ME 586: Lab 2 Report
STM32 Debugger, Simulator, and GPIO Practice

Harsh Savla & TJ Wiegman

2022-09-20

Abstract

In this lab, we created assembly programs for the STM32 microcontroller and tested them with both the Keil µVision simulator and a hardware STM32F100RB board. These programs aimed to give us practice using the debugger to change memory values on the fly and step through code in a prescribed fashion, both in simulation and in hardware. We also created a program that utilized inputs and outputs through the GPIO pins of the device; we were able to successfully read those hardware inputs (or simulated hardware inputs) as well as output a physical electrical signal (or simulated physical signal).

1 First Section Checkpoints

1.1 Simulation: Verifying Homework Results

Overview

The first task in this first section was to compile an assembly program from Homework 1 (Problem 3), and compare the (simulated) debugger output to what we calculated by hand for the original assignment.

Procedure

The code used in this section is shown in Appendix A.2.1. This program was compiled and simulated using Keil µVision. Stepping through the code one line at a time, register and memory values were compared to those calculated by hand for the original assignment.

Results and Discussion

The values produced by the simulated hardware matched our paper predictions. This task was helpful for learning how to write and load an assembly program into the simulator and read or write values to registers during operation using the simulated debugger.

1.2 Simulation: Generating the Correct Quotient and Remainder

Overview

The second task was to compile an assembly program written for Homework 1 (Problem 4) and verify it using the simulated debugger.

Procedure

The code used in this section is shown in Appendix A.2.2. This program, which calculates the average of a set of unsigned 8-bit numbers, was compiled and simulated using Keil µVision. Sample data values were injected into RAM using the simulated debugger and the output was checked to see if the algorithm could accurately calculate the arithmetic mean of the sample data values.

Results and Discussion

The program was, after some debugging and modifications to bring it up to the quality shown in the appendix, able to accurately calculate the correct average (and remainder) values for any values injected into RAM. This task was helpful for learning how to modify RAM during operation using the simulated debugger.

1.3 Hardware: Generating the Correct Quotient and Remainder

Overview

The third task was to compile an assembly program written for Homework 1 (Problem 4) and verify it using the hardware debugger.

Procedure

The code used in this section is shown in Appendix A.2.2. This program, which calculates the average of a set of unsigned 8-bit numbers, was loaded onto the STM32 board and run with the hardware debugger. Sample data values were injected into RAM using the debugger and the output was checked to see if the algorithm could accurately calculate the arithmetic mean of the sample data values.

Results and Discussion

The program was able to accurately calculate the correct average (and remainder) values for any values injected into RAM. This task was helpful for learning how to manipulate RAM while in operation using the hardware debugger.

2 Second Section Checkpoints

2.1 Simulation: Read, Interpret, and Output GPIO Signals

Overview

The task was to build an assembly program using the subroutines developed in Homework 2. This program would read in two 3-bit signed numbers from 6 GPIO pins and then output signals to three GPIO pins.

Procedure

The code used in this section is shown in Appendix A.2.4, A.2.3, and A.2.5. This program was run through the simulator and sample data values were injected into RAM using the simulator. The simulated output was checked to see if it matched the requirements.

Results and Discussion

The program was able to set the outputs to low or high correctly. This task was helpful for learing how to manipulate the bits of each number, as well as how to enable and use the GPIO pins.

2.2 Hardware: Read, Interpret, and Output GPIO Signals

Overview

The final task was to do much the same as in Section 2.1, but this time the code was run on STM32 hardware rather than in simulator.

Procedure

The program shown in the appendix was now loaded on STM32. The input switches were connected to the processor pins through the external wires. The output was observed by the state of the LEDs connected to the output pins.

Results and Discussion

The program was able to interface with the GPIO signals, both input switches and output LEDs. The internal logic correctly interpreted the input signals and generated the correct outputs according to the required algorithm.

3 Conclusion

This lab taught us how registers store and manipulate binary numbers, as well as how arithmetic and other operations are done on those binary values. In particular, the practice working with masking and bit shifts (as in Appendix A.2.3) made us much more comfortable manipuating binary values in the registers.

We also learned about using peripherals, such as GPIO, in order to interface the microprocessor with physical hardware such as switches and LEDs. An important ”best practice” that we learned during this part of the lab was to ensure that common internal registers, such as APB2, are not accidentally overwritten–one must only change the flags needed, and not the others.

A final and more unusual lesson learned from this lab was to transfer code only in plaintext files: we tried to copy-and-paste a snippet from a powerpoint we created earlier, and when we tried to compile it, we ended up with all kinds of strange errors that we later realized were due to invisible whitespace characters!

A Appendix

A.1 Flow Charts

Figure 1: Flowchart accompanying code from Appendix A.2.2.

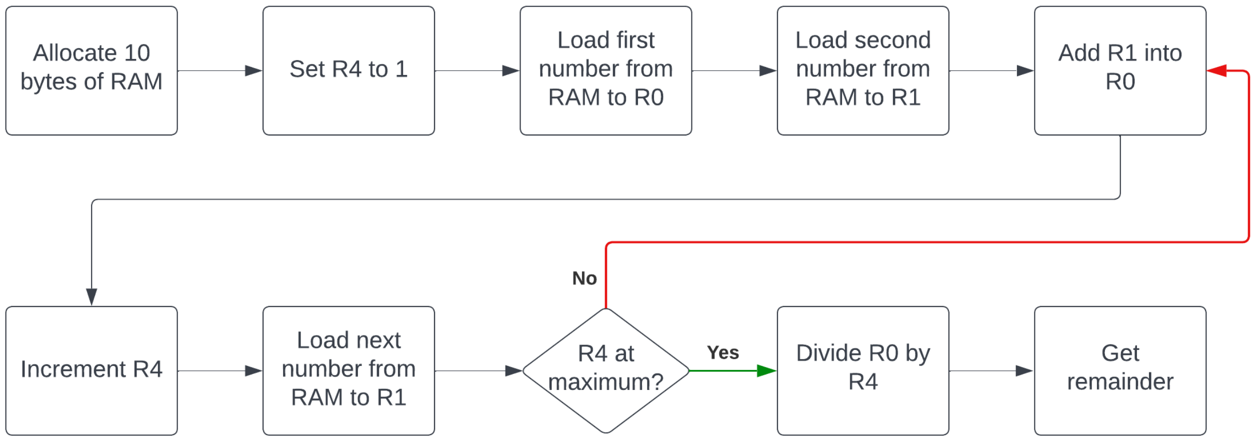


Figure 2: Flowchart accompanying code from Appendix A.2.5.

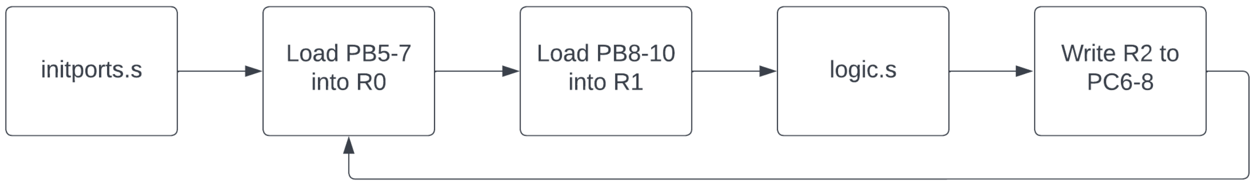


Figure 3: Flowchart accompanying code from Appendix A.2.4.

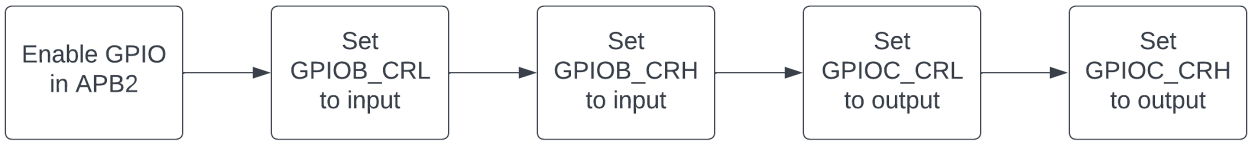
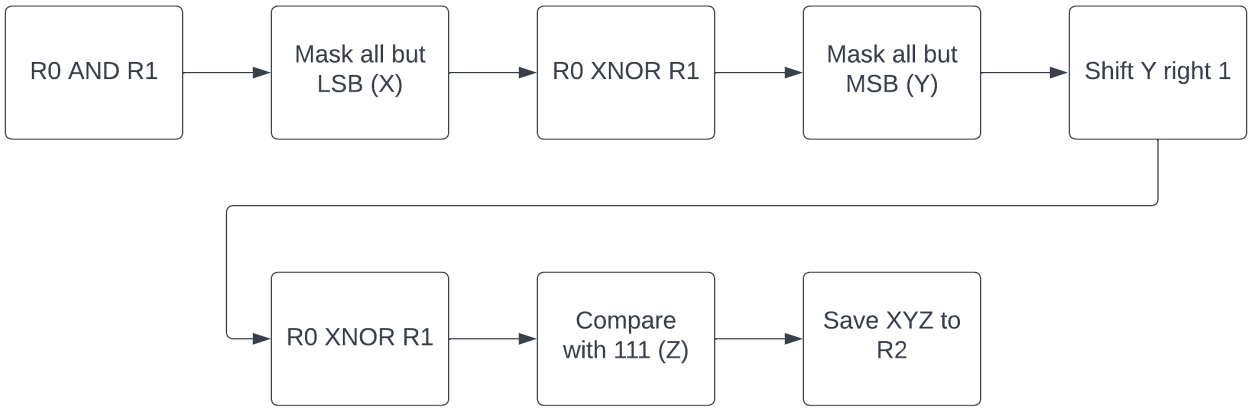


Figure 4: Flowchart accompanying code from Appendix A.2.3.



A.2 Code

A.2.1 Homework 1, Problem 3

```
var1 EQU 0x20000014

    AREA ARMex, CODE, READONLY
        ENTRY
__main PROC
        EXPORT __main

        ldr R0, =var1
        ldrb R1, [R0]
        mov R2, R1
        add R2, #0x25
        strb R2, [R0]
        ror R2, #4

        ENDP
    END
```

A.2.2 Homework 1, Problem 4

```
avg_size EQU 10 ; how many numbers to average?

        AREA MyData, DATA, READWRITE ; allocates RAM
array1   SPACE 10

        AREA ARMex, CODE, READONLY ; code goes here
            ENTRY
__main  PROC
            EXPORT __main

            mov R4, #0x01 ; R4 is number of samples
            ldr R5, =array1
            ldrb R0, [R5], #1
            ldrb R1, [R5], #1

loop_add add R0, R1 ; R0 is running total
```

```
          add R4, #1 ; one more sample in R0
          ldrb R1, [R5], #1
          cmp R4, #avg_size
          bne loop_add

b_div  udiv R3, R0, R4 ; R3 is quotient
          mul R1, R3, R4
          sub R2, R0, R1 ; R2 is remainder
done   b done
       ENDP
    END
```

A.2.3 LOGIC.s

```
       AREA ARMex, CODE, READONLY
logic PROC
       EXPORT logic
       ; push LR to stack
       push {LR}

       ; Check X
       and R3, R0, R1
       and R2, R3, #0x01 ; mask to only 1st bit

       ; Check Y
       eor R3, R0, R1
       mvn R3, R3
       and R3, #0x04 ; mask to only 3rd bit
       lsr R3, #1
       orr R2, R3

       ; Check Z
       eor R3, R0, R1
       mvn R3, R3
       and R3, #0x07
       cmp R3, #0x07
       beq allsame
nsame mov R3, #0x00
       b save2
allsame mov R3, #0x04
save2 orr R2, R3

       ; End subroutine and go back to caller
       pop {LR}
       bx LR

       ENDP
    END
```

A.2.4 INITPORTS.s

```
RCC_APB2ENR EQU 0x40021018
IOPB EQU 2_00001000 ; same as 0x08
IOPC EQU 2_00010000 ; same as 0x10

GPIOB_CRL EQU 0x40010C00
GPIOB_CRH EQU 0x40010C04

```

```
GPIOC_CRL EQU 0x40011000
GPIOC_CRH EQU 0x40011004

       AREA ARMex, CODE, READONLY
initports PROC
       EXPORT initports
       ; push LR to stack
       push {LR}

       ; Adjust APB2 state
       ldr R3, =RCC_APB2ENR
       ldr R1, [R3] ; save current APB2 state
       orr R1, #IOPB+IOPC
       str R1, [R3]

       ; Adjust GPIOB pin modes
       ldr R3, =GPIOB_CRL
       ldr R1, =0x44444444
       str R1, [R3]

       ldr R3, =GPIOB_CRH
       str R1, [R3]

       ; Adjust GPIOC pin modes
       ldr R3, =GPIOC_CRL
       ldr R1, =0x33333333
       str R1, [R3]

       ldr R3, =GPIOC_CRH
       str R1, [R3]

       ; End subroutine and go back to caller
       pop {LR}
       bx LR

       ENDP
   END
```

A.2.5 DIGITAL.s

```
GPIOB_IDR EQU 0x40010C08
GPIOC_ODR EQU 0x4001100C

       AREA ARMex, CODE, READONLY
       ENTRY
__main PROC
       EXPORT __main
       IMPORT initports
       IMPORT logic

       bl initports

loop ldr R3, =GPIOB_IDR
       ldrh R4, [R3]
       and R0, R4, #0x00E0
       ror R0, #5 ; store PB5-7 into R0
       and R1, R4, #0x0700
       ror R1, #8 ; store PB8-10 into R1

       bl logic
```

```

    ldr R3, =GPIOC_ODR
    lsl R2, #6
    str R2, [R3]

    b loop
    ENDP
END
```