code, you will see the following message:

```
Display Summary of DigitalIO (DIO) Object Using 'PCI-6014'.

    Port Parameters:  Port 0 is line configurable for reading and writing.

      Engine status:  Engine not required.

DIO object contains no lines.
```

This means MATLAB has connected to the DIO portion of the DAQ card and is awaiting further instructions. Your next task is to tell MATLAB which of the 8 digital I/O lines you would like to use, and whether they should be inputs or outputs. For this program, you will be using lines 0 through 2 as outputs, and you can tell MATLAB that by adding the following code:

```
% Add output lines to Digital I/O Object
DOutputs = addline(DIO, 0:2, 'out')
```

Recall that the `0:2` is just a quick way of creating the vector `[0 1 2]`. Now if you run the code, you will get an additional message,

```
   Index:  LineName:  HwLine:  Port:  Direction:
   1       ''         0        0      'Out'
   2       ''         1        0      'Out'
   3       ''         2        0      'Out'
```

that shows you now have access to hardware lines 0, 1, and 2, that the DAQ has configured these as outputs, and that DIO0 is the first line, DIO1 is the second, and DIO2 is the third (based on their index).

There are two ways to control whether each line is on or off. The first method involves giving a vector of 0's and 1's to the digital output lines. For example, to test your circuit by turning all of the lights on, then pausing until the user hits a key, then turning all the lights off, you can add the code (note - lines 12 through 14 have been skipped, but will be added in later):

```
15 % Writing values to output lines using binary
16 putvalue(DOutputs, [1 1 1]);
17 fprintf('Press return to continue\n');
18 pause
19 putvalue(DOutputs, [0 0 0]);
```

and run your program. At this time, make sure that all three of your lights come on properly. If not, the most common mistake is to have put the LED in backwards - hit a key to turn the outputs off, fix any wiring mistakes you may have, and run the code again. If you cannot find a problem with your wiring, let a TA or the instructor know.

Also, before continuing, make sure MATLAB is not still waiting for you to hit return. At the bottom of the MATLAB window, just next to its Start button, MATLAB will indicate that it is waiting by stating "Paused: Press any key." Before running any other script, you will want to make sure that the previous script has completed running. You can do that by either hitting return or hitting CTRL-C to cancel the run entirely. If multiple runs have stacked up, stop them all before going on to the next step.

It is important to note that the data in the first entry of the vector is assigned to the first indexed line. For example, putting in the vector `[1 1 0]` will turn the red and yellow lights on, since red is the `DIO0` hardware line, which is indexed first, and yellow is the `DIO1` hardware line, which is indexed second. If you think of the three lines as binary digits, with green being 2^2, yellow being 2^1, and red being 2^0, you will need to enter the binary digits in *reverse order* to light the lights properly.

The question at this point, of course, is why not just have `DIO2` represent the 1's digit? The answer to that is given by the second way to turn the lights on. Instead of entering a vector of 0's and 1's, you can give the `putvalue` command an integer and it will convert the number to binary for you. Assuming you have set up your lines correctly, it will use your first indexed line as the 1's digit and your last indexed line as the largest, or most-significant digit. To demonstrate this, add the following to your code:

```
21 % Writing values to output lines using base 10
22 MyVal = 0
23 while 0<=MyVal & MyVal<=7
24     putvalue(DOutputs, MyVal)
25     fprintf('Displaying %0.0f as %c %c %c\n', ...
26         MyVal, dec2bin(MyVal, 3))

32     MyVal = floor(input('Enter a number between 0 and 7: '));
33 end
34
35 % Turn all outputs off
36 putvalue(DOutputs, [0 0 0])
```

then run your code. Put in numbers between 0 and 7 and notice how the lights properly display the binary version of the number (assuming a 1 is represented by a light that is on). Entering a number outside the range of 0 to 7 will stop the program.

The `dec2bin` command will take an integer and return a *string* containing the digits of the binary equivalent. The second argument - in this case 3 - makes sure that the string that is returned is at least that many characters long. This is useful since we want the number 0 to translate in binary as "000" and not just a single "0." The replacement code `%c` grabs a single letter from a string. The code below shows several different runs of the command:

```
>> dec2bin(10, 4) % 10 base 10 is 1010 base 2
ans =
1010
>> dec2bin(10, 5) % 10 base 10 is 1010 base 2;
                  % padded with 0 to make 5 chars
ans =
01010

>> dec2bin(10, 6) % 10 base 10 is 1010 base 2;
                  %padded with 00 to make 6 chars
ans =
001010

>> dec2bin(17, 1) % 17 base 10 is 10001 base 2; needs 5 chars so
                  % second argument of 1 is irrelevant
ans =
10001

>> dec2bin(17, 10) % 17 base 10 is 10001 base 2;
                   % padded with 00000 to make 10 chars
ans =
0000010001
```

Note that these are *strings* not numbers. As proof:

```
>> dec2bin(10,4)*1.0

ans =
    49    48    49    48
```

Finally, the last line of code makes sure all of the output channels are set back to 0 V before continuing - you will generally want to make sure your card is completely off at the end of a program.

1.8 Assignment

The assignment for this week includes expanding your circuit and writing four programs to work with the DAQ card. While you will not be writing a report for the work in lab this week, you will need to save your files to your OIT account for future reference. To get credit for this lab, you will need to have the TA check off that you have completed the four assignments, transferred the files to your OIT account, cleaned up your lab area, and cleared the Work directory on the PC.

In each case, make sure to include code that turns off all the LEDs at the end of the script. Leaving the CB-68LP energized may lead to accidental damage caused by stray voltages.

1.8.1 Adding a Digit

You will be expanding your circuit and your code to include a fourth binary digit such that your circuit can display values between 0 and 15. Save the `BasicDIO.m` file as `FourBits.m`. Use `DIO3` for the extra output. Change the code to accept numbers up to 15 instead of just 7. To make the circuit, you will use the remaining resistor, LED (should be the large red one), and red wire. Connect the LED between Pin J1 and Pin Y1 with the longer lead in Pin J1. Use the `MATLAB:CB-68LP Pinout` page to determine where to connect the other ends of the wire. When you believe you have the circuit working, show the TA your circuit and demonstrate your code at work.

1.8.2 Landing Lights

The lights on a landing strip are programmed to blink in such a way that you can tell the start from the end of the landing strip by the order in which the lights blink. Now that you have four lights, write a brand new script called `LandingLights.m` that will cause your lights to blink, one at a time, from left (most significant bit) to right (least significant bit), and repeat that ten times. Note that you may copy significant parts of your previous code into this program.

There are several ways to accomplish this task, most of which requiring at least one `for` loop. You may want to use the `pause()` function to generate brief pauses between commands turning the lights on and off. For example, `pause(0.01)` will pause for one-hundredth of a second. When you believe you have the circuit working, show the TA your circuit and demonstrate your code at work.

1.8.3 Random Lights

Next you will be writing a program to blink lights representing random numbers in binary. The script, called `RandomLights.m`, can be built by expanding on the `LandingLights` program or you can again start from scratch. Your program should first ask the user for a number of times to run a loop. Set up a `for` loop to run as many times as specified such that MATLAB generates a random integer between 0 and 15 and displays it in binary using the lights. The loop should pause for 0.1 seconds between numbers. When you believe you have the circuit working, show the TA your circuit and demonstrate your code at work.

1.8.4 Binary Game

Finally, save your `RandomLights.m` code as `BinaryGame.m` since this final script will require only minor modifications from that script. Still ask the user for a number of times to run a loop and generate and display a random number between 0 and 15 each time. However, instead of pausing for 0.1 seconds, this time ask the user to enter the decimal equivalent of the number that shows up in lights. Once the user has entered a number, have your code print out whether or not they were correct and, if not, what the answer should have been. At the end of the game, you should give the user a final score of how many correct entries they made out of the total number of attempts. When you believe you have the circuit working, show the TA your circuit and demonstrate your code at work. If you really want to work more on the program, you might also time the test and build some equation that generates a score based both on accuracy and speed.

1.8.5 Saving Files and Cleaning Up

To save the files, you will want to transfer them to your OIT directory. Go to the Start Button, then All Programs, then SSH Secure Shell, then Secure File Transfer Client. Log into your favorite OIT machine. The left side of the screen will be the file structure for the PC you are on and the right side will be the file structure of your OIT account. On the PC, your files are stored in

```
C:\Documents and Settings\mrglocal\MyDocuments\MATLAB
```

On the OIT account, go into your EGR53 directory and then create a new folder called `daq1` (you can make a new directory in the file transfer program). You can then drag and drop all your script files into your `daq1` folder. You do *not* need to copy any other files, such as `.asv` files. Close the SSH Secure File Transfer Client, then repeat this process for all people in the group. You do *not* need to change who is logged into the PC, only who is connecting with the SSH Secure File Transfer Client program.

Once you are sure all the files have been copied over, disassemble your circuit, straighten out the long wires, and put the smaller components back into the plastic bin. Next, delete all the files in the directory for MATLAB listed above - the easiest way is just to select them in the file list in the MATLAB window and delete them. *Leave MATLAB running at this point!* When you think your lab station is cleaned up and the files are cleared off, ask a TA to verify that you have all your parts and that you have emptied out the appropriate MATLAB directory - this will serve as attendance for the lab and you cannot receive a grade higher than 0 for this work without getting checked off for having properly stowed your gear and dumped your files. When checked off, close MATLAB, log out of the PC, and return the parts to the front of the room. Finally, make sure you know who you worked with in case the attendance record malfunctions.

1.9 Circuit Diagrams and Network Listings

1.9.1 Basic Digital Output

Item	First	Second	Note
LEDG	Pin J7	Pin Y7	Put longer lead of LED in row J
LEDY	Pin J13	Pin Y13	Put longer lead of LED in row J
LEDR	Pin J19	Pin Y19	Put longer lead of LED in row J
RES1	Pin D7	Pin G7	
RES2	Pin D13	Pin G13	
RES3	Pin D19	Pin G19	
RED1	Pin A7	Line L49	DIO2 - 2^2 digit
RED2	Pin A13	Line L17	DIO1 - 2^1 digit
RED3	Pin A19	Line L52	DIO0 - 2^0 digit
BLK1	Pin Y23	Line L7	DGND

Table 1.1: Network Listing for Basic Digital Output

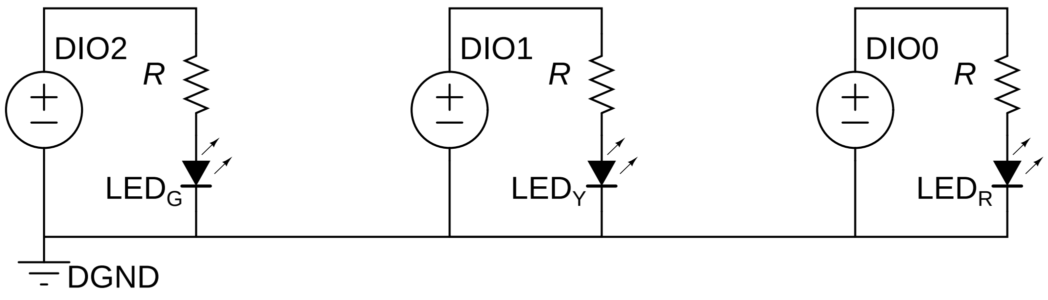

Figure 1.1: Circuit Diagram for Basic Digital Output

1.10 Code for `BasicDOutput.m`

Note: Extra spaces were left for code to take measurements from digital lines. Starting with MATLAB R2009a and the MX-series drivers from National Instruments, the NI PCI 6014e cards are not able to simultaneously set and read digital voltages. Hopefully this situation will be resolved; until then...blank space.

```
% Prepare workspace
clear; format short e

% Clear out DAQ object
delete(daqfind)

% Create Digital I/O Object
DIO = digitalio('nidaq', 1)

% Add output lines to Digital I/O Object
DOutputs = addline(DIO, 0:2, 'out')



% Writing values to output lines using binary
putvalue(DOutputs, [1 1 1]);
fprintf('Press return to continue\n');
pause
putvalue(DOutputs, [0 0 0]);

% Writing values to output lines using base 10
MyVal = 0
while 0<=MyVal & MyVal<=7
    putvalue(DOutputs, MyVal)
    fprintf('Displaying %0.0f as %c %c %c\n', ...
        MyVal, dec2bin(MyVal, 3))





    MyVal = floor(input('Enter a number between 0 and 7: '));
end

% Turn all outputs off
putvalue(DOutputs, [0 0 0])
```