ECE/CSE 474 Lab Report 1

Task 1a - Blink a LED

Procedure:

1. Create a new IAR workbench and copy and paste the code on the right for turning on LED 4.
2. Revise the provided main program and header file so that PN0, PN1, and PF4 were also enabled allowing LED 1, LED 2, and LED 3 to turn on.
3. Turned on or off LEDs by setting corresponding GPIODATA bits to 1 or 0.
4. Created a “wait” function to create a delay and looped the LEDs turning on and off with the wait function between each change. Shown on right of provided code.

```
#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
  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 t

```
#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__
```

```
  while (1) {
    // Turn on lights
    GPIODATA_F = 0x10;
    GPIODATA_N = 0x1;

    wait(400000);
    // Turn on different
    GPIODATA_F = 0x1;
    GPIODATA_N = 0x2;

    wait(400000);
  }
}

void wait(int delay) {
  // I cannot for the li
  int i = 0;
  while (i < delay) {
    i++;
  }
}
```

Task 1b - Turn LED on/off when a button is pressed

1. Set up switches by enabling pin J as done with LEDs but with the GPIODIR set to inputs and also enabling write permissions, GPIOCR, and pull-up resistors, GPIOPUR. Also, configure LEDs as outputs.
2. After setting up switches, configured switches inside while loop so that each button press would be read.
3. Set up if-else statement logic to allow each switch to light up certain LEDs.

```
GPIODIR_F = 0x11; // Set PF0 and PF4
GPIODEN_F = 0x11; // Set PF0 and PF4

// Do the same but for PN0 and PN1 (o
GPIODIR_N = 0x3;
GPIODEN_N = 0x3;

// Configure buttons
// Set direction to input
GPIODIR_J = 0xFC;
GPIODEN_J = 0x3;

// Set pull-up resistor since button
GPIOPUR_J = 0x3;
// Enable write permissions
GPIOCR_J = 0x3;

while (1) {
  // read PJ0 into SW1, PJ1 into SW2
  uint32_t SW1 = GPIODATA_J & 0x01;
  uint32_t SW2 = GPIODATA_J & 0x02;

  // Configure buttons - remember, no
  if (SW1 && SW2) {
    GPIODATA_F = 0x1;
    GPIODATA_N = 0x0;
  } else if (SW2) {
    GPIODATA_F = 0x10;
    GPIODATA_N = 0x0;
  } else if (SW1) {
    GPIODATA_N = 0x1;
    GPIODATA_F = 0x0;
  } else {
    GPIODATA_N = 0x2;
    GPIODATA_F = 0x0;
  }
}
```

Results:

For the first task we were able to utilize the LEDs built in the TIVA launchpad and light them up in an alternating pattern so imitate blinking lights. The LEDs would alternate with a small delay.

For the second task, we were able to utilize the buttons to light up the corresponding LEDs with each press. If SW1 and SW2 weren’t being pressed, LED 4 would be lit. If SW1 was pressed, LED 3 would light. If SW 2 was pressed, LED 2 would be lit. Finally, if both switches were pressed, LED 1 would light.

However, we misread the specifications. SW1 was supposed to light LED 1 and SW2 was supposed to light LED 2.

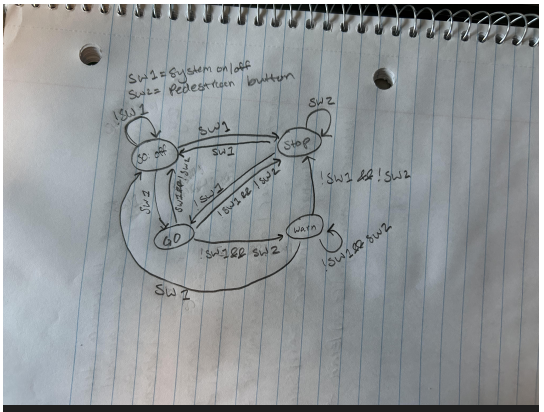
Task 2: Traffic light

Procedure:

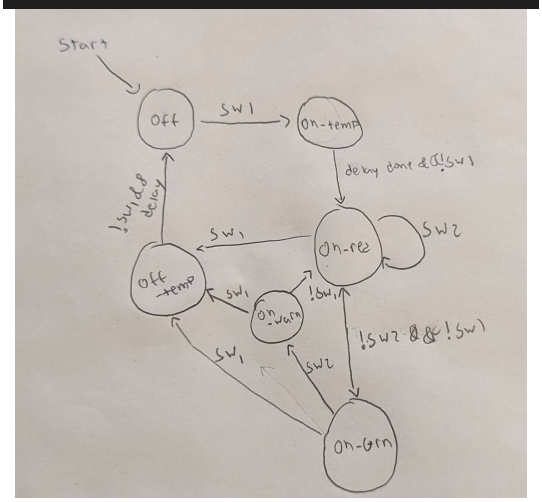
1. We first took a look at the requirements, which were as follows:
    a. The traffic light starts in the off state, it turns on/off whenever the on/off switch (SW1) is pressed.
    b. Then, the light oscillates between the stop (red) and go (green) states. If the user presses the pedestrian button (SW2) when in the go state, then the light turns to the warn (yellow) state, then to the stopstate. Pressing the pedestrian button in the stop state does nothing.
2. We then made a Finite State Machine (on the right) to document our plan for the code. We planned for 4 possible states, and easy simple button presses to go through each one
3. In our header file, we put our finite state machine's states, and a lot of the configs for external switches. We also included some relevant helper method prototypes, as well as a bunch of macros for the light sand the switches.. See right for our header file.
4. In our code file, we implemented all the helper methods and the init/begin methods.
5. Our code consisted of detecting the state the code was in, and turning on certain buttons or creating a delay as needed. Each light, connected to a different port, had a different macro. Since all our switches were also connected to the same port, we had to carefully manage macro usage to make sure the state for each switch is preserved. To do that, we used this code below:

```
void set_light(int color) {
  GPIODATA_L |= color;
  GPIODATA_L &= (color | 0x3);
}
```

6. As we implemented our code, we realized that we didn't have a button debounce time. As a result, button signals would still fluctuate a bit even when a button was released. This caused our initial FSM code to malfunction, thinking a phantom button was pressed. As a result, we added temporary "waiting" states, which led to the FSM drawn to the right:




```
// Possible states for the finite state machine
enum states{off_off, off_temp, on_temp, on_red, on_warn, on_green} curState;

// Base registers
#define BASEREG  (*((volatile uint32_t *)0x400FE000))
#define GPIOPUR  (*((volatile uint32_t *)0x400FE510))
#define RCGCGPIO (*((volatile uint32_t *)0x400FE608))

// Task 2 LED Configs
#define GPIOAMSEL_L (*((volatile uint32_t *)0x40062528))
#define GPIODIR_L   (*((volatile uint32_t *)0x40062400))
#define GPIODEN_L   (*((volatile uint32_t *)0x4006251C))
#define GPIOAFSEL_L (*((volatile uint32_t *)0x40062420))
#define GPIODATA_L  (*((volatile uint32_t *)0x400623FC))
#define GPIOPDR_L   (*((volatile uint32_t *)0x40062514))

// Switch states
#define SW1 0x1
#define SW2 0x2

// Light colors
#define OFF 0x0
#define RED 0x4
#define YLW 0x8
#define GRN 0x10

// Sets LED lights configured to the correct macro defined above in header files
void set_light(int color);

// Gets the input of the switch according to macro defined in header
unsigned long switch_input(int iswitch);

// Initializes the board, sets registers and starting state
void init();

// Starts the FSM
void begin();

// Waits for x time
void wait(int time);

// Waits for x time, returns 1 if sw1 is pressed
// If we aren't yellow and SW2 is pressed, returns 2
// Returns 0 (false) otherwise.
int wait_on(int time, int isYellow);
```



Results:

We built a system that has 2 buttons and 3 lights, a red, yellow, and green light. When the user presses the on/off switch (SW1), the system turns on or off. Whenever SW1 is pressed, based on the current state (on or off), the system will go to a short “debounce” state, where the system will create a short delay whenever the button is released to allow for the button to debounce. Lights will be off during these states and during the off state.

When the system turns on, it starts in the stop state, which turns a red light on. The system will enter the go state, keeping on only a single green light after around 3-5 seconds, during which a press of the on/off button will turn off the system.

When the system is green, it will naturally return to the off state in 3-5 seconds, and if the pedestrian switch (SW2) was pressed, then the machine will switch to the warn state, turning on a yellow light for around 2-3 seconds. Then the system will turn to the red state, and the sequence will repeat.