**I. Introduction**

In this lab we looked at example assembly code and observed the behavior on the board through debugging. Our task was then to explain the behavior in reference to the code (and specific lines). Moving forward, these tasks will be the fundamentals of the process when attempting larger embedded systems so it’s important to have these basics down.

**II. Procedure/Discussion**

For the example code testing, code was simply imported into Code Composer then debugged. Behavior on the board was observed then attributed with an explanation to specific lines of code. The numbered list below describes these observations in detail. For the second part involving the red LED program, the InputOutput example program was used as the foundation. The 2 button logic was adjusted to match the desired LED logic in the lab instructions. A port 1 input read subroutine was added for the LED, but writing to P1OUT needed to be done carefully so as to not set the resistors for the buttons as pull downs.

1. GPIO: There is no activity on the board as the ports 4.3 through 4.0 are not LEDs as the comments in the code would suggest. Pins 4.3-4.0 is set to be an output after being configured for GPIO. Then it outputs only to pins 4.3 and 4.1, then 4.3 and 4.0, then 4.2 and 4.0, then finally 4.2 and 4.1. This occurs in the (ORR R0, R0, #10) lines. The register that contains the output for the port pins is cleared with (BIC R0, R0, #0x0F) line.
2. InputOutput: At the start of the main function, Port 1 and 2 are initialized and the switch pins P1.1 and P1.4 are set as inputs and pins P2.2-P2.0 are set as outputs. In the main loop the switch pins are read in and checked for activation. If switch 1 is pressed, the blue LED turns on. For switch 2, the red LED activates and if both are pressed, the green LED lights up. If no switches are pressed then no LED lights up. If any other case occurs, then all LEDs turn on. This logic is achieved using CMP lines (CMP R0, #0x10 to check switch 1 for example). After the switch check the program will jump to a subroutine that will activate the appropriate LED pin to that case by setting the appropriate bits in P2OUT (MOV R0, #BLUE/#RED/#GREEN).
3. SquareWave: Pins P2.1 and P2.2 are made outputs after being configured as GPIO. The drive strength for these pins are also set high in the initialization. Initially the Green LED(2.1) is set as active (ORR R0, R0, #0x02), and the blue LED is set as off (BIC R0, R0, #0x04). Then in the main loop the LEDs alternate being active by toggling their bits every loop (EOR R0, R0, #0x06).
4. SSR: Pins P1.4 and P1.1 are configured as input buttons in the standard initialization ritual and are set as pull-up. In the main function it waits for button 1 to be pressed, otherwise it loops in a standby sort of state. Once button 1 is pressed it enters the next loop that waits for the button to be released. When released the SSR turns on (subroutine is called, ORR R0, R0, #0x04), then another stage of the main loop starts looping until button 2 is pressed. A similar situation occurs where the program moves on to another loop until the button is released, then the program jumps back to the top of the loop where it hits a subroutine that turns off the SSR (BIC R0, R0, #0x04). In this initial stage of the loop that subroutine is constantly called (BL SSR\_Off).
5. Switch: P1.1 and P1.4 are configured as pins and P2.2-P2.0 are configured as outputs for the buttons and RBG LEDs respectively. In the main function loop, the RGB LEDs are turned off at the top into a subroutine that reads the inputs. The condition checks are similar to the InputOutput program, as pressing switch 1 will turn on the blue LED, switch 2 the red LED, both for the green LED, and if none no LEDs turn on. Unknown cases will turn on all the LEDs. After reading the buttons, the main function jumps to the continue section. This section reads the switch at P1.5 and shifts its status into the P1.0 bit (LSR R0, R0, #5). Then LED1 is cleared (BIC R6, R6, #LED1), and the shifted switch bit are put into LED1’s pin (ORR R6,R6, R0).

**III. Conclusion**

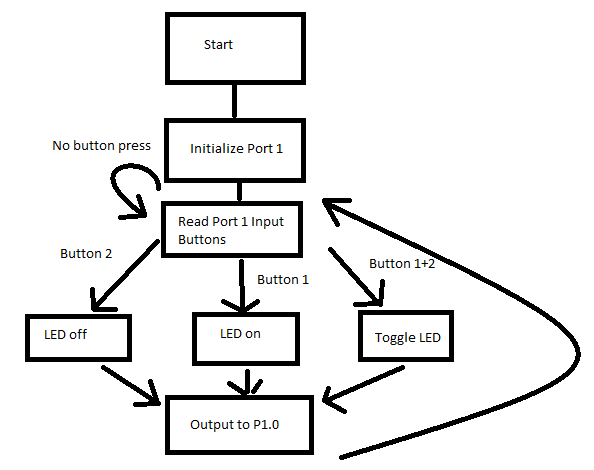
The final result of this lab was satisfactory, the demonstration went without complications and aside from funny business with the port 1 output, the part 2 program worked. Obviously the goal of the lab was to have a program that worked as per the given instructions, and this was a basic task which will most definitely be used in higher level systems.

The main thing learned in this lab is the balancing act of outputting to the port 1 LED without flipping the button resistors. If I could redo this lab it would’ve been convenient to know that the resistors being flipped to pull downs was causing the strange logic bug.

**References:**

Lecture 06-IntroIO-450 (on Titanium)

**Appendix:**



Pseudocode:

Main:

-Call Port1 initialization subroutine

Loop:

-Call port1 input read subroutine

-compare button inputs (loaded to register 0) to values indicating cases

-Jump to appropriate subroutine

Switch 1 pressed:

-set bit 0 of P1OUT to 1

Switch 2 pressed:

-set bit 0 of P1OUT to 0

Both switches pressed:

-toggle bit 0 of P1OUT

No button press:

-no change to bits

-Jump back to loop

Start of Code:

; InputOutput.s

; Runs on MSP432

; Test the GPIO initialization functions by setting the LED

; color according to the status of the switches.

; Daniel Valvano

; June 20, 2015

; This example accompanies the book

; "Embedded Systems: Introduction to the MSP432 Microcontroller",

; ISBN: 978-1512185676, Jonathan Valvano, copyright (c) 2015

; Section 4.2 Program 4.1

;

;Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu

; You may use, edit, run or distribute this file

; as long as the above copyright notice remains

;THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED

;OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF

;MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.

;VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,

;OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

;For more information about my classes, my research, and my books, see

;http://users.ece.utexas.edu/~valvano/

; built-in LED1 connected to P1.0

; negative logic built-in Button 1 connected to P1.1

; negative logic built-in Button 2 connected to P1.4

; built-in red LED connected to P2.0

; built-in green LED connected to P2.1

; built-in blue LED connected to P2.2

.thumb

**.text**

**.align** 2

P1IN .field 0x40004C00,32 ; Port 1 Input

P2IN .field 0x40004C01,32 ; Port 2 Input

P2OUT .field 0x40004C03,32 ; Port 2 Output

P1OUT .field 0x40004C02,32 ; Port 1 Output

P1DIR .field 0x40004C04,32 ; Port 1 Direction

P2DIR .field 0x40004C05,32 ; Port 2 Direction

P1REN .field 0x40004C06,32 ; Port 1 Resistor Enable

P2REN .field 0x40004C07,32 ; Port 2 Resistor Enable

P1DS .field 0x40004C08,32 ; Port 1 Drive Strength

P2DS .field 0x40004C09,32 ; Port 2 Drive Strength

P1SEL0 .field 0x40004C0A,32 ; Port 1 Select 0

P2SEL0 .field 0x40004C0B,32 ; Port 2 Select 0

P1SEL1 .field 0x40004C0C,32 ; Port 1 Select 1

P2SEL1 .field 0x40004C0D,32 ; Port 2 Select 1

;RED .equ 0x01

;GREEN .equ 0x02

;BLUE .equ 0x04

SW1 .equ 0x02 ; on the left side of the LaunchPad board

SW2 .equ 0x10 ; on the right side of the LaunchPad board

LED .equ 0x01

**.global** main

.thumbfunc main

**main:** .asmfunc

BL Port1\_Init ; initialize P1.1 and P1.4 and make them inputs (P1.1 and P1.4 built-in buttons)

; BL Port2\_Init ; initialize P2.2-P2.0 and make them outputs (P2.2-P2.0 built-in LEDs)

loop

BL Port1\_Input ; read both of the switches on Port 1

**CMP** R0, #0x10 ; R0 == 0x10?

BEQ sw1pressed ; if so, switch 1 pressed

**CMP** R0, #0x02 ; R0 == 0x02?

BEQ sw2pressed ; if so, switch 2 pressed

**CMP** R0, #0x00 ; R0 == 0x00?

BEQ bothpressed ; if so, both switches pressed

**CMP** R0, #0x12 ; R0 == 0x12?

BEQ nopressed ; if so, neither switch pressed

; if none of the above, unexpected return value

;MOV R0, #(RED+GREEN+BLUE) ; R0 = (RED|GREEN|BLUE) (all LEDs on) ; turn all of the LEDs on

**B** loop

sw1pressed

ORR R0, #0x01

BL Port1\_Output

**B** loop

sw2pressed

BIC R0, #0x01

BL Port1\_Output ; turn the red LED on

**B** loop

bothpressed

LDR R1, P1OUT

LDRB R0, [R1]

EOR R0, #0x01

BL Port1\_Output ; turn the green LED on

Wait

BL Port1\_Input

**CMP** R0, #0x12

BEQ loop

**B** Wait

nopressed

;MOV R0, #0 ; R0 = 0 (no LEDs on)

;BL Port2\_Output

LDR R1, P1OUT

LDRB R0, [R1]

BL Port1\_Output ; turn all of the LEDs off

**B** loop

.endasmfunc

;------------Port1\_Init------------

; Initialize GPIO Port 1 for negative logic switches on P1.1 and

; P1.4 as the LaunchPad is wired. Weak internal pull-up

; resistors are enabled.

; Input: none

; Output: none

; Modifies: R0, R1

**Port1\_Init:** .asmfunc

LDR R1, P1SEL0

**MOV** R0, #0x00 ; configure P1.4 and P1.1 as GPIO

STRB R0, [R1]

LDR R1, P1SEL1

**MOV** R0, #0x00 ; configure P1.4 and P1.1 as GPIO

STRB R0, [R1]

LDR R1, P1DIR

**MOV** R0, #0x01 ; make P1.4 and P1.1 inputs (and P1.0 as output)

STRB R0, [R1]

LDR R1, P1REN

**MOV** R0, #0x12 ; enable pull resistors on P1.4 and P1.1

STRB R0, [R1]

LDR R1, P1OUT

ORR R0, #0x12 ; P1.4 and P1.1 are pull-up

STRB R0, [R1]

BX LR

.endasmfunc

;------------Port1\_Input------------

; Read and return the status of the switches.

; Input: none

; Output: R0 0x10 if only Switch 1 is pressed

; R0 0x02 if only Switch 2 is pressed

; R0 0x00 if both switches are pressed

; R0 0x12 if no switches are pressed

; Modifies: R1

**Port1\_Input:** .asmfunc

LDR R1, P1IN

LDRB R0, [R1] ; read all 8 bits of Port 1

**AND** R0, R0, #0x12 ; select the input pins P1.1 and P1.4

BX LR

.endasmfunc

;---------------------------------------

**Port1\_Output:** .asmfunc

LDR R1, P1OUT

ORR R0, #0x12

STRB R0, [R1]

BX LR

.endasmfunc

;------------Port2\_Init------------

; Initialize GPIO Port 2 red, green, and blue LEDs as

; the LaunchPad is wired.

; Input: none

; Output: none

; Modifies: R0, R1

;Port2\_Init: .asmfunc

; LDR R1, P2SEL0

; MOV R0, #0x00 ; configure P2.2-P2.0 as GPIO

; STRB R0, [R1]

; LDR R1, P2SEL1

; MOV R0, #0x00 ; configure P2.2-P2.0 as GPIO

; STRB R0, [R1]

; LDR R1, P2DS

; MOV R0, #0x07 ; make P2.2-P2.0 high drive strength

;STRB R0, [R1]

;LDR R1, P2DIR

;MOV R0, #0x07 ; make P2.2-P2.0 out

;STRB R0, [R1]

;LDR R1, P2OUT

;MOV R0, #0x00 ; all LEDs off

;STRB R0, [R1]

;BX LR

;.endasmfunc

;------------Port2\_Output------------

; Set the output state of P2.

; Input: R0 new state of P2 (only 8 least significant bits)

; Output: none

; Modifies: R1

;Port2\_Output: .asmfunc

; LDR R1, P2OUT

; STRB R0, [R1] ; write to P2.7-P2.0

; BX LR

; .endasmfunc

.end