EGCP 450-01 Lab 2: External I/O in ARM Assembly

James Samawi – 9/14/2020

Introduction

In this lab, I worked with GPIO ports to create an external circuit that toggles an LED with the press of a switch. I also added a delay subroutine that temporarily delays the toggling of the LED by 250ms (or a quarter of a second). Learning how to work with GPIO ports is applicable for future sophisticated projects because it broadens the horizon of embedded systems such that it enables features that are outside of the host system/microcontroller.

Procedure/Discussion

I will split this lab into two parts: the first is getting the LED to toggle when the switch is pressed (and powered on when the switch is not pressed), and the second being the addition of the delay subroutine. I will begin discussing the structure of part one and the physical construction of the external circuit. For this program, the “main” subroutine will be used only for redirecting the Link Register (LR). This is a method used in my EGCP 280 class, LR is pushed into the stack and with the help of PC it is able to branch to the specified subroutines and will return to the original location in main when finished (LR will be popped from stack at the end of main). Main will execute “Port4_Init” (GPIO pin P4.0 dedicated to external LED) and “Port5_Init” (GPIO pin 5.0 dedicated to external switch) subroutines, and with it, the P4OUT will be enabled initially as instructed. After that is done, LR is finally directed towards the “Loop” subroutine where the program will repeatedly loop. Putting the delay loop on the side for now, I started by implementing the “Switch_Input” subroutine from the previous lab to gather the input data from the user in the beginning of Loop. Afterwards, the program starts comparing R0 (the register that stored the value of P1IN from R1) to its two possible values: 0x0 when not pressed, and 0x1 when pressed. Under “pressed”, the program branches to the “LED_Toggle” subroutine (uses EOR to toggle LED on/off) and exits back to Loop. Under “not_pressed”, the program branches to the “LED_On” subroutine (uses ORR to power LED) and exits back to Loop. As for the physical construction of the external circuit, I mainly followed the image of the circuit/schematic in the Appendix section. I say mainly because I did not have the 470 Ohm resistor, instead I used a 1k-Ohm resistor which worked fine, the dimming of the LED was not a problem. Everything else was implemented and the result of this lab was a success.

The second part of this lab, the delay loop, was a necessary addition to this program because the toggling of the LED when pressing the switch was not noticeable without the loop. I approached this problem mathematically at first, and then I used brute force/trial and error later on. Since the processor of the MSP432P401R runs at around 3MHz, the target period is the reciprocal of that. That is the first known variable, the second known variable is the time to execute which is 250ms. The unknown variable is the clock cycle, which will be matched later within the CCS program.

$$T_{CYC} * T_S = Exec.\ Time$$

$$T_{CYC}\ (1 / 3MHz) = 250ms$$

$$T_{CYC} = 3{,}000{,}000 * 0.250s = 750{,}000\ cycles.$$

After acquiring this number of cycles, I was unsure where else to use this number to find the correct hexadecimal value to count down from within the delay loop. Refer to the following implemented ARM instructions below:

```
     MOV R0, #0x0D090
     ORR R0, R0, #0x30000
wait
     SUBS R0, R0, #0x01
     BNE wait
     BX   LR
```

The max number of bits the MOV instruction can handle is 16 bits. The simple solution to this restriction, as seen above, is to use the ORR instruction to add more numbers to the same register. This is a simple program that counts down from a large number repeatedly until the exit condition of “#0x01” is reached. To utilize the number of cycles we found previously, we must utilize CCS and the ability to count clock cycles upon halting on a breakpoint. I started by placing breakpoints on line 2 and line 6 in the above code. While debugging, the program halted on the first breakpoint and I made sure to reset the counter before proceeding. I started the program once more and then CCS determined how many clock cycles occurred since the start of the “wait” loop until the ending statement. I repeated this process multiple times and came across the hexadecimal number (#0x3D090) which took exactly 750,000 cycles to complete. I later learned that 0x3D090 was 250,000 in decimal, which means the faster method to find the unknown hex number was to take the execution time and multiply that by 1,000,000 (and then converting to hex). This delay loop was now placed at the beginning of the previously mentioned Loop subroutine.

- If I were to change the delay from 250ms to 0.5ms, I would start by calculating the new required clock cycles (3 million times 0.0005) which is 1,500 cycles. To find the hexadecimal number to count down from, that would be 0.0005 * 1 million = 500 and that is 0x1F4 in hex. Using the same CCS clock cycle event feature, I would verify the duration of the loop to be 1,500 cycles. The disadvantage of my delay loop is the variance when it comes to the branch-if-not-equal instruction (BNE) will vary between 1-4 cycles, so it won't exactly be 250ms. With SUBS, the minimum time is 250,000(1+1)/3MHz = 166ms and the max time is 250,000(1+4)/3MHz = 416ms. Maybe in future labs when we revisit Systick timer then there will be a more precise, simpler, advantageous method.
- How does the use of a pull-up or pull-down resistor relate to using a positive-logic or a negative-logic interface? In the context of switches, a positive-logic switch uses a pull-down resistor and a negative-logic switch uses a pull-up resistor. They are opposites, this is because one side that has a higher potential must have another side with a lower potential.

As for the problems I faced in this lab, there were two predominant issues. The first was a hardware issue, I managed to incorrectly place the LED on my breadboard. It was a rookie mistake; I placed the anode and the cathode on the same wire in the breadboard. Interestingly enough, this was the same issue I had in the previous lab when I tried running the "GPIO" program and it was the only program I couldn't display the results for. This gave me the most problem and it took a lot of software revisions to figure out the problem was outside. The next issue was that I forgot to update the new port 5 address offsets, and I wondered why R0 would not change when I pressed the switch while stepping through the code.

Conclusion

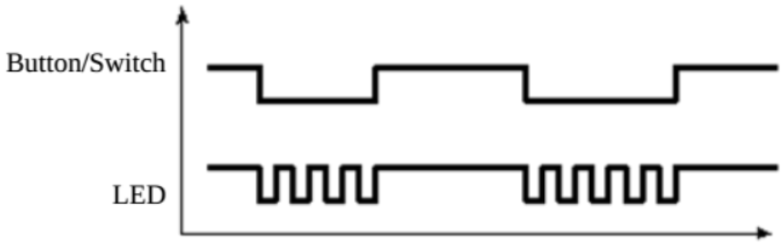
This lab taught me how to properly create an ARM assembly program to control an external device. I also learned how to utilize features on CCS that can count the clock cycles in your program. These two points are applicable to future large-scaled projects because embedded systems control a variety of devices and are not limited to the microcontroller itself. Also, it is important to get familiar with counting clock cycles and execution times in embedded systems since there are a large amount of time sensitive devices. This lab went well, the only thing I would change is putting more time into the delay loop, as there was possibly a more efficient method of delaying a program that did not have a large minimum and maximum runtime gap. I enjoyed this challenging lab and I look forward to the finite state machine lab next time.

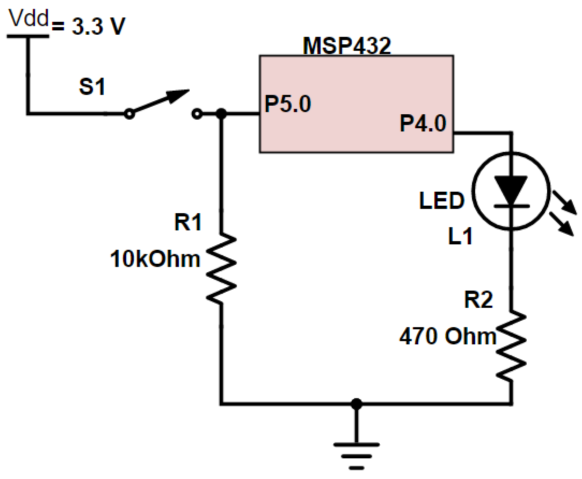
References

No other references besides the class book and lecture slides were used.

Appendix

Images:



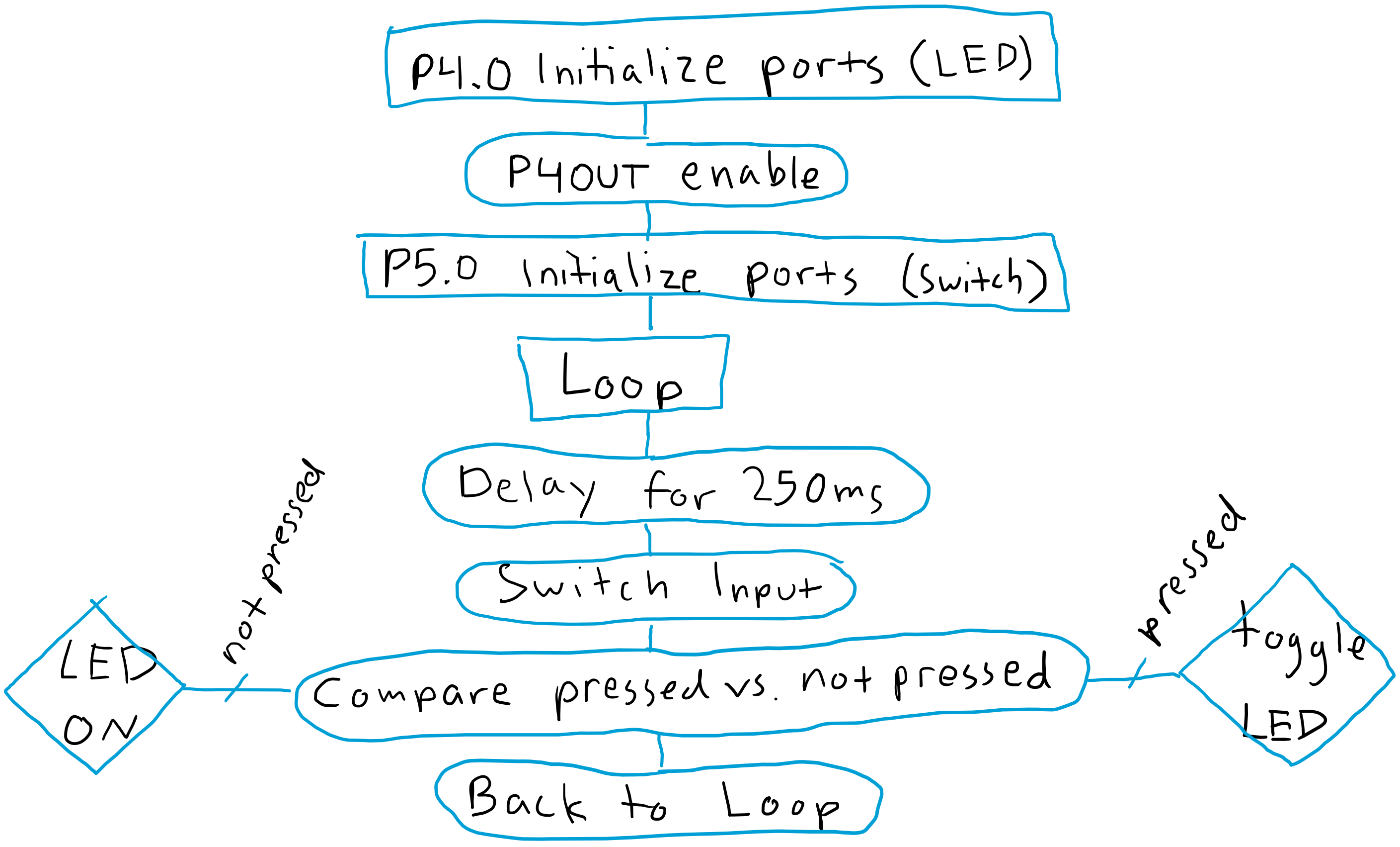


Pseudocode:

- Initialize port 4.0 for external LED output
    - Enable P4OUT so LED is initially on
- Initialize port 5.0 for external switch input

- Loop:
    - Branch to Delay to wait 250 ms
    - Branch to Switch Input to gather input data (0x0 or 0x1)
    - Compare and determine if press or no press
        - If P5IN == 0x1 (the switch is pressed), set P4.0 to toggle LED on and off repeatedly
        - If P5IN == 0x0 (the switch is not pressed), turn LED on by enabling P4OUT
    - Repeat loop

Flowchart:



Source code:

```
; LAB 2: External switch with 250 ms delay toggles external LED
; James Samawi
; September 14, 2020

; Ports:
```

```
;   External negative logic buttons connected to P4.0
;   External positive logic buttons connected to P5.0

; Pseudocode:
;     Initialize port 4.0 for external LED output
;     Enable P4OUT so LED is initially on
;     Initialize port 5.0 for external switch input
;     Loop:
;          Branch to Delay to wait 250 ms
;          Branch to Switch Input to gather input data (0x0 or 0x1)
;          Compare and determine if press or no press
;                If P5IN == 0x1 (the switch is pressed), set P4.0 to
toggle LED on and off repeatedly
;                If P5IN == 0x0 (the switch is not pressed), turn LED
on by enabling P4OUT
;     Repeat loop

; ROM:
       .thumb

       .text
       .align  2

; External, Negative logic LED:
P4OUT   .field 0x40004C23,32  ; Port 4 Output
P4DIR   .field 0x40004C25,32  ; Port 4 Direction
P4SEL0  .field 0x40004C2B,32  ; Port 4 Select 0
P4SEL1  .field 0x40004C2D,32  ; Port 4 Select 1
; External, Positive logic Switch:
P5IN    .field 0x40004C40,32  ; Port 5 Input
P5DIR   .field 0x40004C44,32  ; Port 5 Direction
P5REN   .field 0x40004C46,32  ; Port 5 Resistor Enable
P5SEL0  .field 0x40004C4A,32  ; Port 5 Select 0
P5SEL1  .field 0x40004C4C,32  ; Port 5 Select 1

       .global main
       .thumbfunc main

main: .asmfunc
      push {LR}
      BL Port4_Init
      BL Port5_Init
      BL Loop
      pop {LR}
      BX         LR
      .endasmfunc

Port4_Init: .asmfunc
    ; initialize P4.0 LED and make it output
    LDR  R1, P4SEL0
    LDRB R0, [R1]
    BIC  R0, R0, #0x01              ; configure LED pins as GPIO
    STRB R0, [R1]
    LDR  R1, P4SEL1
    LDRB R0, [R1]
    BIC  R0, R0, #0x01              ; configure LED pins as GPIO
    STRB R0, [R1]
    ; make LED pins out
    LDR  R1, P4DIR
    LDRB R0, [R1]
    ORR  R0, R0, #0x01              ; output direction
    STRB R0, [R1]
    LDR  R1, P4OUT
    LDRB R0, [R1]
    ORR  R0, R0, #0x01              ; LED high initially
    STRB R0, [R1]
    BX   LR
    .endasmfunc

Port5_Init: .asmfunc
    ; configure P5.0 switch as GPIO input
    LDR  R1, P5SEL0
    LDRB R0, [R1]
    BIC  R0, R0, #0x01              ; configure P5.0 as GPIO
    STRB R0, [R1]
    LDR  R1, P5SEL1
    LDRB R0, [R1]
    BIC  R0, R0, #0x01              ; configure P5.0 as GPIO
    STRB R0, [R1]
    ; make P5.0 in
    LDR  R1, P5DIR
    LDRB R0, [R1]
    BIC  R0, R0, #0x01              ; input direction
    STRB R0, [R1]
    ; disable pull resistor on P5.0 (since external)
    LDR  R1, P5REN
    LDRB R0, [R1]
    BIC  R0, R0, #0x01              ; disable pull resistor
    STRB R0, [R1]
    BX   LR
    .endasmfunc

Loop: .asmfunc
     BL   Delay
     BL   Switch_Input               ; status = R0 = 0x00 or 0x01
continue
     CMP R0, #0x01
     BEQ pressed
     CMP R0, #0x00
     BEQ not_pressed
     BL  Loop
pressed                                         ; LED toggles when
button pressed
     BL   LED_Toggle
```

```
      BL  Loop
not_pressed                                        ; LED on when button
not pressed
      BL  LED_On
     BL  Loop
     .endasmfunc

Switch_Input: .asmfunc
      LDR  R1, P5IN
     LDRB R0, [R1]                 ; 8-bit contents of register
     AND  R0, R0, #0x01            ; get just P5.0
      BX   LR                                      ; return 0x00 or 0x01
      .endasmfunc

Delay:  .asmfunc
      ; delay by 250 ms (quarter second)
      MOV R0, #0x0D090
      ORR R0, R0, #0x30000
wait                                          ; delays for 750,000 cycles
      SUBS R0, R0, #0x01
      BNE wait
      BX   LR
      .endasmfunc

LED_On: .asmfunc
     LDR  R1, P4OUT
     LDRB R0, [R1]
     ORR  R0, R0, #0x01            ; LED is high
     STRB R0, [R1]
     BX   LR
     .endasmfunc

LED_Toggle: .asmfunc
     LDR  R1, P4OUT
     LDRB R0, [R1]
     EOR  R0, R0, #0x01            ; toggles LED 1-> 0 or 0 -> 1
     STRB R0, [R1]
     BX   LR
     .endasmfunc

     .end
```