Practical 4

Mattew Rothenburg
EEE3096S
University of Cape Town
South Africa
RTHMAT003

October 18, 2024

1 Introduction

The goal of practical 4 was to implement code on the STM32 UCT development board using assembly programming. Several task were required. The LED's on the board should blink displaying the binary value of incrementing value. The value should increment every 0.7 seconds by default, and by 0.3 seconds when SW1 is held down. When SW0 is held down the LED's should increment by 2. SW1 and SW0 can both be held at the same time. When SW2 is held the pattern should freeze on the current step and only return when released. If SW3 is held the pattern should display the value 0xAA until released.

2 Method

In order to view whether a push button was pressed the input data register was used. First the address of this register was loaded into R0, then using this register the current value of the input data register was stored in R4.

```
LDR R0, GPIOA_BASE          @ GPIOA
    address
LDR R4, [R0, #0x10]         @ Load IDR
```

In order to access the specific bits corresponding to each push button a bit mask was used. A bit mask was bitwised AND'd against the IDR and stored in another register. This ensured that the all bits were 0 except the one corresponding to a pushbutton. Therefore that particular pushbutton could be evaluated. The bit would be 0 if the pushbutton was pressed and 1 if it was not. This register was then compared against 0, if true the pushbutton was known to be pressed.

```
@Check pushbutton 3
MOVS R6, #0b0100            @ Bitmask
ANDS R6, R4, R6             @ Extract
    SW2
CMP R6, #0b0000
```

In order to set the LED's to a specific pattern the output data register was used. GPIOB's address was loaded into a register and then a value representing a specific pattern was stored at this register with a specific offset.

```
LDR R0, GPIOB_BASE          @ GIOB
    address
STR R3, [R0, #0x14]         @
    store pattern into GPIOB
```

By default the value of the LED's should increment by 1 every 0.7 seconds. The register R2 was used to represent the counter for the LED pattern and is set to increment once in each loop. In order to implement a delay loop a subtraction loop was implemented, a large value representing the length of the delay would be set, and this value decreases in value until it hits zero. As this operation takes some time this simulates a delay.

```
delay_loop:
    SUBS R3, R3, #1             @
        count down
    BNE delay_loop          @ loop
        back if not 0
    B main_loop                 @
        return to top
```

When SW0 was pressed the code would jump to a label to increment the LED counter by one more before returning to the code, thus incrementing the value by 2 (including the default increment.)

```
increment_again:
    ADDS R2, R2, #1
    B increment_return
```

When SW1 is pressed the code sets a smaller delay value thus decreasing the delay to 0.3 sec.

```
short_delay:
    LDR R3, SHORT_DELAY_CNT    @ Load short delay counter value
    B delay_loop
```

Upon SW2 being pressed the code automatically jumps straight back to the top of the execution loop. This traps the processor and does not allow any of the other code to execute, this essentially pauses the pattern while SW2 is being held.

```
    CMP R6, #0b0000
    BEQ main_loop              @ jmp if pressed
```

When SW3 is pressed the code jumps to a label where the LED pattern is set to 0xAA, after this the code jumps back to the top of the execution loop once again blocking the rest of the code, thus the counter register doesn't change.

```
    @Check pushbutton 2
    MOVS R6, #0b1000           @ Bitmask
    ANDS R6, R4, R6            @ Extract SW3
    CMP R6, #0b0000
    BEQ set_pattern             @ jmp if pressed
set_pattern:
    MOVS R3, #0xAA
    LDR R0, GPIOB_BASE
    STR R3, [R0, #0x14]
    B main_loop
```

3 Conclusion

In conclusion all code executed well. The use of assembly proved challenging but gave good insight to the inner workings of the STM32.

4 GitHub Link

Click Here for Github

```
/*
 * assembly.s
 *
 */

 @ DO NOT EDIT
  .syntax unified
    .text
    .global ASM_Main
    .thumb_func

@ DO NOT EDIT
vectors:
  .word 0x20002000
  .word ASM_Main + 1

@ DO NOT EDIT label ASM_Main
ASM_Main:

  @ Some code is given below for you to start with
  LDR R0, RCC_BASE      @ Enable clock for GPIOA and B by setting bit 17 and 18 in RCC_AHBENR
  LDR R1, [R0, #0x14]
  LDR R2, AHBENR_GPIOAB @ AHBENR_GPIOAB is defined under LITERALS at the end of the code
  ORRS R1, R1, R2
  STR R1, [R0, #0x14]

  LDR R0, GPIOA_BASE    @ Enable pull-up resistors for pushbuttons
  MOVS R1, #0b01010101
  STR R1, [R0, #0x0C]
  LDR R1, GPIOB_BASE    @ Set pins connected to LEDs to outputs
  LDR R2, MODER_OUTPUT
  STR R2, [R1, #0]
  MOVS R2, #0           @ NOTE: R2 will be dedicated to holding the value on the LEDs



@ TODO: Add code, labels and logic for button checks and LED patterns

main_loop:

    LDR R0, GPIOA_BASE        @ GPIOA address
    LDR R4, [R0, #0x10]       @ Load IDR

    @Check pushbutton 3
    MOVS R6, #0b0100          @ Bitmask
    ANDS R6, R4, R6           @ Extract SW2
    CMP R6, #0b0000
    BEQ main_loop             @ jmp if pressed

    @Check pushbutton 2
    MOVS R6, #0b1000          @ Bitmask
    ANDS R6, R4, R6           @ Extract SW3
    CMP R6, #0b0000
    BEQ set_pattern            @ jmp if pressed

    ADDS R2, R2, #1

    MOVS R6, #0b0001          @ Bitmask
    ANDS R6, R4, R6           @ Extract SW0
    CMP R6, #0b0000
    BEQ increment_again        @branch if pressed


    increment_return:

        MOVS R3, R2
        LDR R0, GPIOB_BASE        @ GIOB address
        STR R3, [R0, #0x14]       @ store pattern into GPIOB

        LDR R3, LONG_DELAY_CNT    @ Long delay counter val

```

```
        MOVS R6, #0b0010          @ Bitmask
        ANDS R6, R4, R6           @ Extract SW2
        CMP R6, #0b0000
        BEQ short_delay            @ Branch if pressed

        delay_loop:
            SUBS R3, R3, #1            @ count down
            BNE delay_loop       @ loop back if not 0
            B main_loop                @ return to top

short_delay:
    LDR R3, SHORT_DELAY_CNT    @ Load short delay counter value
    B delay_loop


set_pattern:
    MOVS R3, #0xAA
    LDR R0, GPIOB_BASE
    STR R3, [R0, #0x14]
    B main_loop

increment_again:
    ADDS R2, R2, #1
    B increment_return

@ LITERALS
.align
RCC_BASE:           .word 0x40021000
AHBENR_GPIOAB:      .word 0b1100000000000000000
GPIOA_BASE:         .word 0x48000000
GPIOB_BASE:         .word 0x48000400
MODER_OUTPUT:       .word 0x5555
LONG_DELAY_CNT:     .word 1400000    @ Adjust for 0.7 second delay
SHORT_DELAY_CNT:    .word 600000     @ Adjust for 0.3 second delay
```

UNIVERSITY OF CAPE TOWN

IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

DEPARTMENT OF ELECTRICAL ENGINEERING

EEE3095S/EEE3096S Practical 4 Demonstrations/Solutions 2024

Total Marks Available: 15

NB Please take a photo of this mark sheet and submit it with your report!

Group No.	30	
	Stn 1	Stn2
Student no.	RTHMAT003	
Name	Matthew Rothenburg	
Signature	[signature]	

Action + Mark Allocation	Mark
By default, the LEDs should increment by 1 every 0.7 seconds (with the count starting from 0). [2 marks]	✓
While SW0 is being held down, the LEDs should change to increment by **2** every 0.7 seconds. [2 Marks]	✓
While SW1 is being held down, the increment timing should change to every **0.3** seconds. [2 Marks]	✓
If SW0 and SW1 are both held down, the LEDs should increment by **2 every 0.3 seconds**. [2 marks]	✓
While SW2 is being held down, the LED pattern should be set to **0xAA**. Naturally, the pattern should stay at 0xAA until SW2 is released, at which point it will continue counting normally from there. [2 marks]	✓
While SW3 is being held down, the pattern should **freeze**, and then resume counting only when SW3 is released. [2 marks]	✓
Check code: well-written, well commented code. Code pushed to Git. [3 marks]	✓
Check code: Assembly is used to implement the above; if not, the student automatically gets **zero** for the demo.	✓
Total	15 /15

Tutor Name:	Khavish
Tutor Signature:	[signature]