Lab 1: Digital Input and Output

Report

**Owen Blair  
9/14/2021**

Contents

[Introduction 1](#_Toc82495073)

[Implementation 1](#_Toc82495074)

[Listing 1 2](#_Toc82495075)

[Listing 2 2](#_Toc82495076)

[Listing 3 3](#_Toc82495077)

[Listing 4 3](#_Toc82495078)

[Listing 5 4](#_Toc82495079)

[Testing and Verification 4](#_Toc82495080)

[Questions 5](#_Toc82495081)

[Conclusion 7](#_Toc82495082)

[Appendix 1: Full lab 1 C code 8](#_Toc82495083)

# Introduction

The purpose of this lab is to be able to become familiar with how to use the input and output pins of the PIC32. This includes writing to a pin and reading from a pin. In this case the pins that will be used are internally connected to buttons that are mounted to the Cerebot 32MX7cK PCB. Pins on this board can be configured to be an input or an output. This can be done using the TRIS register of the specific bit that is to be configured. It is also important to note that the Cerebot is designed to operate with 3.3V inputs and that anything below 0.66V is read as a logical value of zero and an input higher than 2.64V is read as a logic value of one. The only hardware needed for this lab is the Cerebot MX7cK processor board, a cable to connect the board to a computer via USB and a computer to program the board with.

The result of this lab is a program that can read what state button 1 and 2 (bit masked as BTN1 and BTN2) is in and turn on a corresponding LED. The truth table that dictates the desired behavior of the LEDs according to the state of the buttons can be seen in Table 1 below.

*Table 1: Truth Table For LED Behavior Given Button Inputs*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Inputs | | Outputs | | | |
| BTN 2 | BTN 1 | LED4 | LED3 | LED2 | LED1 |
| Off | Off | Off | Off | Off | On |
| Off | On | Off | Off | On | Off |
| On | Off | Off | On | Off | Off |
| On | On | ON | Off | Off | Off |

The entirety of the lab 1 code is in Appendix 1.

# Implementation

The implementation of the code for this lab went as planned. The only problems that were encountered were problems that were human error induced. This includes things like the misspelling of function names and pasting prewritten code sections in the wrong place.

The first section implemented was the initialization of the pins used as outputs, for the LEDs, and inputs, the buttons as well as ensuring that the pins have no preexisting statuses on the outputs. This is shown below in Listing 1. This section was implemented after the system initialization function and before any variables were assigned before the while(1) loop. The initialization of the buttons as outputs used the peripheral library function PORTSetPinsDigitalIn() with the inputs being the mask for port G, IOPORT\_G and the masks of button 1, BTN1, and button 2, BTN2, having a logical OR operator in between them. Initializing the outputs to be LEDs 1, 2, 3 and 4 was similar to how the buttons were initialized with the exceptions that the PORTSetPinsDigitalOut() peripheral library function was used and instead of the masks for the buttons the masks for all four LEDs sere logically ORed together. The definitions of each of the bit masks can be found in the CerebotMX7ck.h file. Lastly, the peripherial library function PORTClearBits() was used to clear the outputs of port g so that ant previous state of the LEDs would effect the operating of the LEDs while running the program.

### Listing 1

//Assign both bit 6 and 7 (buttons) as inputs

PORTSetPinsDigitalIn(IOPORT\_G, BTN1 | BTN2);

//Assign and clear the LED bits

PORTSetPinsDigitalOut(IOPORT\_G, BIT\_12 | BIT\_13 | BIT\_14 | BIT\_15);

PORTClearBits(IOPORT\_G, BIT\_12 | BIT\_13 | BIT\_14 | BIT\_15);

The code in Listing 2 is the main loop. The while(1) is an infinite loop where the functions read\_buttons(), decode\_buttons(), and control\_leds() were called continuously. The variables x and y that were initialized before the while(1) loop and are used to track button and control signal data as it goes from the read\_buttons() to decode\_buttons(), to the control\_leds() function and then the control signals out to the LEDs. Using pre-while(1) loop initialized variables also meant that I wasn’t initializing new variables every while(1) loop cycle and made ensuring that every function had the correct output and input easer.

### Listing 2

int main()

{

initialize\_system();

//Assign both bit 6 and 7 (buttons) as inputs

PORTSetPinsDigitalIn(IOPORT\_G, BTN1 | BTN2);

//Assign and clear the LED bits

PORTSetPinsDigitalOut(IOPORT\_G, BIT\_12 | BIT\_13 | BIT\_14 | BIT\_15);

PORTClearBits(IOPORT\_G, BIT\_12 | BIT\_13 | BIT\_14 | BIT\_15);

//Assign a variable to use

int x = 0;

int y = 0;

while(1)

{

x = read\_buttons();

y = decode\_buttons(x);

control\_leds(y);

}

return 1;

}

Listing 3 is the function read\_buttons(). This is the function definition that the program uses every time the read\_buttons() function is called. It has no input and outputs an integer. It reads the buttons by first assigning an integer variable, x, that will be used as the output. Next the periferial libaray function PORTReadBits(IOPORT\_G, BTN1 | BTN2) is used to read the state of the buttons. This function returns a 16-bit value from the port and x is set to the read value. Because the function is only reading two buttons the x variable is moved bitwise to the right six times to eliminate the need for a decode button function. This is reliable because the only bits that can affect the value of the x variable are the bits that are associated with buttons 1 and 2. The rest will be zeros. This shifting makes decoding the buttons easy. Lastly, the ‘return x;’ line is used to return the value x and leaves the function.

### Listing 3

int read\_buttons(void)

{

int x = 0;

x = PORTReadBits(IOPORT\_G, BTN1 | BTN2);

x = x >> 6;

return x;

}

The decode\_buttons() function, as seen in Listing 4, is a lot less interesting than the rest of the code because the read\_buttons() function does some preprocessing of the read data from the buttons before returning a value. The only thing this function does is return the value of the integer that was put in. The special thing is that if an integer is put in that is not a 0, 1, 2, or 3 the program defaults and returns a negative one. The negative one is meant as an error message and stops a ‘false high’ output from reaching the control\_leds() function. There is also come commented code that helps explain what control\_leds() should do with each return value.

### Listing 4

int decode\_buttons(int buttons)

{

switch(buttons)

{

case 0:

return 0;//All off

case 1:

return 1;//BTN1 on, BTN2 off

case 2:

return 2;//BTN1 off, BTN2 on

case 3:

return 3;//BTN1 on, BTN2 on  
 default:

return -1;//Throw an error value

}

}

Control\_leds() is used to assign what LED should be turned on based upon what value is passed to it. This function is in Listing 5. Control\_leds() has an input of an integer variable. Within the function definition this integer variable is labeled leds. The first thing that control\_leds() does is ensure that all LEDs are off. This ensures that any previously on LEDs do not stay on if a new button state is passed to the function. This is done using the PORTClearBits() peripheral library function. PORTClearBits() takes an input of IOPORT\_G and LED1, LED2, LED3, and LED4 logically ORed together. LEDs 1 to 4 have their bitmasks defined in CerebotMX7ck.h file. Control\_leds() uses a switch statement to help simplify the code and the eliminates need for if statements. If control\_leds() is passed an integer value between and including zero and 3 it writes the corresponding LED on. For example, if an integer value of 2 is passed to the function, only LED 3 will turn on. Each LED corresponds to the input integer plus one. So, an input value of zero will write LED 1 high, an input of 1 will write LED 2 high and so on and so forth until after a value greater than four is passed to the function. Then the function will not activate any of the switch cases.

### Listing 5

void control\_leds(int leds)

{

//Ensure LEDs are off

PORTClearBits(IOPORT\_G, LED1 | LED2 | LED3 | LED4);

int x =0;

switch(leds)

{

case 0: //Led 1 on

PORTWrite(IOPORT\_G,LED1);

break;

case 1: //Led 2 on

PORTWrite(IOPORT\_G,LED2);

break;

case 2: //Led 3 on

PORTWrite(IOPORT\_G,LED3);

break;

case 3: //LED 4 on

PORTWrite(IOPORT\_G,LED4);

break;

}

}

# Testing and Verification

Several software tools and visual observation was used to validate the functioning of the program. These included the watch variables, breakpoints, step into and step over software tools. The functionality of the program was also validated visually through observation of LEDs 1 to 4 to ensure that the behavior matched the behavior in Table 1.

The watch variables tool was used in conjunction with breakpoints, step into and step over tools. The watch variables tool allowed for a user to see the variables within the program change while the breakpoints and step into/over functions slowed the processor’s speed of computations to a speed that a human could follow. This meant that I could hold the buttons on the Cerebot PCB in a position, click step over and then examine the watch variables window to see what the program was reading from the input bits. Using step over tool again then allowed for the examination of data as it was processed and eventually what variable was being written to the output pins.

The visual inspection is more straight forward to use as a development/debugging tool because there was no manipulation of MPLAB X needed. Visual inspection consisted of examining of the PCB mounted LEDS and checking to see if all combinations of buttons being pressed were turning on the correct LED. Although this method of validation was easer than using software tools it also yielded less information than the software tools. The only information that visual inspection would return would be what combination of buttons was turning on what LED(s).

The steps to validate the functioning of the lab 1 program are as follows (assuming all hardware to run the program is setup correctly, the program is already loaded onto the Cerebot processor board):

1. If the answer to any of the following questions is no, then the program is not working correctly.
2. While not pressing any buttons inspect LEDs 1 through 4. Is LED 1 on and all other LEDs off?
3. While only pressing button 1 inspect LEDs 1 through 4. Is LED 2 on and all other LEDs off?
4. While only pressing button 2 inspect LEDs 1 through 4. Is LED 3 on and all other LEDs off?
5. While only pressing buttons 1 and 2 inspect LEDs 1 through 4. Is only LED 4 on and all other LEDs off?
6. If the answer to all previous questions was yes, then the program works correctly.

# Questions

Given the bit positions within the port of the buttons, the appropriate button ‘state’ is identified by taking the input integer from the PORTReadBits() function and shifted over 6 bits. This would make the values being passed from function to function a 0, 1, 2, or 3. Given the function prototypes, the variables that will be needed to declared in the main function are two integer variables to store the data returned from the read\_buttons() function and the value returned from the decode\_buttons() functions. Within each function a separate integer variable needs to be declared so that the function can return a value and do operations on. This lab does not need pass by reference values depending on how you pass data from function to function. If variables are declared outside of the while(1) loop pass by reference is not needed.

Using Figure 1 below, the following expressions and corresponding vlaues were found for the voltage at the pin and the current flowing through R2 for while the button is open and while the button is closed.

Button Open:

Button open values:

Vpin = 3.293E-3 Volts  
 R2 = 3.293E-10 Amps

Button Closed:

Button closed values:

Vpin = 3.297 Volts  
 R2 = 3.297E-7 Amps

Diagram, schematic

Description automatically generated External pull-down and pull-up resistors are incredibly important. The purpose of the pull up/down resistors is to be able to control the amount of voltage at a pin and the current flowing into the pin form the outside world. If an input doesn’t have a pull-down resistor, then the pin will take all the current from the Vdd source and could damage the internal circuitry of the pin and possibly the processor. For example, if R1 was missing from Figure 1 and the button was pressed then all the current flowing through the button would then go through the pin and into the PIC32 circuitry. If R1 was missing and the button wasn’t pressed, then the pin would have a voltage of 1.65 because it is connected between the two 10MΩ ‘leakage’ resistors and leave the pin floating.

If an LED has a voltage drop of 0.7 volts, the PIC32 IO pin has an internal 200Ω resistor[[1]](#footnote-1) and is writing a high of 3.3 volts the current the pin is sourcing is 13mA. This is within the maximum of 18 mA that a single IO pin can source/sink[[2]](#footnote-2). All IO pins simultaneously can sink a maximum of 200mA[[3]](#footnote-3).

# Conclusion

This lab was a success and LEDs 1 through 4 were turning on at the correct times without issue. One drawback of this program is that if when the while(1) loop is running, and something takes a while to run a button press/depress might not be detected. This is mitigated in the program by having the PIC32 update as fast as possible. If more code was added after the read buttons there could be more problems because of the time needed for the code that executes between read\_button() events.

# Appendix 1: Full lab 1 C code

/\*\*\*\*\*\*\*\*\*\*\*\*\* Project 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File name: Project1.c

\*

\* Author: Richard W. Wall

\* Date: August 10, 2013 - Tested with MPLab X

\* Notes: This project converts the binary encoded buttons to

\* set one of the four Cerebot MX7cK LEDs.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <plib.h>

#include "CerebotMX7cK.h"

#include "Project1.h"

int main()

{

initialize\_system();

//Assign both bit 6 and 7 (buttons) as inputs

PORTSetPinsDigitalIn(IOPORT\_G, BTN1 | BTN2);

//Assign and clear the LED bits

PORTSetPinsDigitalOut(IOPORT\_G, BIT\_12 | BIT\_13 | BIT\_14 | BIT\_15);

PORTClearBits(IOPORT\_G, BIT\_12 | BIT\_13 | BIT\_14 | BIT\_15);

//Assign a variable to use

int x = 0;

int y = 0;

while(1)

{

x = read\_buttons();

y = decode\_buttons(x);

control\_leds(y);

}

return 1;

}

/\* initialize\_system Function Description \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SYNTAX: void initialize\_system();

\* PARAMETER1: No Parameters

\* KEYWORDS: initialize

\* DESCRIPTION: Sets Registers to default (0) to prevent non zero values

\* from popping up.

\* RETURN VALUE: None

\* END DESCRIPTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initialize\_system()

{

Cerebot\_mx7cK\_setup(); // Initialize processor board

}

/\* read\_buttons Function Description \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SYNTAX: int read\_buttons(void);

\* KEYWORDS: button, read, inputs

\* DESCRIPTION: Reads the status of the input buttons. Button status is

\* reported for button bit positions only. All other bits in the

\* returned value are set to zero as shown below:

\*

\* Port G Bit position [15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0]

\* Port G Bit value [ 0| 0| 0| 0| 0| 0| 0| 0|B2|B1| 0| 0| 0| 0| 0| 0]

\*

\* B1 will be 1 if BTN1 is pressed otherwise B1 will be zero

\* B2 will be 1 if BTN2 is pressed otherwise B2 will be zero

\*

\* PARAMETERS: None

\* RETURN VALUE: int variable containing button status

\* NOTES: BTN3 on the Cerebot MX7cK is used in this function. Adding

\* BTN3 detection is an assignment left to the reader.

\* Refer to Cerebot MX7cK reference manual.

\* END DESCRIPTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int read\_buttons(void)

{

int x = 0;

x = PORTReadBits(IOPORT\_G, BTN1 | BTN2);

x = x >> 6;

return x;

}

/\* decode\_buttons Function Description \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SYNTAX: int decode\_buttons(int);

\* KEYWORDS: button, decode, LEDs

\* DESCRIPTION: Maps the button status to light one of four LEDS

\*

\* Port G Bit position [15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0]

\* Port G Bit value [L4|L3|L2|L1| X| X| X| X| X| X| X| X| X| X| X| X]

\*

\* LED1 = ON for L1 = 1, LED1 = OFF for L1 = 0

\* LED2 = ON for L2 = 1, LED2 = OFF for L2 = 0

\* LED3 = ON for L3 = 1, LED3 = OFF for L3 = 0

\* LED4 = ON for L4 = 1, LED4 = OFF for L4 = 0

\* X = Don't care

\*

\* PARAMETER1: button status

\* RETURN VALUE: LED 1 through 4 ON/OFF control

\* NOTES: Refer to Cerebot MX7cK data sheet for button and LED bit

\* positions

\* END DESCRIPTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int decode\_buttons(int buttons)

{

switch(buttons)

{

case 0:

return 0;//All off

case 1:

return 1;//BTN1 on, BTN2 off

case 2:

return 2;//BTN1 off, BTN2 on

case 3:

return 3;//BTN1 on, BTN on

}

}

/\* control\_leds Function Description \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SYNTAX: void control\_leds(int leds);

\* KEYWORDS: led, output, lights

\* DESCRIPTION: lights the LEDs specified the bit positions in the input

\* argument.

\* PARAMETER1: integer LED1 through LED4 ON/OFF control

\* RETURN VALUE: None.

\* NOTES: Only bit positions related to LED1 through LED4 are modified

\* by this function. The BRD\_LEDS mask constant initially clears

\* all bit LATG positions associated with LED1 through LED4.

\* The BRD\_LEDs mask is also used to limit the Port G bits that

\* can be modified to thoes in the BRD\_LEDS mask field.

\* The read-modify-write instruction is not atomic.

\* Refer to Cerebot MX7cK data sheet for LED bit positions

\* END DESCRIPTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void control\_leds(int leds)

{

//Ensure LEDs are off

PORTClearBits(IOPORT\_G, LED1 | LED2 | LED3 | LED4);

int x =0;

switch(leds)

{

case 0: //Led 1 on

PORTWrite(IOPORT\_G,LED1);

break;

case 1: //Led 2 on

PORTWrite(IOPORT\_G,LED2);

break;

case 2: //Led 3 on

PORTWrite(IOPORT\_G,LED3);

break;

case 3: //LED 4 on

PORTWrite(IOPORT\_G,LED4);

break;

}

}

/\* End of Project1.c \*/

1. Pg 7, digilent\_pro\_MX7\_rm.pdf, from 340 Handouts folder [↑](#footnote-ref-1)
2. Pg 8, digilent\_pro\_MX7\_rm.pdf, from 340 Handouts folder [↑](#footnote-ref-2)
3. Pg 8, digilent\_pro\_MX7\_rm.pdf, from 340 Handouts folder [↑](#footnote-ref-3)