ME 586: Lab 4 Report
Interrupt Routines

Harsh Savla & TJ Wiegman

2022-10-11

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 use interrupt routines to detect an external clock signal (or a simulated input) and measure the frequency of that signal relative to the internal clock of the microcontroller.

1 Lab Checkpoints

1.1 Simulation

Overview

The first checkpont focused on debugging the interrupts separately. Each interrupt was disabled so that only one was run at a time, without any other program logic running simultaneously, and affected memory values were examined in the simulated debugger.

Procedure

First, all serial logic was commented out. To test the `initint` routine, which counted up a `clicks` variable each time it was called by the external interrupt pin, the `inittime` interrupt routine was commented out. After confirming it worked as expected, `initint` was commented out and `inittime` was uncommented in its stead. This interrupt routine is called every 100 ms in order to reset `clicks` to zero, copying the old value into a different variable `outclicks`. After confirming this routine also worked as expected, `initint` was uncommented once again, and both routines run simultaneously in order to confirm they did not interfere with one another. After that proved successful, the serial logic was uncommented and the program was able to run as a whole. The main program code is shown in Appendix A.2.1, and the subroutines to enable the various interrupts are in the sections following it.

Results and Discussion

The simulated program ran flawlessly at all stages, as all routines were functional as-written. Thankfully, only minor modifications were necessary in order to test the individual subroutines separately, so it was fairly easy to put the whole program back together when it came time to test it as a complete package.

1.2 Hardware

Overview

The final task was to do the same as before, but this time the code was run on STM32 hardware instead of the simulator.

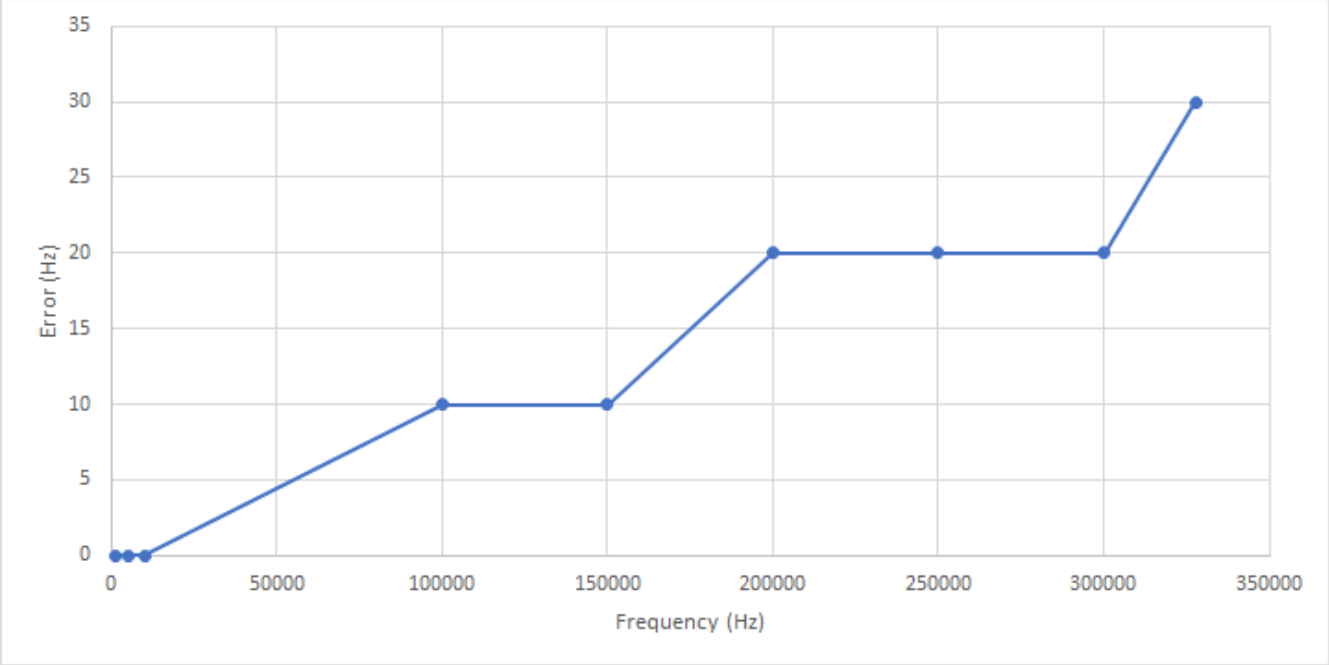
Procedure

The program was loaded on the STM32 board and the square wave generator was connected to the external interrupt pin as instructed in the lab manual. The square wave voltage was set so as to not exceed 5V. The interrupt waveform was also verified with an oscilloscope before connecting it to the board in order to confirm the waveform and peak-to-peak voltage. The code was run for various frequencies and the counter values reported back over serial were recorded. After the program ran sucessfully, the error (difference between actual input frequency and the STM32's measured frequency) was plotted.

Results and Discussion

The program ran flawlessly and the we were able to obtain the desired results. While ploting the error margin against the actual frequency, we observed that the error increased at higher frequnecies, as shown in Figure 1. We also found that the program can only measure frequencies up to 327 680 Hz, as it utilizes 16-bit signed numbers to store the value. Inputting higher frequencies caused the board to (erroneously) report very high negative frequencies due to overflow errors.

Figure 1: Measurement error increased at higher frequencies.



2 Conclusion

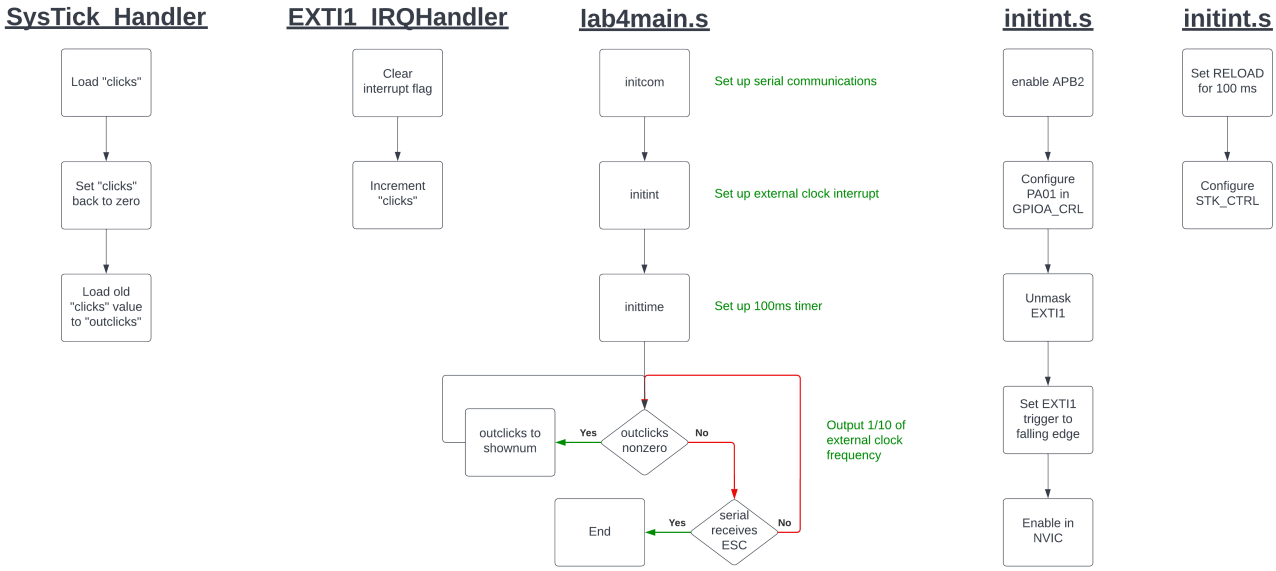
This lab taught us how to properly configure the interrupt channels and handlers. We also learned how to properly generate signals with the square wave generator and connect it to the STM32 board. Finally, we were given a sharp reminder that overflow errors will crop up whenever one tries to store values into memory locations with insufficient width.

We also learned that carefully writing code pays off, as it was quite pleasant to learn that there is no need to debug anything when there are no bugs!

A Appendix

A.1 Flow Charts

Figure 2: Flowchart for all the routines given in this appendix.



A.2 Code

A.2.1 lab4main.s

This file contains both the main program logic as well as the interrupt handler routines, `SysTick_Handler` and `EXTI1_IRQHandler`.

```
; Harsh Savla & TJ Wiegman
; ME 58600
; 2022-09-26
; lab4main.s

EXTI_PR EQU 0x40010414

; allocate some RAM for clicks and outclicks
       AREA MyData, DATA, READWRITE
clicks SPACE 2
outclicks SPACE 2

; program code
       AREA ARMex, CODE, READONLY
       ENTRY
__main PROC
       EXPORT __main
       IMPORT initcom
       IMPORT initint
       IMPORT inittime
       IMPORT shownum
       IMPORT checkcom
       IMPORT getchar

       ; Initialize serial communications
       bl initcom

       ; Initialize clicks and outclicks to zero
       mov R0, #0
       ldr R3, =clicks
       strh R0, [R3]
       ldr R3, =outclicks
       strh R0, [R3]

       ; Set up external clock interrupt
       bl initint

       ; Set up 100ms timer
       bl inittime

       ; Main loop
chloop ldr R3, =outclicks
       ldrh R1, [R3]
       cmp R1, #0x0000
       bne shclick

       ; Check if received serial comm
       bl checkcom
       cmp R0, #0xFF
       bne chloop

       ; Check if received character is ESC
       bl getchar
       cmp R0, #0x1B ; ASCII value for ESCAPE
       beq done
```

```
56          b chloop
57
58  shclick mov R0, R1 ; copy outclicks from R1 to R0
59          mov R1, #0
60          strh R1, [R3] ; reset outclicks to zero
61          bl shownum ; R0 still holds old outclicks value
62          b chloop
63
64  done b done
65          ENDP
66
67  EXTI1_IRQHandler PROC
68          EXPORT EXTI1_IRQHandler
69          ; push LR to stack
70          push {LR}
71
72          ; Clear interrupt flag
73          ldr R3, =EXTI_PR
74          mov R1, #0x02
75          str R1, [R3]
76
77          ; Increase clicks value
78          ldr R3, =clicks
79          ldrh R1, [R3]
80          add R1, #1
81          strh R1, [R3]
82
83          ; End subroutine and go back to caller
84          pop {LR}
85          bx LR
86          ENDP
87
88  SysTick_Handler PROC
89          EXPORT SysTick_Handler
90          ; push LR to stack
91          push {LR}
92
93          ; Get clicks, then reset
94          ldr R3, =clicks
95          ldrh R1, [R3]
96          mov R2, #0
97          strh R2, [R3]
98
99          ; Move clicks to outclicks
100         ldr R3, =outclicks
101         strh R1, [R3]
102
103         ; End subroutine and go back to caller
104         pop {LR}
105         bx LR
106         ENDP
107     END
```

A.2.2 initint.s

```
; Harsh Savla & TJ Wiegman
; ME 58600
; 2022-09-26
; initint.s

; a subroutine to enable the external IRQ interrupt pin
RCC_APB2ENR EQU 0x40021018
GPIOA_CRL EQU 0x40010800
EXTI_IMR EQU 0x40010400
EXTI_FTSR EQU 0x4001040C
NVIC_ISER0 EQU 0xE000E100

; program code
       AREA ARMex, CODE, READONLY
       ENTRY
initint PROC
       EXPORT initint
       ; push LR to stack
       push {LR}

       ; Enable GPIO A
       ldr R3, =RCC_APB2ENR
       ldrb R1, [R3]
       orr R1, 0x04 ; enable bit 2 (0100)
       strb R1, [R3]

       ; Configure PA01
       ldr R3, =GPIOA_CRL
       ldrb R1, [R3]
       and R1, #0x0F ; set PA01 to 0000
       orr R1, #0x40 ; set PA01 to 0100
       strb R1, [R3]

       ; Unmask External Interrupt 1
       ldr R3, =EXTI_IMR
       mov R1, #0x02
       str R1, [R3]

       ; Set trigger to falling edge
       ldr R3, =EXTI_FTSR
       mov R1, #0x02
       str R1, [R3]

       ; Enable in NVIC (?)
       ldr R3, =NVIC_ISER0
       mov R1, #0x80
       str R1, [R3]

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

A.2.3 inittime.s

```
; Harsh Savla & TJ Wiegman
; ME 58600
; 2022-09-26
; inittime.s

; a subroutine to set up the SysTick timer interrupt for every 100ms
STK_CTRL EQU 0xE000E010
STK_LOAD EQU 0xE000E014

; program code
       AREA ARMex, CODE, READONLY
       ENTRY
inittime PROC
       EXPORT inittime
       ; push LR to stack
       push {LR}

       ; Set timer to 100 ms
       ldr R3, =STK_LOAD
       ldr R1, =0x249EFF
       str R1, [R3]

       ; Enable timer, systick interrupt, and 24MHz counter
       ldr R3, =STK_CTRL
       mov R1, #0x07
       str R1, [R3]

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