California State University, Northridge

Department of Electrical & Computer Engineering

Lab 8

Writing and Calling Subroutines

November 3, 2022

ECE 425L

Assigned by: Neda Khavari

Written By: Pathum Addarapathirana & Cristian Robles

Introduction:

In lab 8 we are introduced to writing and calling subroutines. The objective of this lab is to learn how to write and call subroutines in separate files, perform calls in assembly and C, and to understand switch debouncing.

Procedure:

Equipment Used

- Keil uVision4
- Keil Debugger
- LPC2148 Education Board

Description of Procedure

1. Set up startup code and constants. This is the code needed to start the lab that includes my startup code from lab 3. Initially we have to import our delays that we created in separate files. (more on that later) We also set the constants used in the pin configuration we will use later to control the LEDs and delays for ease and better understandability of code.

```
1         GLOBAL  user_code
2         AREA    reset,CODE,READONLY
3  ;user_code  ;this label is neccessary
4
5         ENTRY
6         IMPORT RTN
7         IMPORT cDelay
8
9  IO0BASE     EQU 0xE0028000
10 ;IO0PIN     EQU 0
11 IO0DIR      EQU 0x8
12 IO0SET      EQU 0x4
13 IO0CLR      EQU 0xC
14 ;IO0PIN     EQU 0xE0028000
15
16 ;Lab 8 stuff
17 B_var       RN      7
18 onoff       RN      8
19
20 delayone    EQU     0x00061A80 ;1 second dely      //4mil is about 1 second delay for arm
21 debounce    EQU     0x000186A0   ;for testing purposes only
22
23 user_code;this label is neccessary
```

2. In the next part of our code we set the pins we will be working with as GPIO and set the direction of the pins to be outputs. This was done somewhat differently from previous labs (shown in next step).

```
26 PINSEL0     EQU     0xE002C000      ;pin function for port 0, equate symbolic name PINSEL0 as address 0xE002C000
27
28                                     ;;Selecting funcion as GPIO by writing all zeros to given address
29             MOV     r0,#0           ;moves #0 into register r0
30             LDR     r1,=PINSEL0     ;puts what is stored in register 0xE002C000 (PINSEL0) in r1 register; CANNOT PUT MOV, outputs error in kei
31             STR     r0,[r1]         ;copies value stored in r0 to memory address specified by r1
```

3. When setting IO Direction and the initial clear and set of the LEDs we simply created a loop for these tasks as you will see in the next steps, some tasks require doing these steps again. So for a more clear and shorter code we implemented a loop.

```
37  ;Task1                          ;Turn on lights using delay from another file
38
39          ;Initially Assigning B constant to zero
40          MOV B_var,#0
41
42          ;set IODIRECTION
43          BL      setIODIR            ;;Selecting signal direction of each port pin, we put '1' in each to bit to make it an output pi
44
45          BL      clrset              ;initial clr and set
```

4. For the first task we are to set the lights to turn on one by one using loops and a 1 second delay. We calculated the delay for one second in ARM is about 4 million loops (as shown in step 1 screenshot, the 'delayone' hex number). We first load the delay to a register then we branch link to another file that we named RTN. This loop follows the same logic as the previous lab where we subtract 1 from the imported number in r5. Then once that is done, PC is loaded with the next line or link register of the previous file so it continues with the next step after the loop. That step is a loop where we do a logical left shift of bit 8 in a loop 8 times for all 8 LEDs. This is shown in the code below.

```
47  task1
48              CMP     B_var,#8        ;B_var = 8 BEQ next loop for task 2
49              BEQ     task2
50
51  turnLow
52              LDR     r5,=delayone    ;delaying onesec
53              BL      RTN             ;branching to external file delay_arm.s
54                                      ;for logical shift
55              LDR     r0,=0x00000100  ;binary 100000000 (bit 8)
56              LSL     r0,B_var        ;Logical shift left of B_var
57
58              LDR     r1,=IO0BASE     ;forcing low
59              STR     r0,[r1,#IO0CLR]
60
61              ADD     B_var,B_var,#1  ;+1 B_var
62
63              B       task1
```

```
1           GLOBAL  RTN
2           AREA    mycode,CODE,READONLY
3           ENTRY
4           ;STMFD  sp!,{LR}
5
6
7
8   RTN         SUBS    r5,#1
9               BNE     RTN             ;jump to given address if not zero
10              MOVEQ   pc,lr
11              ;LDMFD  sp!,{PC}        ;works without LDMFD and STMFD
12
13          B RTN
14          END
```

5. For the next task we are to do the same as task 1 but this time use a delay written in C in another file. We follow the same logic as task 1 for this task but instead of loading delay one to r5 we load it to r0 as that is what a C file will initially take as an input. For the C file shown below we simply use a while loop to subtract 1 from the number imported to g from r0.

```
task2               ;same as task 1 in C delay

            ;set IODIRECTION
            BL      setIODIR
            ;initally clearing so all lights are back off from task1
            BL      clrset
            ;initally setting B_var to zero
            MOV     B_var,#0

turnLow2
            CMP     B_var, #8           ;
            BEQ     endref

            LDR     r0,=0x00000100  ;binary 100000000 (bit 8)
            LSL     r0,B_var        ;Logical shift left of B_var

            LDR     r1,=IO0BASE     ;forcing low
            STR     r0,[r1,#IO0CLR]

            LDR     r0,=delayone
            BL      cDelay          ;delay subroutine here

            ADD     B_var,B_var,#1  ;+1 B_var

            CMP     B_var, #8           ;
            BEQ     endref

            B       turnLow2
```

```
#include<stdio.h>

void cDelay(g){
  int a=0;
  while(g>a){
    g--;
}
}
```

6. For task 3 we are to simply use the pushbutton to turn on and off 4 leds from the set of 8 leds. In this task we have to recognize the use of a debouncer, which is needed when you physically press a button there will be small bounces which will

register the push multiple times. So for that reason a debouncer is needed which in our case is just a small delay that will account for it.

7. We start this step first by redoing IO direction as we need to set pin 14 as an input pin. This is shown below.

```
 99  task3
100             BL      clrset
101
102             LDR     r0,LEDPINS      ;Assigning output for all pins except pin 14 (binary 1011111100000000)
103             LDR     r1,=IO0BASE     ;puts 0xE0028008 or IO0DIR to r1 register
104             STR     r0,[r1,#IO0DIR]         ;copies value stored in r0 (0x0000BF00) to memory address of r1
105
106             BL      clrset
```

8. Then we create a loop, just like the previous lab, to check the status of pin 14. Once a push is detected we branch to a new loop called onoff_b.

```
108 sts14
109             ;Checking pin14 status
110             LDR     r0,=IO0BASE     ;also IO0PIN!
111             LDR     r9,[r0]
112             TST     r9,#0x00004000  ;testing if pin 14 (binary 100000000000000) is a zero (AND operation)
113             BEQ     onoff_b
114             B       sts14
```

9. Onoff_b simply manages the on and off logic as shown below. It basically turns on the LED's if off and turns off if on. But this is where we implement the debouncer loop or delay as shown below. This accounts for the fluctuations of the input right before the ON and OFF of the delays. Shown below is a screenshot of onoff_b.

```
123 onoff_b
124             LDR     r5,=debounce                ;<----- debouncing loop
125 LP          SUBS    r5,#1
126             BNE     LP
127
128             TST     onoff,#1
129             BEQ     on4led
130             B       off4led
```

10. On4led and off4led are loops that simply force low or force high the 4 leds respectively.

```
116 on4led
117             LDR     r0,=0xF00
118             LDR     r1,=IO0BASE     ;forcing low, LED ON
119             STR     r0,[r1,#IO0CLR]
120             MOV     onoff, #1
121             B       rst14
```

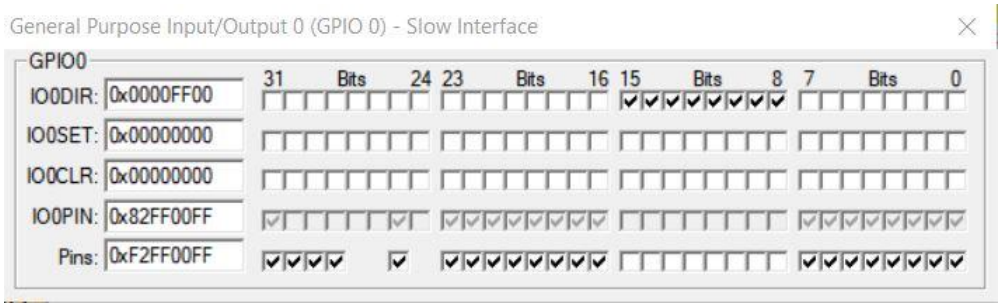
```
132  off4led
133                 LDR     r0,=0xF00
134                 LDR     r1,=IO0BASE     ;forcing high, LED OFF
135                 STR     r0,[r1,#IO0SET]
136                 MOV     onoff, #0
```

11. This screenshot below is the rst14 loop. This loop simply resets the leds which is needed after turning on or off the LEDs before checking the status of pin 14 again.

```
140  rst14          ;Set Pin 14 to 1 again
141             ;Assigns ALL PINS to Input/Output, not sure why this needs to be done
142             LDR r0,=0x0000FF00
143             LDR r1,=IO0BASE
144             STR r0,[r1,#IO0DIR]             ;Assigning all as output pins
145
146             LDR r0,LEDPINS          ;setting output for all pins but pin14
147             LDR r1,=IO0BASE
148             STR r0,[r1,#IO0DIR]
149
150             B       sts14
```

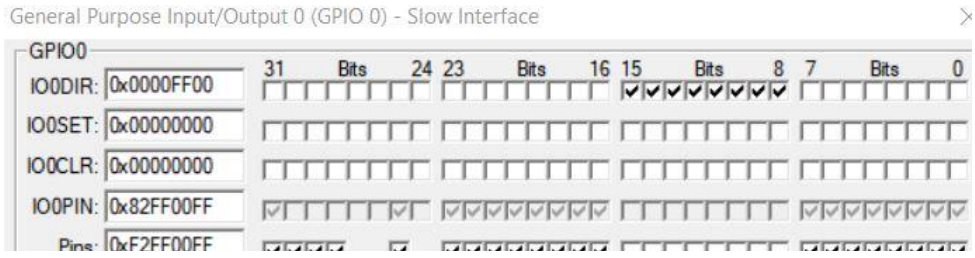
Results:

Task 1



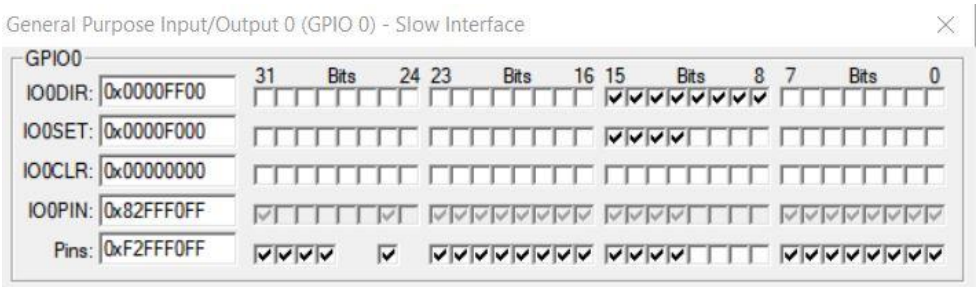
In task 1 all LEDs turned on after the delay in between them. This was done with the assembly subroutine

Task 2



In task 2 all LEDs turned on after the delay in between them. This was done with the C subroutine

Task 3



In task 3 the first four LEDs toggled on and off between each button press. We used a after each button press to debounce the physical board but it is not noticeable in simulation

Conclusion:

In conclusion, we were able to successfully write and call subroutines using both C and assembly. Both subroutines were able to accomplish the same task which was to create a delay in between flashing LEDs. By using a subroutine we are able to use a block of code repeatedly for tasks that are similar like a delay. Using subroutines also saves space by using less code and it allows our code to be more readable and efficient. In task 3 we were able to successfully debounce the button connected to pin 14 that we were using as an input. Without the debouncing the LPC2148 would register multiple presses when the button was pressed once but after debouncing with a 100ns delay we observed one press being registered after every press. The debouncing did not affect the simulator as the issue was only seen on the physical board. This lab introduced us to subroutines and furthered our knowledge of using pins as general purpose input and outputs.