Embedded Control Laboratory 1-1

Digital Inputs/Outputs

SIMULATED LABORATORY

→ Readings

- Lab Manual: *Chapter 3 - Programming in C*
- Lab Manual: *Chapter 4 - The Silicon Labs C8051F020 and the EVB (Input/Output Ports and Timers)*
- Lab Manual: *Chapter 5 - Circuitry Basics and Components*
- Lab Manual: *Chapter 8 - Troubleshooting*

→ Laboratory Goals

General

- Familiarization with Code:Blocks IDE and procedures for development and testing of an embedded C program.
- Learn basic use of *General Purpose Inputs and Outputs (GPIO)*

→ Motivation

The ability of a microprocessor to interface with other digital devices used as sources of input (e.g., switches, a keyboard, etc.), destinations for output (e.g., relays, motors, LEDs, etc.) or for input and output (e.g., another microprocessor) opens up a world of possibilities. Moreover, for a microprocessor to serve as the controller for a host system, it must be able to acquire input (digital or analog), and it must be able to provide output (digital or analog) in response to those inputs.

The most common form of a digital port is a general purpose input/output (GPIO) while less commonly there are fixed input ports and fixed output ports. GPIO ports and pins may have some or all bits programmed via software instructions for either input or output. All of the ports on the C8051F020 used in this laboratory are fully configurable.

In this lab, you will be introduced to configuring GPIO ports as digital input and digital output ports. With the attainment of the stated objectives, you will have gained an understanding of how to develop a simple C program for the C8051. You will be able to configure the C8051 to acquire a digital input from an external source and use this value to determine the output on one of its digital output ports.

→ General Lab Description

Hardware

In this lab you will develop the components necessary for a portion of a microprocessor- controlled game. In this lab, you will connect a regular LED, a bi-color LED, 2 pushbuttons, a slide switch, and a DC motor. Additionally, you will create software for the microcontroller to read inputs from the pushbuttons and slide switch and produce outputs to the LEDs and DC motor.

The pushbutton and slide switch, Figures 1 and 2, will act as simple switches when connected in the manner shown in the circuit schematic. When the pushbutton is pressed or the slide switch is in the "on" position, the circuit is closed causing the voltage to drop across the resistor and a logic LOW 0 V to be read at the microcontroller input pin. The pushbutton returns to a normally-open state when released causing a logic HIGH 5 V to the input pin. The slide switch will have a logic HIGH at the input pin when the switch is in the "off" position as the slide switch is open and there is no voltage drop across the resistor. When the slide switch is moved to the "on" position, the input to the port bit is grounded (0 V), corresponding to a digital logic LOW. These switches will be connected to the microcontrollers Port 2 and Port 3.

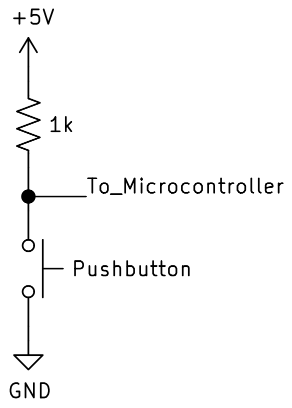

Figure 1: Pushbutton Schematic

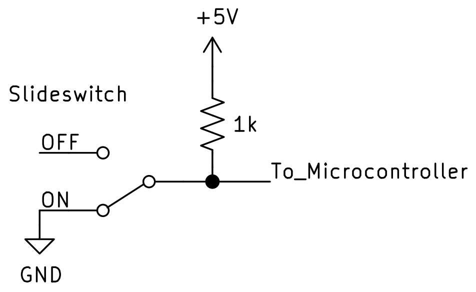

Figure 2: Slideswitch Schematic

The output signals from the C8051 control the motor, the LED, and the Bi-color LED, as shown in Figure 3. Similar to the description for input signals, output voltages of 0 V and 5 V are used to turn these devices "on" and "off," respectively. It is necessary to study the schematics to determine the correlation between "on" and "off" states. It is important not to confuse these circuit voltages with the concepts of TRUE/FALSE used in the program. By studying the schematic, recognize that in order to turn the motor on, the output signal needs to be set LOW 0 V. Similarly, the output at for the LED must be a digital LOW 0 V to turn the LED on. A digital HIGH 5 V on these Port pins will turn the devices off. The control of a Bi-color LED requires 2 output bits. To turn the Bi-color LED OFF two possible output states are possible, both bits can be HIGH or both bits can be LOW. To turn it on with one of its colors, one bit must be HIGH, and the other must be LOW. To turn on the other color, the reverse is necessary. In the initialization routine of the software, the Port 2 and Port 3 bits need to be configured for the appropriate input and output states.

The circuit for this laboratory is shown in the simulation window and shares portions of the schematics as shown here. In order to determine connections to the microcontroller from the circuit in the simulator, ***you must run the template code with your RIN at least once.*** Once run, labels will appear on the inputs/outputs of the circuit.

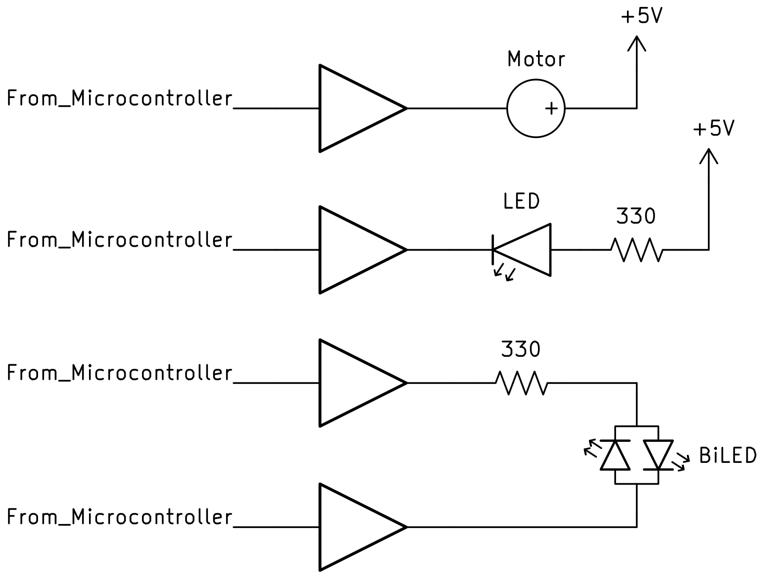

Figure 3: Output Schematics

Software

Software should be written such that the states of the inputs (pushbuttons and slide switch) are checked and the outputs (LED, Bi-color LED, Motor) are adjust appropriately. The code should first initialize the correct GPIO port pins to be either outputs or inputs and then enter an infinite loop which continually performs the checks and adjustments and prints out the states of the inputs. The GPIO port pins associated with each of the inputs and outputs is determined by the Simulator[1].

The following conditions should be implemented within the software:

1. The slide switch is "off": all outputs are off.
2. The slide switch is "on": the LED is on and ...
   (a) Pushbutton 1 is pressed: Bi-color LED is green
   (b) Pushbutton 2 is pressed: Bi-color LED is red
   (c) Both Pushbuttons are pressed: Bi-color LED is green and the motor is on.

The software should not modify any GPIO port pins if they are not used by this laboratory. This requirement implies that the initialization of the ports must be done using bit-masking. Additionally, the code must use "sbits"[2] in order to interact with the inputs and outputs as opposed to direct SFR writes. Finally, the code must be written in a modular fashion; that is, segments of code that relate to a specific functionality should be placed in separate functions (e.g., port initializations) and called from the main loop.

[1]Code must be run once with the RIN filled in at the top in order to populate the port pin assignments.

[2]"sbits" are only available in the actual development environment for the C8051. Here, the behavior is emulated using `#define` statements.

→ Laboratory Success Conditions

- Successfully implement the output controls as described in the previous section

→ General Lab Guidance

- Template code is provided for this lab on Piazza
- It is strongly suggested that this lab be written in two parts:

Step 1 : Implement the port initialization and then enter an infinite while loop where the inputs are checked and the outputs are set to a known value. If all seems correct, switch the output values to ensure correct output control functionality.

Step 2 : Add output control via the inputs as required.

- Ensure that your RIN is added at the top of your code.

```
#define RIN  xxxxxxxxx  // Must be above #include"C8051_SIM.h"
```

- Remember to run the template code once with just your RIN entered to get the circuit-to-microcontroller connections.
- Debugging of both hardware and software generally takes the bulk of the development time. In line with the suggestion above, it is always a good idea to write/test/write/test, etc. That is, write a small portion of code, ensure it is correct, then move on to the next. A similar method should be used when implementing hardware.
- Pseudocode for the lab is provided in the next section document on Piazza. A discussion of pseudocode is given in **PseudocodeSpecs.pdf** on Piazza.
- What you want a code to do must be *explicitly* specified. Not turning something on doesn't necessarily mean it gets turned off!
- There are two pushbuttons in the simulator, but they are not labeled Pushbutton 1 or 2. You may pick which one is which.
- A common issue in this lab that you will not have to deal with is conflicting output requests; for example, turning everything off each loop only to turn it on further on in the loop[3]. For LEDs, this would cause them to be dimmer than if they were turned on correctly. You should
- Don't forget to call `Sim_Update()` in each loop!

[3]This isn't an issue in the simulated version of the lab as the microcontroller only updates after `Sim_Update()` is called. Because of this, the off then on behavior is never applied to the simulation

→ Pseudocode

```
declare global variables
       sbit PB1, PB2, SS, LED0, BILED0, MOTOR
function prototypes
       void Port_Init(void)
       void Set_Outputs(void)
main function
       declare local variables
              (NONE)
       initialization functions
              Sys_Init();
              putchar(' ');
              Port_Init();
       Begin infinite loop
              execute Set_Outputs(void) function to read sbit inputs
                     and set sbit outputs
       End infinite loop
End main function

Functions

void Port_Init(void)
       Set SFRs such that inputs and outputs are configured correctly
       End Port_Init

void Set_Outputs(void)
       If SS is off then
              Configure State Appropriately
              Print "Slide Switch is OFF"
       Else (this means SS is on)
              Configure State Appropriately
              Print "Slide Switch is ON"
              If (PB1 is pushed and PB2 is pushed) then
                     Configure State Appropriately
                     Print "Pushbutton 1 and 2 ACTIVATED"
              Else if (PB1 is pushed and PB2 is released) then
                     Configure State Appropriately
                     Print "Pushbutton 1 ACTIVATED"
              Else if (PB1 is released and PB2 is pushed) then
                     Configure State Appropriately
                     Print "Pushbutton 2 ACTIVATED"
              Else Configure State Appropriately
End Set_Outputs
```