ECE/CSE 474:
Intro to Embedded Systems

Lab 1: GPIO and FSMs

Introduction

In this first lab, you will be investigating the General-Purpose In/Out (GPIO) peripheral on the TIVA LaunchPad using the IAR Embedded Workbench Integrated Development Environment (IDE). Moreover, you will investigate how Finite-State Machines (FSMs) are implemented in a software environment such as C.

By the end of this lab, you will:

- Be familiar with developing software using IAR Embedded Workbench
- Have experience programming the TM4C1294XL board using C code
- Understand the basics of GPIO and its digital applications
- Have experience extracting important information from datasheets
- Design an FSM controller in C

TM4C1294XL board

Figure 1 gives an overview of the layout of the TIVA TM4C1294XL LaunchPad. Both the user manual and the datasheet (found on Canvas -> Files -> Labs -> Datasheets) have much more information about the features and functionality of the board.

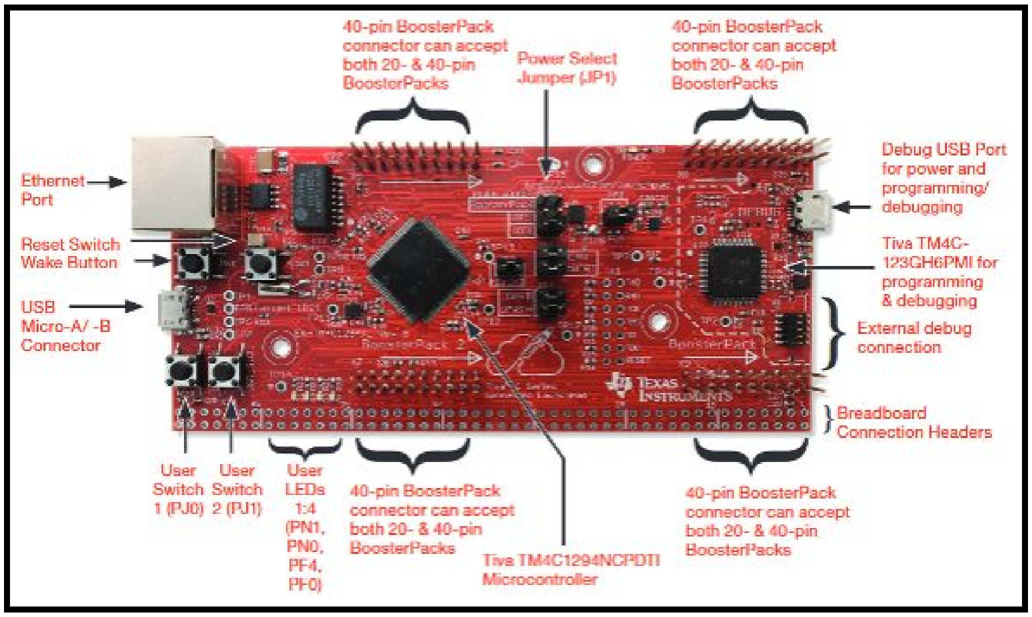

40-pin BoosterPack connector can accept both 20- & 40-pin BoosterPacks
Power Select Jumper (JP1)
40-pin BoosterPack connector can accept both 20- & 40-pin BoosterPacks
Ethernet Port
Debug USB Port for power and programming/ debugging
Reset Switch
Wake Button
Tiva TM4C-123GH6PMI for programming & debugging
USB Micro-A/ -B Connector
External debug connection
Breadboard Connection Headers
User Switch 1 (PJ0)
User Switch 2 (PJ1)
User LEDs 1:4 (PN1, PN0, PF4, PF0)
40-pin BoosterPack connector can accept both 20- & 40-pin BoosterPacks
Tiva TM4C1294NCPDTI Microcontroller
40-pin BoosterPack connector can accept both 20- & 40-pin BoosterPacks

Figure 1. Board Layout of the TIVA TC4M1294XL LaunchPad **[1]**

Three Steps to Get You Started

Step 1: Installing IAR

First, you need to install IAR Embedded Workbench. Follow the **"IAR Installation Tutorial"** under Canvas -> Files -> Labs -> Hardware and Software to fully install the IDE. Next, follow the **"IAR New Project Tutorial"** found in the same folder to initialize a new project for this assignment.

Step 2: Turning on an LED

The next step is to turn on a green LED provided on the TIVA evaluation board. Figure 2 shows the port connections for the four on-board LEDs. LED1 (labeled as "D1") is accessible to Pin 1 of GPIO Port N (labeled as "PN1"). As such, LED2 (D2) is accessible to PN0. To turn an LED on, proper registers associated with the GPIO peripheral need to be configured, and these registers are located in the memory of the ARM Cortex M-4 on the development board. We can find information about all of the registers and their functionality from the TM4C1294NCPDT datasheet. This datasheet will be very useful in this course, so it's a good idea to familiarize yourself with it.

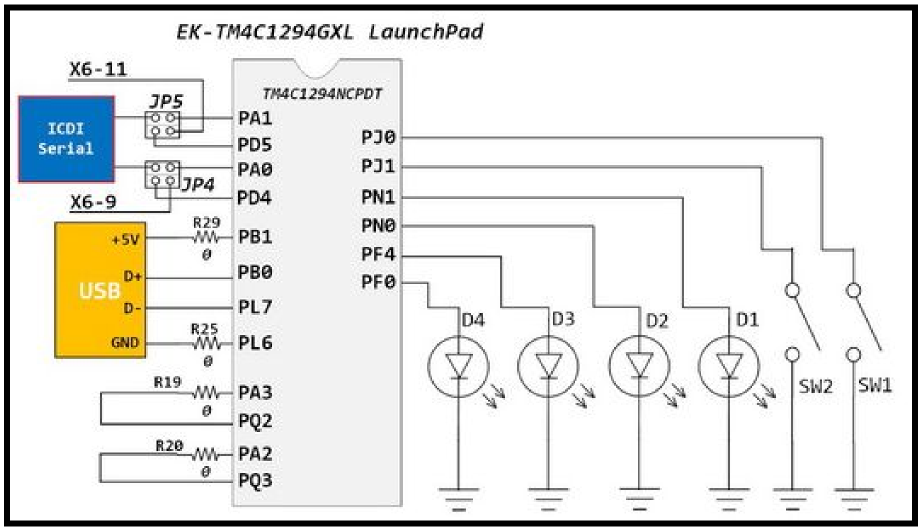


Figure 2. On-board LED and switch layout on the TM4C1294XL LaunchPad ***[2]***

To get you started with programming the microcontroller, you are tasked with turning on LED4. The steps that follow explain the process of turning on LED4:

1. Configure the **RCGCGPIO** register to turn on the GPIO Port F.
   - You may notice, by looking up the **RCGCGPIO** register in the datasheet, that this register can turn on or off any GPIO port. It does so by enabling or disabling a connection to the clock that enables functionality. By default, the clock to all

peripherals, including GPIO ports, are turned off for the sake of low power consumption.

2. Configure the **GPIODEN** register to set Pin 0 of Port F(PF0) as a digital input/output
   - None of the GPIO pins are set to be digital inputs/outputs by default.
3. Configure the **GPIODIR** register to set the direction of the PF0 pin to be output.
4. Configure the **GPIODATA** register to set the **output value** of PF0 to 1, thus turning on LED4. To do this, the least-significant bit (LSB) must be set true, which can be done by bit-masking the register with "0x1" (in hexadecimal) or "0b1" (in binary)
   - Note that since the GPIO pins are configured as outputs, the user **sets** the value of the data register to control the outputs. If the pins were instead configured as inputs, the user would **read** the incoming data from this register.

Here is the main.c file containing code that executes these steps and turns on LED 4:

```
#include <stdint.h>
#include "lab1.h"

int main(void)
{
    volatile unsigned short delay = 0;
    RCGCGPIO |= 0x20;  // Enable PortF GPIO
    delay++;           // Delay 2 more cycles before access Timer registers
    delay++;           // Refer to Page. 756 of Datasheet for info

    GPIODIR_F = 0x1;   // Set PF0 to output
    GPIODEN_F = 0x1;   // Set PF0 to digital port

    GPIODATA_F = 0x1;  // Set PF0 to 1

    while (1) {}

    return 0;
}
```

Here is the header file lab1.h that is included in the main.c file above:

```
#ifndef __HEADER1_H__
#define __HEADER1_H__

#define RCGCGPIO     (*((volatile uint32_t *)0x400FE608))
#define GPIODIR_F    (*((volatile uint32_t *)0x4005D400))
#define GPIODEN_F    (*((volatile uint32_t *)0x4005D51C))
#define GPIODATA_F   (*((volatile uint32_t *)0x4005D3FC))

#endif //__HEADER1_H__
```

Create the files in IAR Embedded Workbench by following section 2.4 of the **"IAR New Project Tutorial"** under Canvas -> Files -> Labs -> Hardware and Software and copy the code above into the corresponding files you created in your project. Download the program to your board and debug it. To do this, click the green "start" icon at the top of the screen, or press Ctrl + D. Then, hit the white "start" icon at the top of the screen, or press F5.

LED4 should light up.

Note: If the LED4 failed to light up, or the code failed to upload, please refer to the "TIVA Board Troubleshooting" file under Canvas -> Files -> Labs -> Hardware and Software.

Step 3: Interacting with User Switches

Reading from the user switches is similar to writing to the LEDs. The difference is you'll have to set the direction of their pins (PJ0 and PJ1) to be inputs using the **GPIODIR** register and attach pull-up resistors (user switches will be active low after attaching the pull-up resistors).

Note: To attach pull-up resistors to pins PJ0 and PJ1, you'll need to use the **GPIOPUR** register.

The datasheet has the information you need.

Assigned Task 1

Modify your existing project to achieve the following:

a. **Change the main program and the header file to turn on and off the LEDs in a periodic pattern.**
   Now that you can light up all four LEDs, write code that will turn on/off these LEDs in a sequential pattern. Consider sequentially turning on one LED at a time, followed by turning them off one at a time.

   Using loops, continuously change the displayed pattern of the LEDs. To create a delay between changing an LED, you can use a loop. The following code introduces a delay of approximately 0.3 seconds:

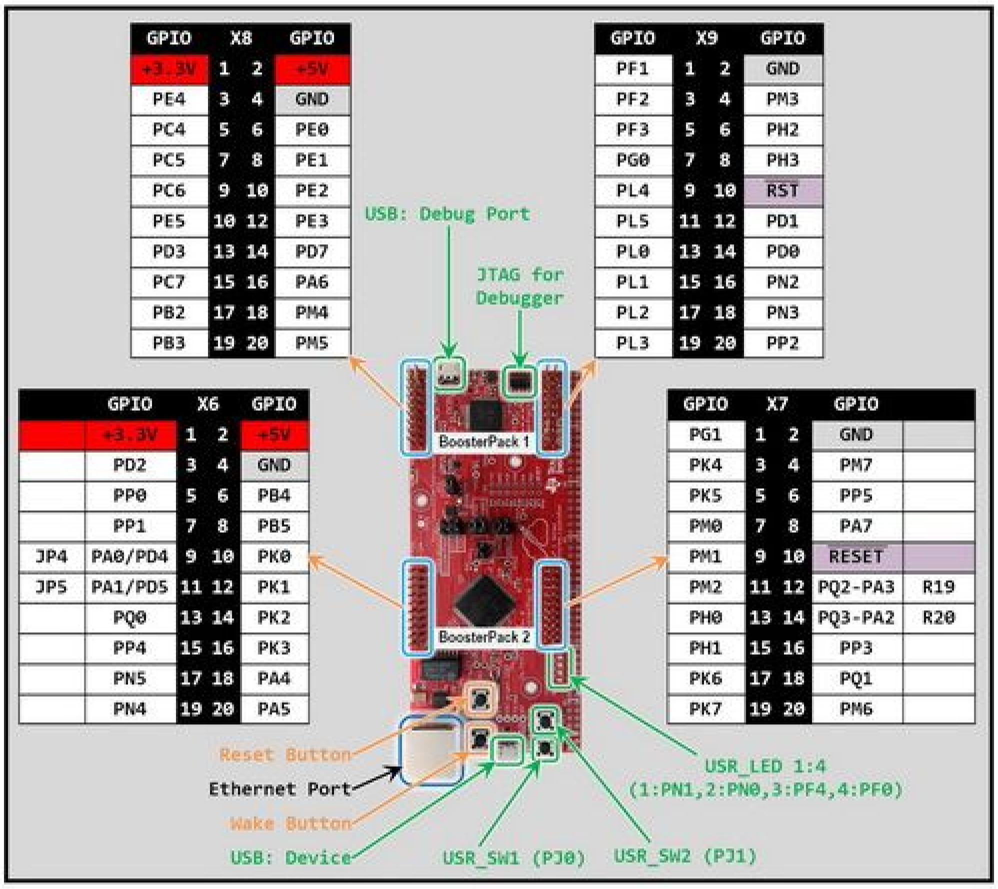
   ```
   for (j = 0; j < 1000000; j++) {}
   ```

b. **Create a new program and turn on the LED using the user switches.**
   Your new program should turn on the LED1 (D1) whenever switch 1 (SW1) is pressed, and turn on the LED2 (D2) whenever switch 2 (SW2) is pressed. The location of these switches can be seen in Figure 3. The GPIO pins associated with the LEDs and SWs can be found in Figure 2.

***Note:** Do not break down the circuit you design on your breadboard in task 2. You will be using this same circuit in future labs.

Assigned Task 2

For this task, you'll be designing a finite state machine that implements a traffic light system. To do this, you will use your TIVA launchpad, the provided external LEDs (Green, Red, and Yellow), and the pushbuttons. Since these are external to the board, you will need a breadboard and jumper wires. Note that Figure 3, an overview of the TM4C1294XL pin-outs, is extremely useful for this task. You should refer to it when configuring GPIO pins and connecting configured pins to your breadboard.



GPIO	X8		GPIO
+3.3V	1	2	+5V
PE4	3	4	GND
PC4	5	6	PE0
PC5	7	8	PE1
PC6	9	10	PE2
PE5	10	12	PE3
PD3	13	14	PD7
PC7	15	16	PA6
PB2	17	18	PM4
PB3	19	20	PM5

GPIO	X9		GPIO
PF1	1	2	GND
PF2	3	4	PM3
PF3	5	6	PH2
PG0	7	8	PH3
PL4	9	10	RST
PL5	11	12	PD1
PL0	13	14	PD0
PL1	15	16	PN2
PL2	17	18	PN3
PL3	19	20	PP2

	GPIO	X6		GPIO
	+3.3V	1	2	+5V
	PD2	3	4	GND
	PP0	5	6	PB4
	PP1	7	8	PB5
JP4	PA0/PD4	9	10	PK0
JP5	PA1/PD5	11	12	PK1
	PQ0	13	14	PK2
	PP4	15	16	PK3
	PN5	17	18	PA4
	PN4	19	20	PA5

GPIO	X7		GPIO	
PG1	1	2	GND	
PK4	3	4	PM7	
PK5	5	6	PP5	
PM0	7	8	PA7	
PM1	9	10	RESET	
PM2	11	12	PQ2-PA3	R19
PH0	13	14	PQ3-PA2	R20
PH1	15	16	PP3	
PK6	17	18	PQ1	
PK7	19	20	PM6	

Figure 3. TM4C1294XL pre-soldered pin-out, user buttons, and port (ethernet, USB, etc.) overview. The pre-soldered pins are grouped in sets, labeled by X#, where "#" is a number between 6 and 9. ***[2]***

External LEDs

In the first task, you used the onboard LEDs. Now, you'll need to use off-board peripheral LEDs. A green, yellow, and red LED can be found in your lab kit. You'll also find three $220\ \Omega$ resistors and a 74LS06 inverter chip. With these, you can create three copies of the circuit shown in Figure 4. This circuit will let you control an external LED by writing to the GPIO pin connected to the input of the inverter.

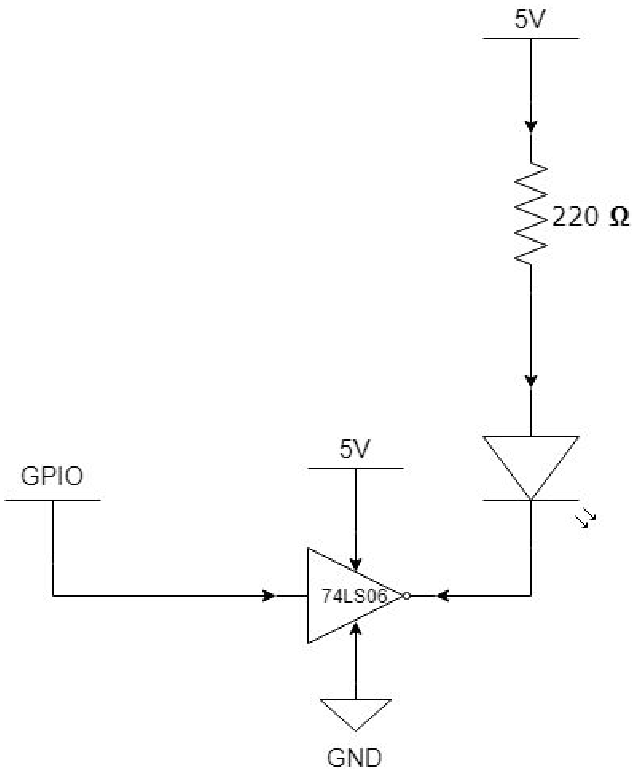


Figure 4. External LED circuit schematic

Recall that properly-working LEDs emit light when sufficient current passes through them. Moreover, current flow through the diode is negligible unless an "on-voltage" is supplied across the diode (specifically, from the anode to the cathode). For the sake of simplicity, we can say that the current will only flow from the anode (positive) to the cathode (negative). For LEDs, the polarity of a pin (positive or negative) is indicated by the length of the pins: the positive anode pin is always longer than the negative cathode pin. In the circuit schematic of Figure 4, the shorter leg of the diode should be connected to the output of the inverter chip, while the longer leg should be connected to the resistor.

External LED code

The following code snippet will enable you to turn on an LED when the signal marked “GPIO” in the above circuit schematic is connected to pin PC4:

```
void LED_init(void)
{
    volatile unsigned short delay = 0;
    RCGCGPIO |= 0x04;                // activate clock for Port C
    delay++;
    delay++;

    GPIOAMSEL_C &= ~0x10;            // disable analog function of PC4
    GPIODIR_C |= 0x10;               // set PC4 to output
    GPIOAFSEL_C &= ~0x10;            // set PC4 regular port function
    GPIODEN_C |= 0x10;               // enable digital output on PC4
}

// turn on LED connected to PC4
void LED_on(void)
{
    GPIODATA_C |= 0x10;
}

// turn off LED connected to PC4
void LED_off(void)
{
    GPIODATA_C &= ~0x10;
}
```

Here are the defines of the registers to be included in the header file. Note that if you are adding the defines to your existing header file, some registers may have already been defined (such as **RCGCGPIO**).

```
#define RCGCGPIO         (*((volatile uint32_t *)0x400FE608))

#define GPIOAMSEL_C      (*((volatile uint32_t *)0x4005A528))

#define GPIODIR_C        (*((volatile uint32_t *)0x4005A400))

#define GPIODEN_C        (*((volatile uint32_t *)0x4005A51C))

#define GPIOAFSEL_C      (*((volatile uint32_t *)0x4005A420))

#define GPIODATA_C       (*((volatile uint32_t *)0x4005A3FC))
```

You will need similar code to control all 3 of your LEDs.

External Buttons

Just like with the LEDs, you will use off-board buttons instead of the buttons on the board. Buttons are simple: when the button is not being pressed, the two ends are disconnected. Once pressed, the two ends are connected in a short circuit.

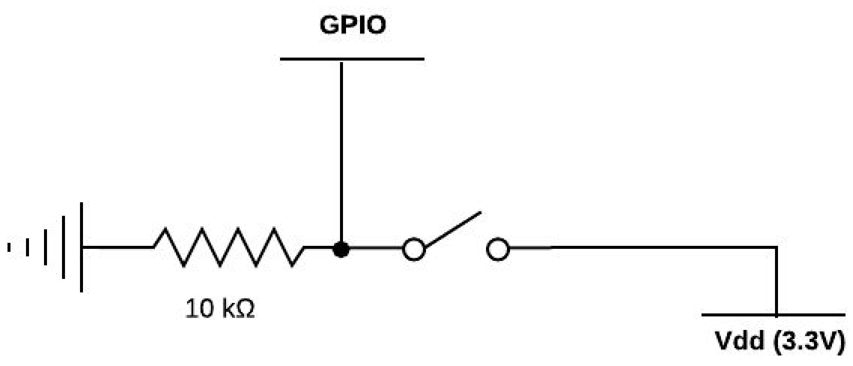


Figure 5. External button (switch) circuit schematic

To read from a button, connect one end to the 3.3V output of your TIVA board. Connect the other end to 1) the GPIO you'll be reading from, and 2) a resistor connected to the ground.

When the button is not being pressed, the GPIO will read LOW due to the connection with the resistor to the ground. When the button is pressed, the GPIO will read HIGH from the 3.3V output.

External Button Code

The following code snippet will enable you to read from a button connected (as described above) to pin PN2.

```
void extern_switch_init(void)
{
    volatile unsigned short delay = 0;
    RCGCGPIO |= 0x1000;   // Enable Port N Gating Clock
    delay++;
    delay++;
    GPIOAMSEL_N &= ~0x4;  // Disable PN2 analog function
    GPIOAFSEL_N &= ~0x4;  // Select PN2 regular port function
    GPIODIR_N &= ~0x4;    // Set PN2 to input direction
    GPIODEN_N |= 0x4;     // Enable PN2 digital function
}

unsigned long switch_input(void)
{
  return (GPIODATA_N & 0x4); // 0x4 (pressed) or 0 (not pressed)
}
```

Here are the defines of the registers to be included in the header file. Note that if you are adding the defines to your existing header file, some registers may have already been defined (such as **RCGCGPIO**).

```
#define RCGCGPIO         (*((volatile uint32_t *)0x400FE608))

#define GPIOAMSEL_N      (*((volatile uint32_t *)0x40064528))

#define GPIODIR_N        (*((volatile uint32_t *)0x40064400))

#define GPIODEN_N        (*((volatile uint32_t *)0x4006451C))

#define GPIOAFSEL_N      (*((volatile uint32_t *)0x40064420))

#define GPIODATA_N       (*((volatile uint32_t *)0x400643FC))
```

You will need code like this to read inputs from two separate buttons.

Important: the pins on this board (like many other microcontroller boards) are **NOT** 5V tolerant. This means that they will be **destroyed** if you connect them to any 5V output (or, specifically, anything higher than ~3.6V). This means that connecting the 5V VBUS pin to any other pin **will destroy that pin**.

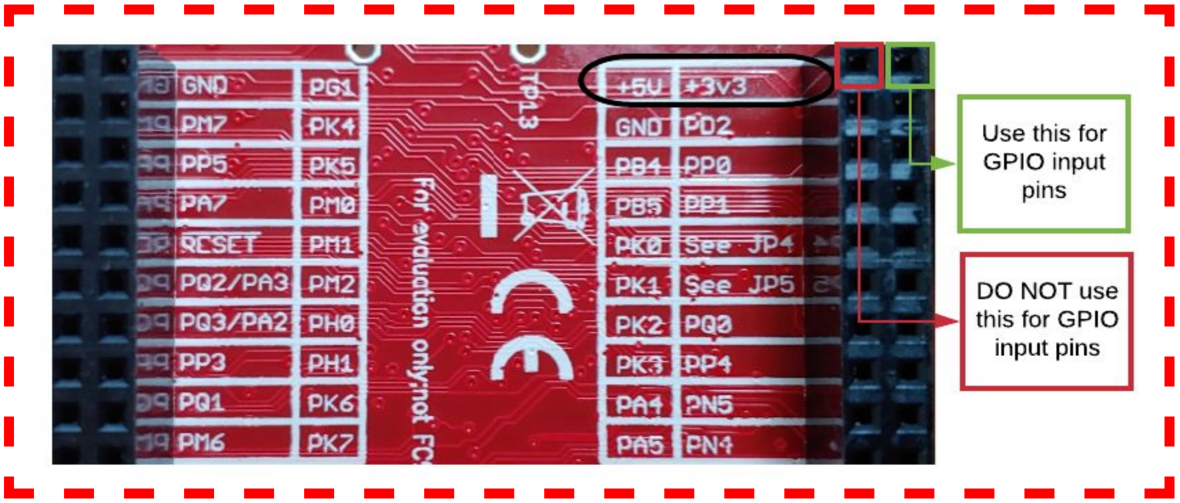


Figure 6. Displays male-pin accessible pin-out of the booster pack 2 pin set (the pin set closest to the ethernet port). Please make sure to only connect the output of the 3v3 pin to any GPIO pin.

To implement your task 2, you need to use two buttons and three LEDs. All accessible pins are listed in Figure 3. The following table lists some suggested pins to use to connect all of these components:

Table 1: Pin Suggestions for Task 2

LEDs/Buttons	**Suggestion Set 1**	**Suggestion Set 2**	**Suggestion Set 3**
System On/Off Button	PL0	PE0	PM0
Pedestrian Button	PL1	PE1	PM1
Red LED (Stop State)	PL2	PE2	PM2
Yellow LED (Warn State)	PL3	PE3	PM6
Green LED (Go State)	PL4	PE5	PM7

Traffic Light Controller

Design a finite state machine representing the traffic light system described below. Draw a block diagram for the system and write the C code that will implement it on your TIVA LaunchPad. **Implement your finite state machine using external peripherals**

System inputs:

1. System On/Off button
   The user will press this button to turn on and off the system. When the system is turned on, it will always start in the ***stop*** state or the ***go*** state. When the system is turned off, all the LEDs should be turned off.
2. Pedestrian button
   The user will press this button to indicate that there is a pedestrian that would like to cross the street.
   - If the button is pressed in the ***stop*** state, nothing will happen.
   - If the button is pressed in the ***go*** state, the state should change to the ***warn*** state, and then change to the ***stop*** state.

System outputs:

1. Red LED
   This LED is on when (and only when) in the ***stop*** state
2. Yellow LED
   This LED is on when (and only when) in the ***warn*** state
3. Green LED
   This LED is on when (and only when) in the ***go*** state

When the system is turned on and the pedestrian button is not pressed, it should switch between the ***go*** state and the ***stop*** state.

Lab Demonstration and Submission Requirements

- Submit demo videos of Task1 and Task2 on their respective due dates. The videos should show the expected results described in Task 1 and Task 2 on the TIVA Board. Feel free to add narration or text in the video. Each video must be less than 90-seconds long. To receive full credits, your videos must thoroughly demonstrate all the required functionalities.
- Write a very brief Lab Report, as framed by the “Lab Report Guide” on Canvas -> Files -> Labs -> Lab Report and Code. Make sure to include a drawing of your FSM design for this lab.
- Revise the style of your Lab Code, as framed by the “Programming Style Guide” on Canvas -> Files -> Labs -> Lab Report and Code.
- Submit your Lab Report as a pdf, and submit your Lab Code (.c and .h files). Please do not submit compressed ZIP folders or other files such as the workspace files (.eww) and project files (.ewp). Before submitting, refer to the rubric shown on the “Report and Code” Canvas Assignment.
- On Padlet, write about a problem you had in the lab and the fix to it, and share a tip or trick you learned while working on the lab. You can also share an aha moment that you discovered while working on the lab. Avoid duplicating comments made by your classmates. NO videos for this Padlet task, please use textual comments. The link to the Padlet can be found on Canvas -> Assignments -> Lab1: Tips and Tricks. Please note that the Tips and Tricks assignment is part of the Community Participation, and thus the *Grace Period* does not apply.

References:

[1] User's Guide: Tiva™ C SeriesTM4C1294ConnectedLaunchPadEvaluationKit. Texas Instruments, 2016, p. 4. [Online]. Available: http://www.ti.com/lit/ug/spmu365c/spmu365c.pdf

[2] "Air Supply Lab - Air Supply Lab", *Airsupplylab.com*. [Online]. Available: http://www.airsupplylab.com/index.php?option=com_content&view=article&id=39&catid=22#on-board-i-o-2. [Accessed: 04- Apr- 2020].