GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Fall 2017

Lab #2: General Purpose Inputs and Outputs

Contents

1 Background Material

1.1 Introductory Comments

The versatility of microcontrollers is a consequence of their ability to combine internal numerical computation with external component interaction to achieve some desired function from a single programmable chip. General-purpose input-output (GPIO) represents the simplest of the various digital interfacing options. The objective of this lab is to introduce GPIO, including how to initialize and then utilize the GPIO module. You will develop code that leverages the hardware abstraction layer (HAL) in order to interact with the GPIO configuration and data registers in the most straightforward manner, and this skill will be applied to the design of systems that read from external switches and write to external light-emitting diodes. For stand-alone or embedded operating capability, data may be stored in volatile memory but code must be stored in non-volatile

memory, so you will use a (non-default) linker command file to locate program code in on-chip flash memory and program data in on-chip RAM memory.

1.2 Relevant Microcontroller Documentation

The overall objective of these lab projects is to teach you how to do embedded design with microcontrollers in a general sense, not just how to approach one specific application using one specific microcontroller. Therefore, the guidance provided herein focuses more on general thought processes and programming recommendations; step-by-step instructions of an extremely specific nature have been intentionally omitted. Use fundamental documentation as your primary source of information as you work through details of implementation. Being able to read and understand such documentation is an important skill to develop, as similar documentation would need to be consulted in order to use other microcontrollers or other application hardware. By making the effort to extract required details from fundamental documentation yourself, you will have developed transferable skills that will serve you well in your engineering career. For this lab, review:

- Datasheet
  - Figure 4-2, for pin assignments on the 100-pin microcontroller component.
- Technical Reference Manual
  - §1.4.1, for an overview of the GPIO module.
  - §1.4.2, for a discussion of GPIO module initialization.
  - §1.4.3, for a discussion of GPIO module utilization.
  - §1.4.6, for `GPxMUXy`, `GPxDIR` and `GPxPUD` register field descriptions (initialization).
  - §1.4.6, for `GPxDAT` register field descriptions (read utilization).
  - §1.4.6, for `GPxSET`, `GPxCLEAR` and `GPxTOGGLE` register field descriptions (write utilization).

In addition to these documents, you should also review the schematic diagrams of the microcontroller daughtercard and the peripheral explorer motherboard.

1.3 Target Hardware Schematic Diagrams and Data Sheets

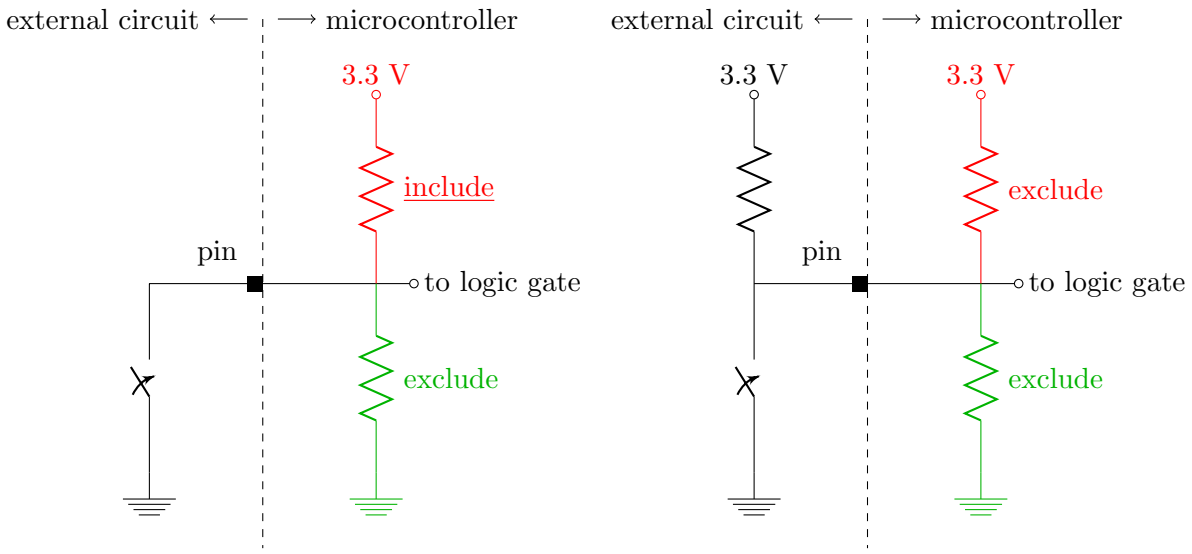
The GPIO pins are located on the microcontroller component, which in turn is located on the microcontroller daughtercard, which in turn is located on the peripheral explorer motherboard. The daughtercard and motherboard are designed such that input devices located on the motherboard (e.g. pushbutton PB1) and output devices located on the motherboard (e.g. light-emitting diode LD1) are electrically connected to certain GPIO pins located on the microcontroller. If it is desired to utilize such input/output devices on the motherboard, then it will be necessary to examine the target hardware schematic diagrams to determine the required electrical connection details.

Our daughtercard uses the F28069 microcontroller component with the 100-pin package option, and pin assignments for this package option are shown in Figure 4-2 of the Datasheet document. Some pins are dedicated to one particular peripheral module, e.g. pin 23 is dedicated as ADCIN-A0. However, other pins may be used for multiple purposes, e.g. pin 87 may serve either as GPIO-0 or as EPWM-1A, depending on the associated multiplexer setting. Which GPIO pins should be used in order to activate motherboard input device PB1 and motherboard output device LD1, what is the logic state convention for PB1 (i.e. will pressing PB1 correspond to reading a `0` or `1` from the

appropriate GPIO pin) and what is the logic state convention for LD1 (i.e. will illumination of LD1 be achieved by writing a `0` or `1` to the appropriate GPIO pin)?

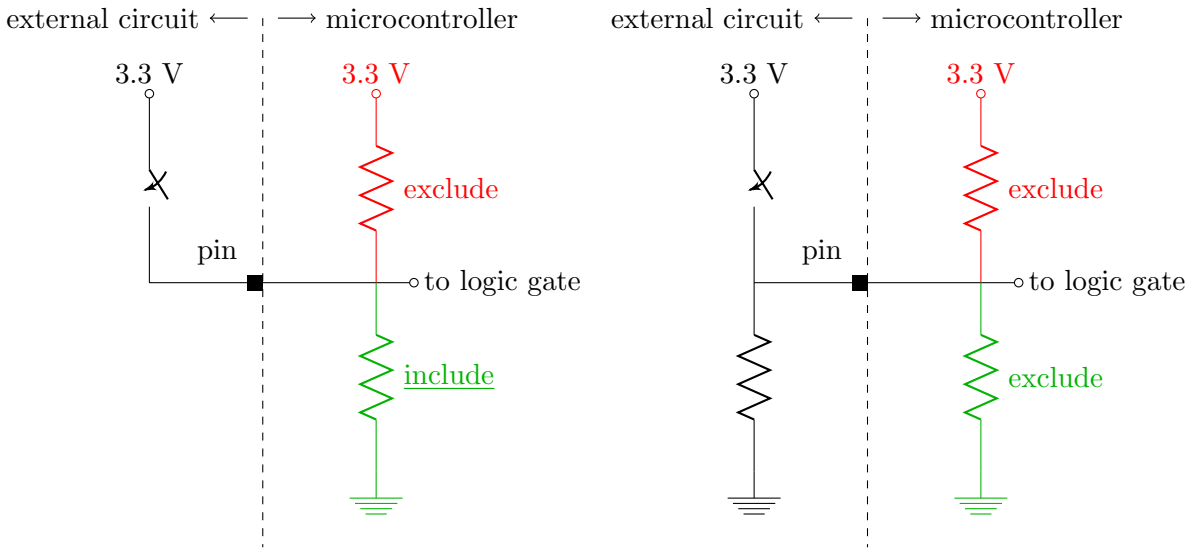
To answer these questions, first locate PB1 and LD1 on the motherboard schematic, and trace the interface circuits back to the 100-pin connector labeled J1; the component SN74LVC2G17 is a non-inverting buffer used to enable adequate current flow through LD1. The pins on connector J1 are numbered 1 through 100, but they are also labeled in a more revealing way. For example, pin 57 on J1 is labeled ADC-A0, and if we consult the daughtercard schematic we see that pin 57 on its connector P1 is also labeled ADC-A0. If we then trace that signal back to the microcontroller chip U1, we see that it enters the microcontroller at pin 23 after passing through an RC low-pass filter circuit. Similarly, pin 23 on J1 is labeled GPIO-0, the same pin on P1 is identically labeled, and the signal traces back to U1 pin 87. The DATASHEET document already established that microcontroller pin 23 was signal ADC-A0 and microcontroller pin 87 was signal GPIO-0, so this analysis indicates that the labels placed on connector J1 of the motherboard schematic may be generally indicative of the ultimately connected microcontroller pins; for multi-purpose microcontroller pins, the connector J1 labels reflect specifically the GPIO signal number.[1]

The concept of pull-up and pull-down resistors, for GPIO pins multiplexed as inputs, is now reviewed. The objective of such resistors is to guarantee that the logic levels presented to internal logic gates are always well defined, i.e. logic 0 or logic 1. A resistor located between a pin and the supply terminal is referred to as a pull-up resistor, whereas a resistor located between a pin and the ground terminal is referred to as a pull-down resistor. In the two schematic diagrams appearing below, a mechanical switch has been introduced so as to provide a means for establishing a voltage of 0 V at the pin when the switch is closed. In the schematic on the left, the external circuit allows the pin to float at an undetermined voltage if the switch is opened; on the other hand, in the schematic on the right, the external circuit includes a pull-up resistor to impose a voltage of approximately 3.3 V at the pin if the switch is opened. Microcontrollers may provide the option of enabling or disabling internally located pull-up (red) or pull-down (green) resistors, in order to reduce the number of required external components. For the cases under consideration, internal resistance is needed only in the left schematic, and it should be a pull-up resistance.



[1]The labels on connector J1 of the motherboard schematic were chosen specifically for the F28335 device that ships with the peripheral explorer kit (see label at bottom of J1). Since we are using the F28069 device instead, we will find that the motherboard schematic labeling of J1 is not always correct. Therefore, to be sure we need to trace the signals of interest back to pins on the F28069 device using the daughtercard schematic as explained above.

Now consider the next two schematic diagrams below, in which a mechanical switch has been utilized for the purpose of imposing a voltage of 3.3 V at the pin when the switch is closed. On the left the pin voltage imposed by the external circuit is undefined when the switch is opened, whereas on the right the pin voltage imposed by the external circuit is approximately 0 V when the switch is opened. If it is desired to minimize the number of external components, internal resistance may be used instead of external resistance, in which case the internal resistance should be a pull-down resistance. Our microcontroller offers only internal pull-up resistance, so pull-down resistance if needed would have to be added externally (as on the right).



2 GPIO Module: Step-by-Step Guidelines

2.1 Initialize the GPIO Registers

2.1.1 Set the Multiplexers

To provide maximum design flexibility, many pins on the microcontroller are internally multiplexed. For example, a single pin may serve as a GPIO pin or in support of some other peripheral circuit. Considering Figure 4-2 of the DATASHEET document, we see that microcontroller pin 87 is internally configurable as either GPIO-0 or EPWM-1A. The internal circuitry that establishes the desired internal connections to the microcontroller pins is referred to as multiplexer circuitry, and the `GPxMUXy` registers are used to set the multiplexers to achieve the desired configuration (`x = A,B` and `y = 1,2`). The two-bit fields within each of the `GPxMUXy` registers specify which of up to four possible internal circuits each microcontroller pin is to be connected to. The HAL-defined variable names available for initializing the `GPxMUXy` registers are as follows:

```
GpioCtrlRegs.GPxMUXy.bit.GPIOz
```

Proper assignment of the fields within these registers requires review of the relevant register diagrams and register field descriptions in §1.4.6 of the TECHNICAL REFERENCE MANUAL document.

2.1.2 Assign the Signal Directions

All microcontroller pins set to provide GPIO functionality may be assigned as inputs or outputs, and the `GPxDIR` registers are used for this purpose (`x = A,B`). The assignment of direction is made

by specifying the appropriate one-bit fields within each of the GPxDIR registers. The HAL-defined variable names available for initializing the GPxDIR registers are as follows:

```
GpioCtrlRegs.GPxDIR.bit.GPIOy
```

Proper assignment of the fields within these registers requires review of the relevant register diagrams and register field descriptions in §1.4.6 of the Technical Reference Manual document.

2.1.3 Configure the Pull-Up Resistors

All microcontroller pins set to provide GPIO input functionality may be internally connected to effective pull-up resistors if desired, and the GPxPUD registers are used for this purpose (x = A,B). If enabled, the pull-up effect is implemented by a transistor circuit. Enabling or disabling pull-up resistors is achieved by specifying the appropriate one-bit fields within each of the GPxPUD registers. The HAL-defined variable names available for initializing the GPxPUD registers are as follows:

```
GpioCtrlRegs.GPxPUD.bit.GPIOy
```

Proper assignment of the fields within these registers requires review of the relevant register diagrams and register field descriptions in §1.4.6 of the Technical Reference Manual document.

2.2 Utilize the GPIO Pins

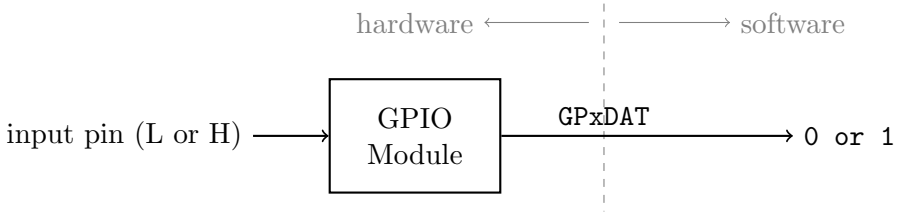
2.2.1 Read from a GPIO Input Pin

The GPxDAT registers indicate the current state of the GPIO pins (after qualification), as opposed to the state of the output latch associated with these pins, and this is true irrespective of how these pins have been configured (GPIO inputs, GPIO outputs, or some other peripheral functions). Each input-output port (x = A,B) has one such register and each bit within these registers corresponds to one GPIO pin. Typically, one reads from the GPxDAT registers but does not write to them.

The diagram below visualizes the signal flow associated with GPxDAT registers. The HAL-defined variable names available for accessing GPxDAT registers are as follows:

```
GpioDataRegs.GPxDAT.bit.GPIOy
```

The relevant register diagrams and register field descriptions may be found in §1.4.6 of the Technical Reference Manual document.



2.2.2 Write to a GPIO Output Pin

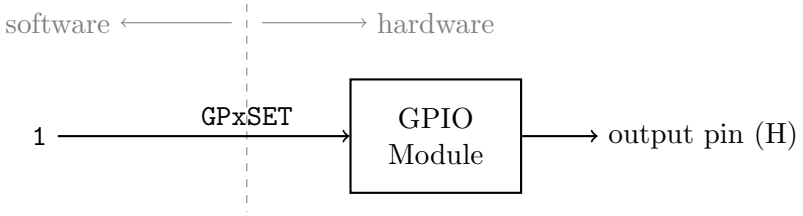
The GPxSET registers are used to drive specified GPIO pins high without disturbing other pins. Each input-output port (x = A,B) has one such register and each bit within these registers corresponds to one GPIO pin. If a GPIO pin is configured as an output, then writing a 1 to the corresponding bit in the GPxSET register will set the output latch and the pin will be driven high. If the pin is not configured as a GPIO output, then the value will be latched but the pin will not

be driven. Only if the pin is later configured as a GPIO output will the latched value be driven onto the pin. Writing a `0` to any bit in the `GPxSET` registers has no effect.

The diagram below visualizes the signal flow associated with `GPxSET` registers. The HAL-defined variable names available for manipulating `GPxSET` registers are as follows:

```
GpioDataRegs.GPxSET.bit.GPIOy
```

The relevant register diagrams and register field descriptions may be found in §1.4.6 of the Technical Reference Manual document.

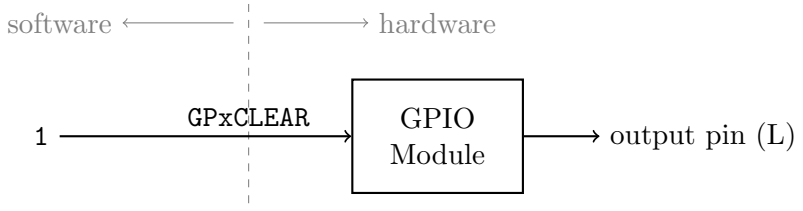


The `GPxCLEAR` registers are used to drive specified GPIO pins low without disturbing other pins. Each input-output port (`x = A,B`) has one such register and each bit within these registers corresponds to one GPIO pin. If a GPIO pin is configured as an output, then writing a `1` to the corresponding bit in the `GPxCLEAR` register will clear the output latch and the pin will be driven low. If the pin is not configured as a GPIO output, then the value will be latched but the pin will not be driven. Only if the pin is later configured as a GPIO output will the latched value be driven onto the pin. Writing a `0` to any bit in the `GPxCLEAR` registers has no effect.

The diagram below visualizes the signal flow associated with `GPxCLEAR` registers. The HAL-defined variable names available for manipulating `GPxCLEAR` registers are as follows:

```
GpioDataRegs.GPxCLEAR.bit.GPIOy
```

The relevant register diagrams and register field descriptions may be found in §1.4.6 of the Technical Reference Manual document.

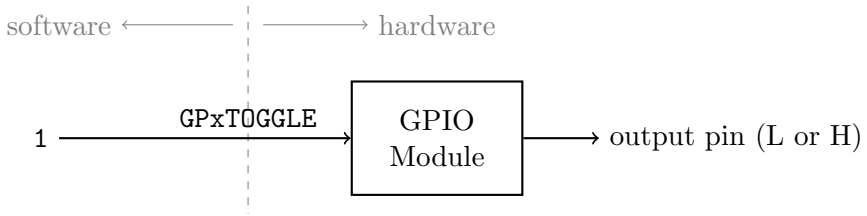


The `GPxTOGGLE` registers are used to drive specified GPIO pins to an opposite level without disturbing other pins. Each input-output port (`x = A,B`) has one such register and each bit within these registers corresponds to one GPIO pin. If a GPIO pin is configured as an output, then writing a `1` to the corresponding bit in the `GPxTOGGLE` register flips the output latch and drives the pin as follows: if the output pin is low, then it will be driven high; if the output pin is high, then it will be driven low. If the pin is not configured as a GPIO output, then the value will be latched but the pin will not be driven. Only if the pin is later configured as a GPIO output will the latched value be driven onto the pin. Writing a `0` to any bit in the `GPxTOGGLE` registers has no effect.

The diagram below visualizes the signal flow associated with `GPxTOGGLE` registers. The HAL-defined variable names available for manipulating `GPxTOGGLE` registers are as follows:

```
GpioDataRegs.GPxTOGGLE.bit.GPIOy
```

The relevant register diagrams and register field descriptions may be found in §1.4.6 of the Technical Reference Manual document.



3 Flash-Based Implementation

3.1 Types of Microcontroller Memory

The normal work flow during embedded code development includes working in CCS edit mode to create or modify the `main.c` file and then switching to CCS debug mode to actually run the resulting executable program on the microcontroller. The linker command file added by CCS during project creation determines which portions of the microcontroller memory will be used for storing the code (program instructions) and for storing the data (program variables).

Throughout the initial stages of embedded code development, it is recommended to use on-chip RAM memory to store both the code and the data. The default linker command file added to our CCS projects is `28069_RAM_lnk.cmd` and, as its name suggests, the corresponding type of debug session is one in which on-chip RAM memory is the only type of memory being used. However, since RAM is volatile memory which does not retain its contents after power is removed, the default type of debug session does not actually prepare the microcontroller for deployment in an embedded stand-alone product. Fortunately, the F28069 microcontroller has internal flash memory, a type of non-volatile memory, in which to store application code to enable true stand-alone operation.

3.2 Flash Programming Procedure

The steps that need to be followed in order to utilize the internal flash memory are not difficult. First of all, once you are satisfied with your RAM-based project, you will need to create a new CCS project dedicated to the flash-based implementation. Use the same procedure as before to create the new project, but include "flash" in its name to distinguish it from the project on which it will be based. Once the flash-specific project has been created, do the following:

1. Copy source file `main.c` from the RAM-based project into the flash-based project.
2. Delete linker command file `28069_RAM_lnk.cmd` from the flash-based project.
3. Right-click the flash-based project and select `Properties`. Under `General`, find `Linker command file:` and `Browse...` to select the linker command file `F28069.cmd`.
4. Right-click the flash-based project and select `Add Files...` to copy the assembly language file `F2806x_CodeStartBranch.asm` from the folder `C:\F28069\flash`.
5. Build the flash-based project and enter the debug mode in order to automatically initiate the process of programming the flash memory.
6. Once programming is complete, terminate the debug session without running code.

7. Disconnect target hardware from development system to begin stand-alone operation.

The purpose of initiating a debug session for the flash-based project is simply to program the flash memory inside the microcontroller, and this activity will occur automatically when the debug session is initiated. You will see messages indicating that the flash memory is being erased, programmed and verified. Once the flash memory has been properly prepared, no more messages will be displayed and the debug session may be terminated. At no point would you actually need to run your program from within the debugger; instead, you would simply disconnect your target hardware from the development system and then use it as a stand-alone embedded component.

3.3 Start-Up Sequence

The F28069 microcontroller contains an internal ROM boot-loader that determines where in memory the code execution will begin after a device reset (e.g. after power up). Three pins—TRST (pin 12), GPIO-34 (pin 68, pin 46 of J1) and GPIO-37 (pin 70, pin 98 of J1)—are automatically sampled when the device is powered up, and the state of these pins determines which of 5 possible start-up sequences will occur. Examination of the TECHNICAL REFERENCE MANUAL §2.2.1 shows that the state of TRST determines if an emulation boot or a stand-alone boot is to occur; §2.2.6 states that both GPIO-34 and GPIO-37 will have pull up resistors enabled by default after device reset; and §2.2.9 indicates that one type of stand-alone boot option called GetMode—which runs program code from flash memory by default—is selected if TRST is low and both GPIO-34 and GPIO-37 are high. Examination of the microcontroller daughtercard schematic shows that TRST is pulled low by a resistor and both GPIO-34 and GPIO-37 are pulled high by resistors due to switch array SW1, and examination of the peripheral explorer motherboard schematic shows that jumper J1 could be used to pull GPIO-34 low. Considering all of this information, stand-alone operation with program code execution from flash memory will be assured if power is applied with the USB cable disconnected and with no jumper placed across J1 on the peripheral explorer motherboard.

3.4 Flash Wait States

The use of on-chip flash memory is certainly a convenient way to achieve stand-alone operating capability. Its use eliminates any need for external non-volatile memory or a host processor from which to boot-load the application code. On the other hand, the flash memory is not as fast as the RAM memory; after a device reset, flash memory accesses will require 15 wait states by default compared to RAM memory accesses which require 0 wait states. These wait states may be thought of as "wasted" clock cycles. In many applications, including the applications pursued in this lab, these wait states would not be a problem and could be ignored altogether. For those applications that cannot afford to waste any clock cycles, there exist two alternatives. One approach would be to configure the flash control registers to reduce the number of wait states to a (nonzero) minimum, and then to enable flash pipelining so as to fetch multiple instructions from flash simultaneously. Another approach is to copy time-critical portions of application code, or even all application code, from flash to RAM prior to execution at run time (assuming there is enough on-chip RAM).

4 Lab Assignment

4.1 Pre-Lab Preparation

Each individual student must work through the pre-lab activity and prepare a pre-lab deliverable to be submitted *by the beginning of the lab session*. The pre-lab deliverable consists of a brief typed

statement, no longer than one page, in response to the following pre-lab activity specification:

1. Read through this entire document, and describe the overall purpose of this week's project.
2. Describe the purpose of chips U5 and U6 located on the peripheral explorer motherboard.
3. Describe in specific terms how the relevant registers will be used to complete the tasks assigned in §4.2. Focus on those HAL variables identified in §2: if you will write to a HAL variable, state the numerical value to be written; if you will read from a HAL variable, state how the numerical value read will be interpreted. You will need to consult circuit diagrams in order to complete this part of your pre-lab preparation; there are 10 pins to consider.
4. Considering the tasks assigned in §4.2, should pull-up resistors on specific GPIO input pins be enabled or disabled? Why?

Please note that it is not essential to write application code prior to the lab session; the point of the pre-lab preparation is for you to arrive at the lab session with firm ideas regarding register usage and other relevant issues in relation to the tasks assigned in §4.2.

4.2 Specification of the Assigned Tasks

4.2.1 Displaying Status of Pushbuttons on LED Array

Develop code that continuously determines the status of pushbuttons PB1 and PB2 and operates the LED array to reflect that status. More specifically, LD1/2 should be on when PB1/2 is pressed and off when PB1/2 is not pressed, and LD3 and LD4 should be always off. The watchdog module should be disabled/enabled at the beginning/end of the device initialization code prior to `while(1)`, and it should be periodically serviced within the `while(1)` continuous loop.

Begin your work for this task in a RAM-based project. After you have your system working as desired, create a corresponding flash-based project and program the on-chip flash memory as previously described. Once the flash memory has been programmed, first remove the USB cable from the target hardware and then connect the 5 V external power supply to the target hardware. Demonstrate proper functioning of your system in stand-alone mode with power cycling reset.

Instructor Verification (separate page)

4.2.2 Displaying Status of HEX Encoder on LED Array

Develop code that continuously determines the status of the HEX encoder and operates the LED array to reflect that status. More specifically, the binary value displayed on the LED array should correspond to the hexadecimal value displayed on the dial of the encoder, with the understanding that a `1/0` is indicated by an on/off LED and that LD1/LD4 is the most/least significant bit. The watchdog module should be disabled/enabled at the beginning/end of the device initialization code prior to `while(1)`, and it should be periodically serviced within the `while(1)` continuous loop.

Begin your work for this task in a RAM-based project. After you have your system working as desired, create a corresponding flash-based project and program the on-chip flash memory as previously described. Once the flash memory has been programmed, first remove the USB cable from the target hardware and then connect the 5 V external power supply to the target hardware. Demonstrate proper functioning of your system in stand-alone mode with power cycling reset.

Instructor Verification (separate page)

GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Fall 2017

Lab #2: General Purpose Inputs and Outputs

Instructor Verification Page

Lab Section	Begin Date	End Date
L01, L02	September 5	September 12
L03, L04	September 7	September 14

To be eligible for full credit, do the following:

1. Submissions required by each student (one per student)
   (a) Upload your pre-lab deliverable to tsquare before lab session begins on begin date.
   (b) Upload your `main.c` file for §4.2.2 to tsquare before lab session ends on end date.
2. Submissions required by each group (one per group)
   (a) Submit a hard-copy of this verification page before lab session ends on end date.

Name #1: _____

Name #2: _____

Checkpoint: Verify completion of the task assigned in §4.2.1.

Verified: _____ **Date/Time:** _____

Checkpoint: Verify completion of the task assigned in §4.2.2.

Verified: _____ **Date/Time:** _____