Lab4 Report

ME 586

Lab Date: 09/24/19

*Lab member: Wei Huang, Jee Hwan Park*

**ABSTRACT**

In this lab, we have used Cortex-M3 Assembly Code to calculate an external clock frequency generated using Agilent Function generator. The external clock signal is to be connected to the IRQ interrupt input of the STM32. An IRQ interrupt request is to be generated whenever the clock goes through a falling edge. This will execute the EXTI1\_IRQHandler and increment a 16-bit counter, ‘clicks’ by one and return. In addition, when 100msec have elapsed, the timer sends an interrupt to the microprocessor, causing it to execute SysTick\_Handler and moves contents of ‘clicks’ to ‘outclicks’ and resets ‘clicks’ to zero. To complete these tasks, we have used the IRQ interrupt and the SysTick timer. To observe the result, function generator was used to generate a square wave function and we have used μVision simulator and hardware to compare the result from the actual characteristics of the generated function. From this lab, we learnt how the interrupt works and how it is applied in the system. The result was very accurate and the relationship between the computed external clock frequency and the actual frequency is linear.

**BODY**

**Overview**

In Lab 04, we finished a program that can detect external signal’s frequency. We accomplished this function by using two interrupts. One is an interrupt generated by external signal. Whenever external clock signal comes to a falling edge, an interrupt function, EXTI1\_IRQHandler, will add one to the counter. Another interrupt is controlled by Systick, which will generate an interrupt request every 100ms. SysTick\_Handler stores current count number into a register and then refresh the counter. Though this process, one can get external signal’s frequency by multiplying the count number by 10.

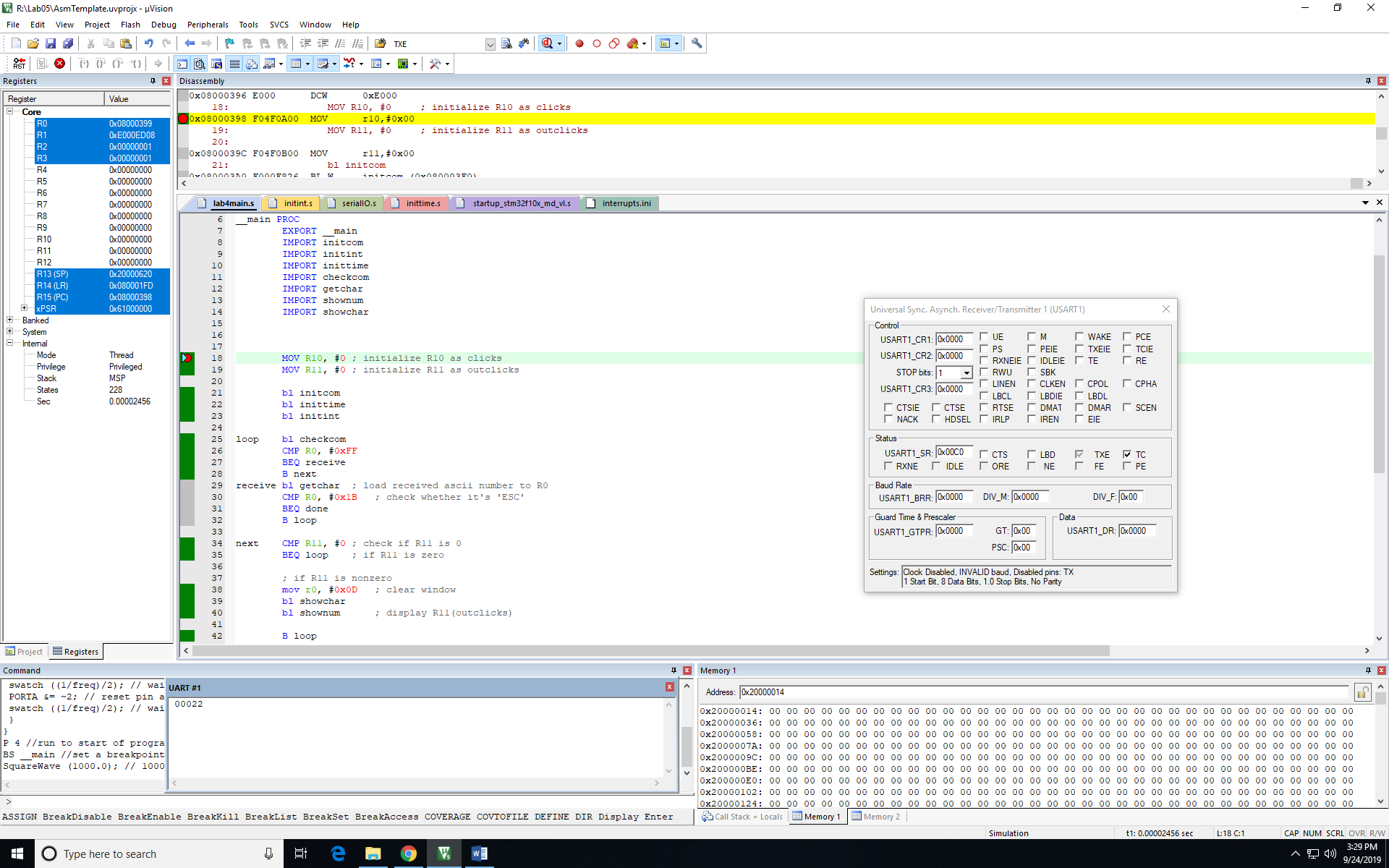
**Procedure**

1. Using software to observe the result with generated signal from software
   1. Comment out Serial I/O operation. To concentrate on IRQ.
   2. Similarly, comment out inittime to exempt any timer interrupt. This will allow programmer to concentrate on IRQ only.
   3. Create an external square signal using software. Create a new file called interrupt.ini to create external signal.
   4. With interrupt.ini created, set up the software correctly by making sure that the interrupt.ini file is correctly added on the initialization file.
   5. Use software simulator to debug the program. In this process make sure to check if IRQ is working correctly by checking if the counter is correctly incrementing.
   6. If IRQ is now working correctly, uncomment inittime and comment out initint.
   7. With inittime uncommented, check if outclicks is correctly incrementing.
   8. After checking each function is working correctly, re-open the original code and execute it using simulator and note the output received. Continue step 8 with different frequency in interrupt.ini
   9. Notice the relationship between signal frequency and the output.
2. Using hardware to observe the result with function generator
3. Change the setting of the software to use STM32.
4. Turn on the HP33120A function generator and properly set up the following characteristics of the signal:
   * Amplitude of the signal is 5 volts
   * Hi-Z mode to change the output from the default output to the output set up for the high-impedance input.
   * Set up the frequency as the frequency used at the section A of the procedure
5. With properly set up setting, run the program
6. Observe the result using Tera Term software.
7. Note down the result and the frequency used in the function generator.
8. Repeat the step from step3 with different frequency used at the section A of the procedure.
9. Notice the relationship between signal frequency and the output.

* After getting all the data (result from the program and the frequency used), use excel to plot the data to observe the relationship easily.

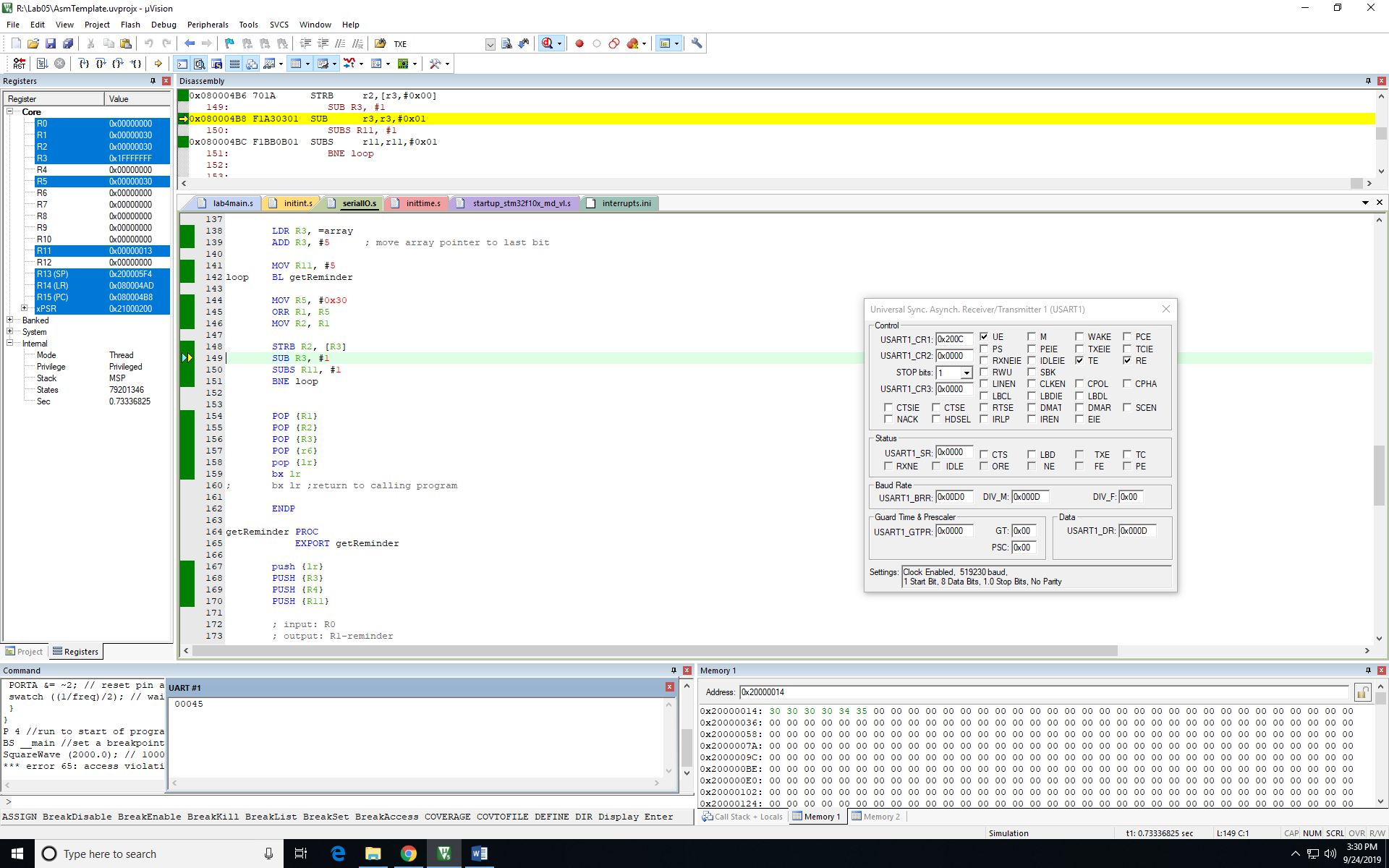
**Result**

1. Using software to observe the result with generated signal from software
   1. Frequency of 1000 Hz



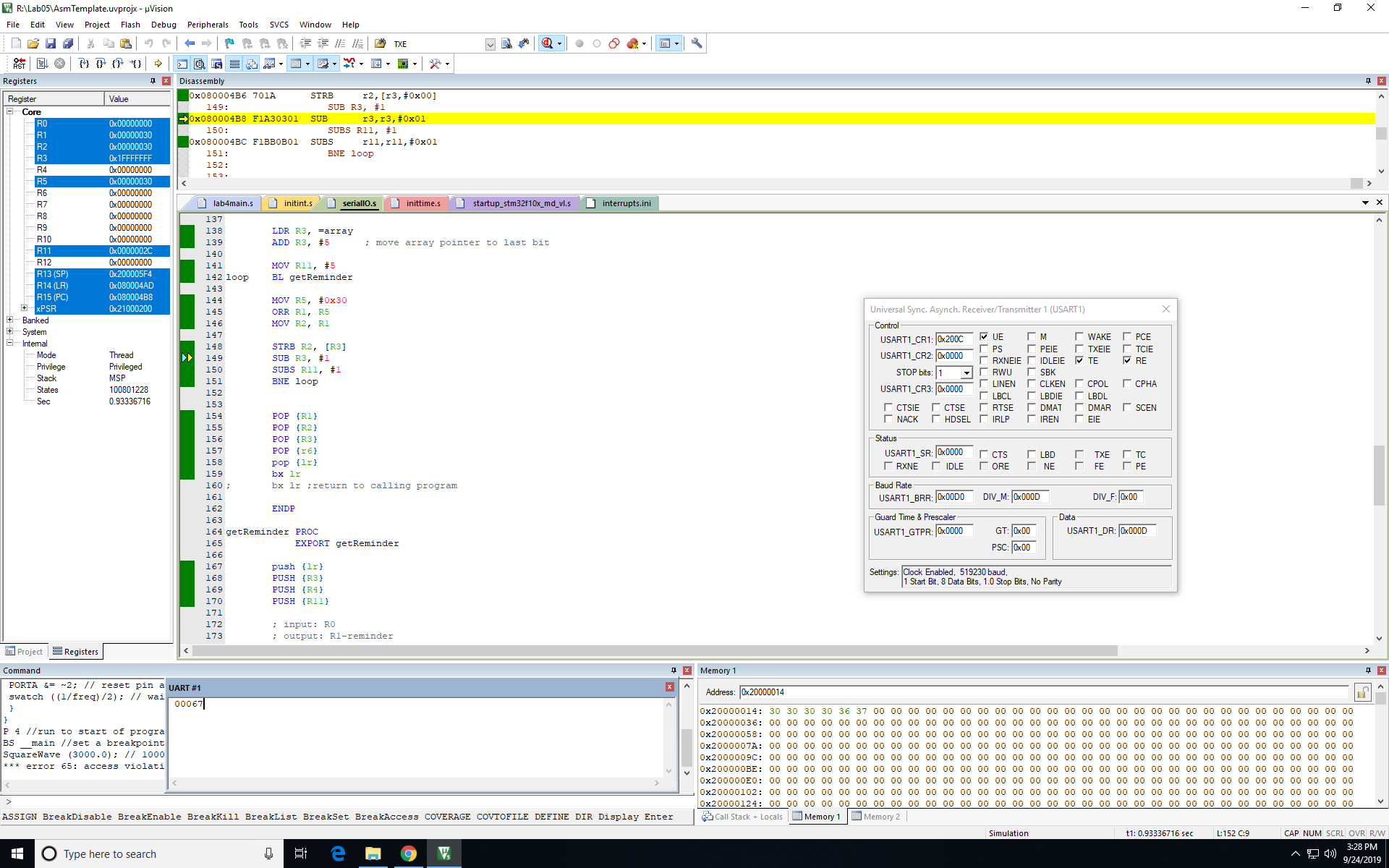
*Fig1. Result on simulator with frequency of 1000Hz*

* 1. Frequency of 2000 Hz



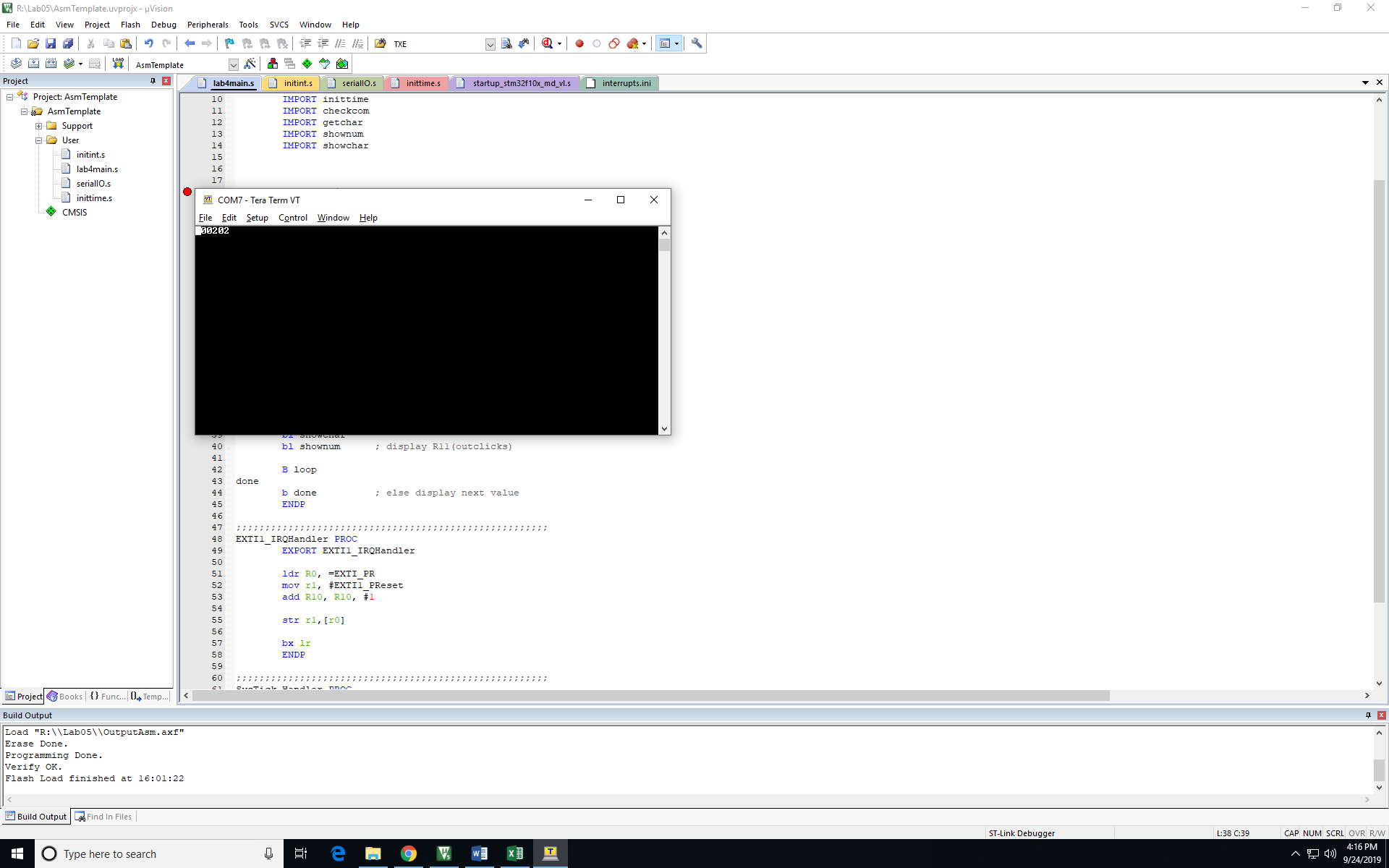
*Fig2. Result on simulator with frequency of 2000Hz*

* 1. Frequency of 3000 Hz



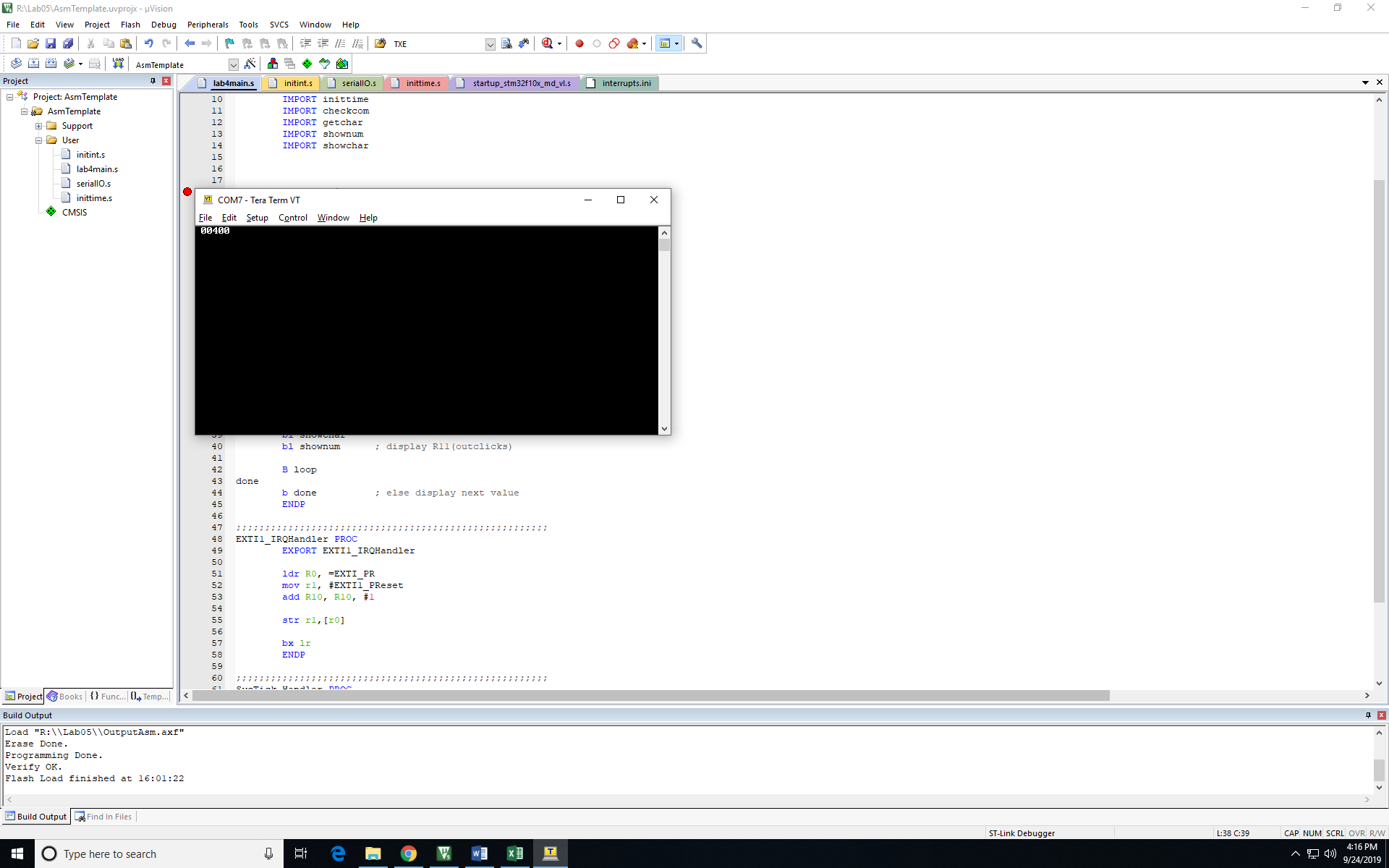
*Fig3. Result on simulator with frequency of 3000Hz*

1. Using hardware to observe the result with function generator
   1. Frequency of 1000 Hz



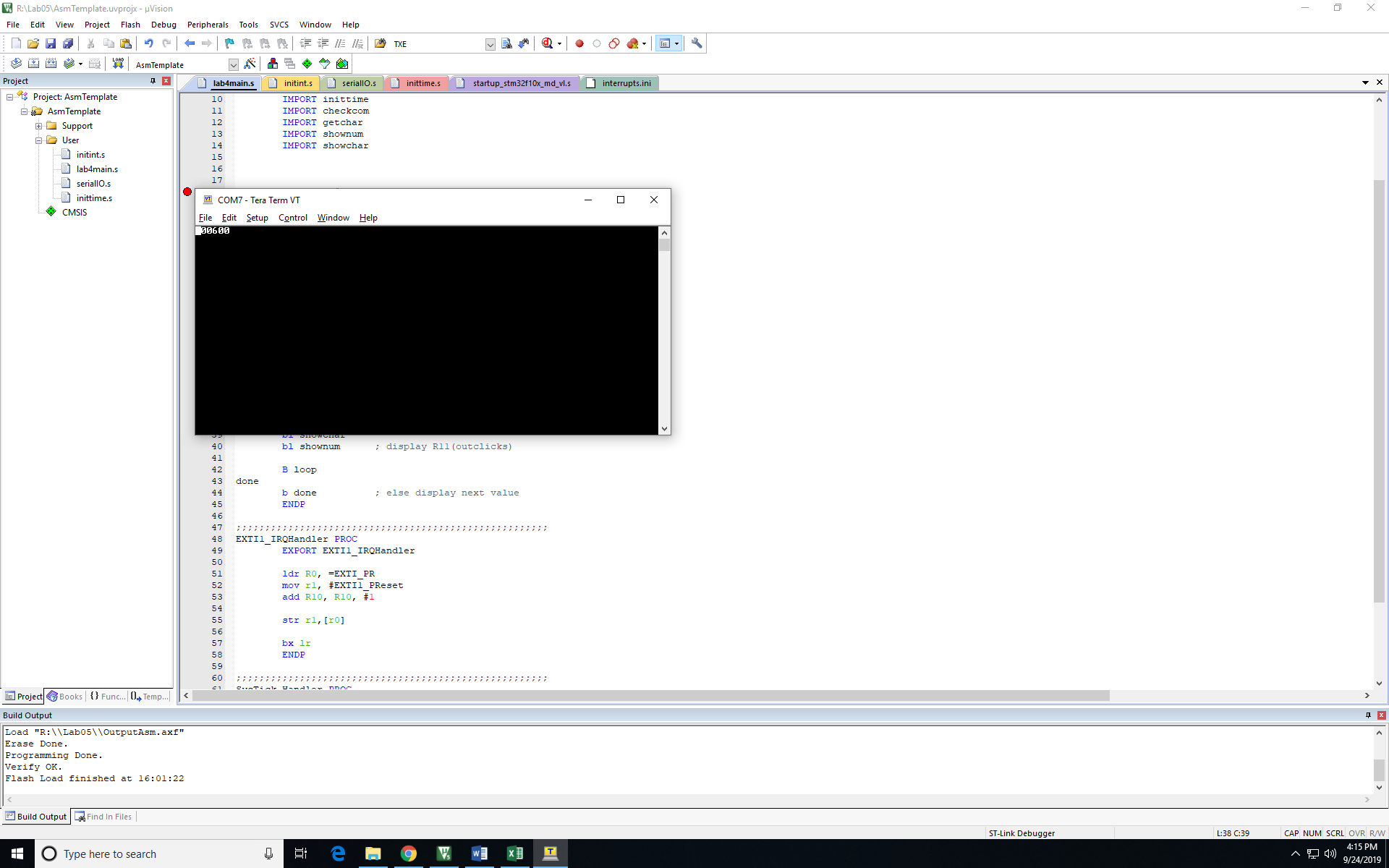
*Fig4. Result using hardware with frequency of 1000Hz*

* 1. Frequency of 2000 Hz



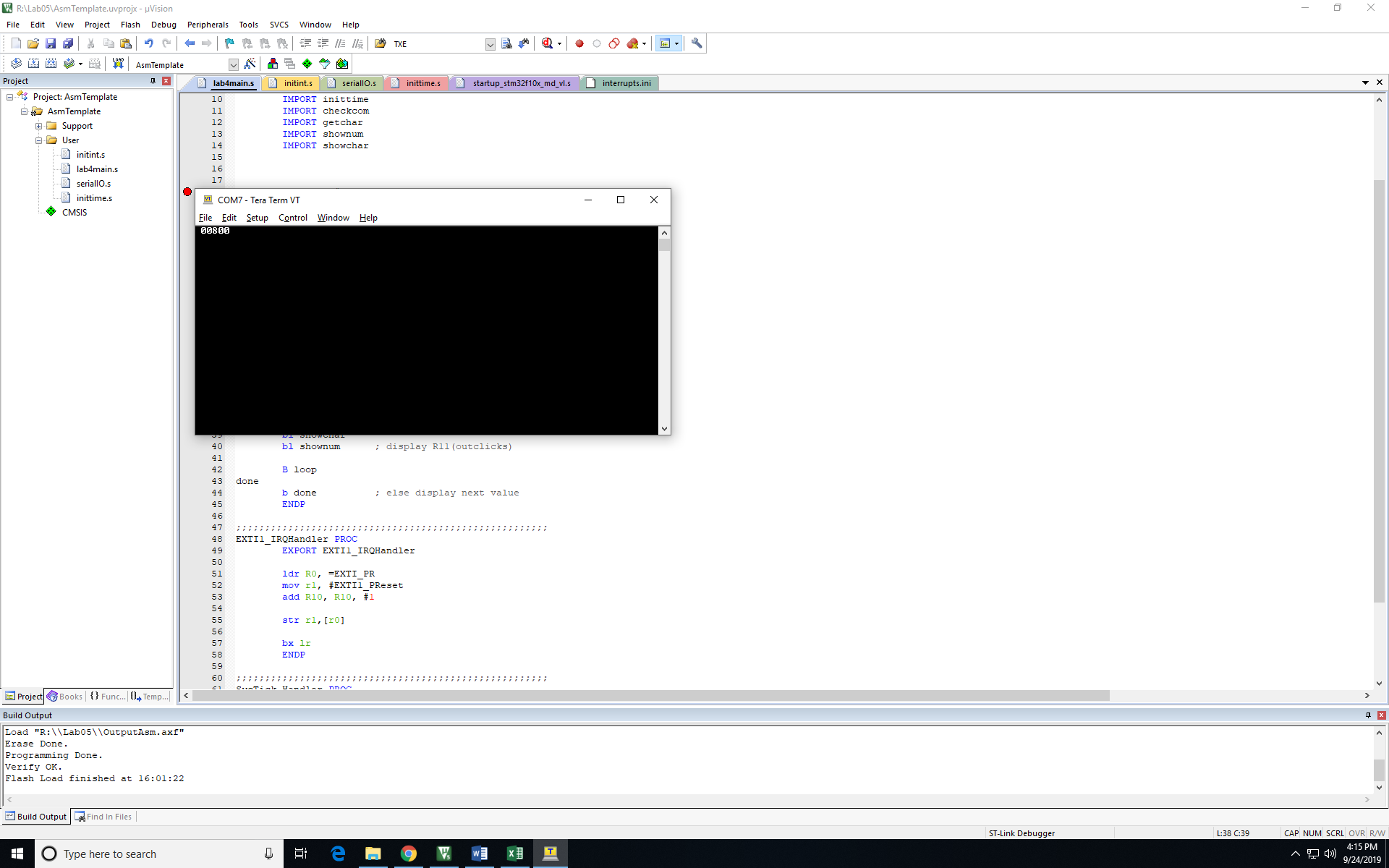
*Fig5. Result using hardware with frequency of 2000Hz*

* 1. Frequency of 3000 Hz



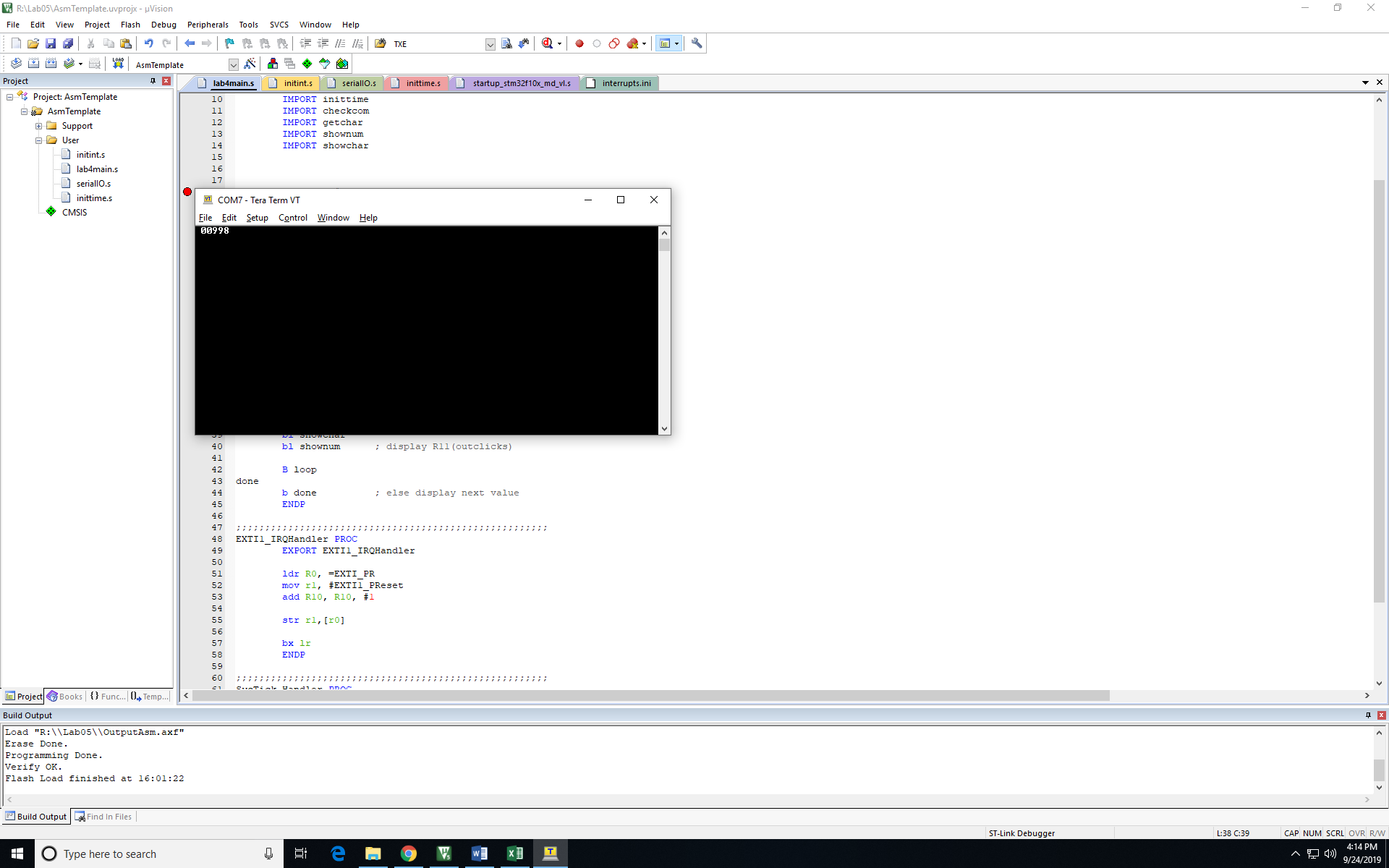
*Fig6. Result using hardware with frequency of 3000Hz*

* 1. Frequency of 4000 Hz



*Fig6. Result using hardware with frequency of 4000Hz*

* 1. Frequency of 5000 Hz



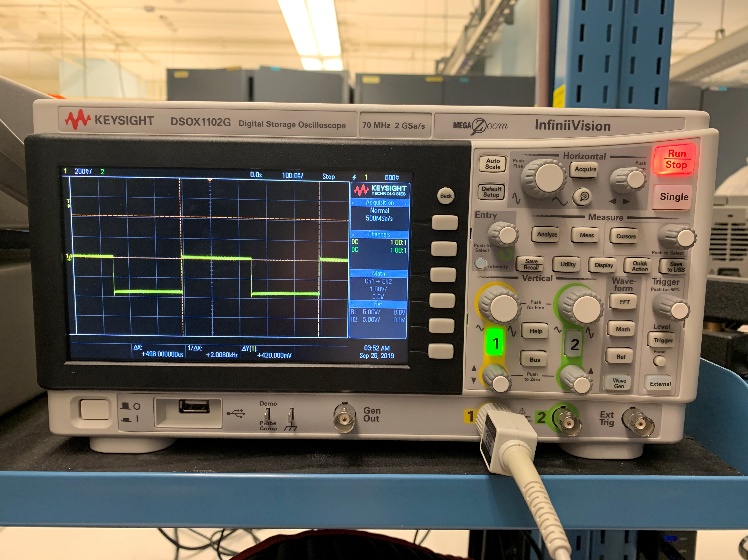
*Fig7. Result using hardware with frequency of 5000Hz*

There is a constant multiple of 2 when we were using hardware to execute the program. Since the linear relationship could be shown, we divided the result by 2 when we are plotting the data.

* Function generator and oscilloscope



*Fig8. Function generator generating 4000Hz signal*



*Fig9. Result from oscilloscope* *with frequency of 2000Hz*

* With computed data, we have used excel to observe the relationship.

|  |  |  |
| --- | --- | --- |
| Frequency (Hz) | USART Result | SILULATOR Result |
| 1000 | 1000 | 22 |
| 2000 | 2000 | 44 |
| 3000 | 3000 | 67 |
| 4000 | 4000 | 88 |
| 5000 | 4990 | 110 |

*Table1. Result of the Hardware and Software simulator*

USART result vs Frequency

Simulator result vs Frequency

Frequency [Hz]

Simulator result

USART Result

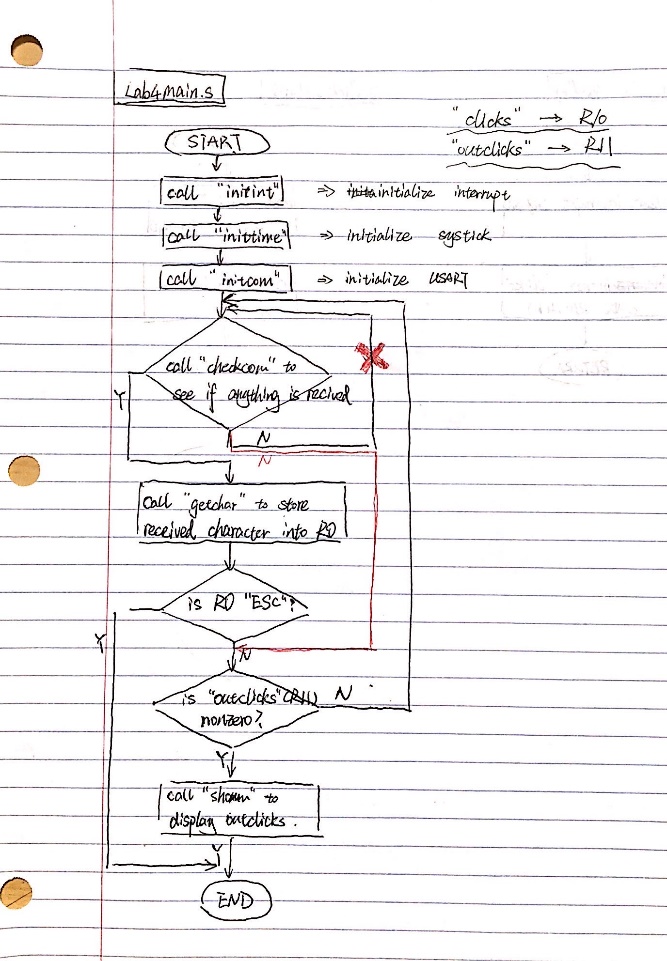
Frequency [Hz]

**Discussion**

From the equation of the chart, R2 value is very close to 1 which indicates that most of the points are close enough to the linear equation and can assume the data is forming linear relationship. Besides, the results we got are almost the same as inputs. Thus, we can safely draw a conclusion that this program has a high accuracy at calculating external clock signal’s frequency and the lab was successful.

However, during the lab, we still met with some troubles. When testing the whole function, we found out that USART window showed nothing, while previous tests on EXIT1 and Systick interrupts all worked fine. By using debug to check memory and register, we found “outclicks” did changed after Systick’s interrupt. However, it didn’t call “shownum” function to display the result as we expected. Finally, we figured out that the structure designed in flowchart is incorrect. We put the statement that checks “outclicks” after “checkcom”, which means that “outclicks” can only be checked if any character is entered in USART. After fixing that bug, we were able to run function normally.

Revised flowchart is shown below. Red line is used for modification.



*Fig10. Flowchart (revised version)*

**Conclusion**

Overall, the lab was very fruitful to be able to understand how the interrupt works and how it can be implemented in the program. Moreover, we were able to understand how to use SysTick timer. We were able to interact with the software (μVision debugger and simulator) to calculate an external clock frequency generated using Agilent Function generator or signal generated from the software. During the lab, we faced weird multiples were applied when using the STM32 to execute the program. However, we were able to see the linear relation between input frequency and the result. We believe by doing this lab, our debugging skill is enhanced. By designing the project using a flowchart, we were able to increase the ability to design an assembly coding. All the assigned tasks were completed and we want to thank the professor and the TA for useful resources and the help.

**Appendix:**

**lab4main.s:**

EXTI\_PR EQU 0x40010414

EXTI1\_PReset EQU 0x02

AREA ARMex, CODE, READONLY

ENTRY

\_\_main PROC

EXPORT \_\_main

IMPORT initcom

IMPORT initint

IMPORT inittime

IMPORT checkcom

IMPORT getchar

IMPORT shownum

IMPORT showchar

MOV R10, #0 ; initialize R10 as clicks

MOV R11, #0 ; initialize R11 as outclicks

bl initcom

bl inittime

bl initint

loop bl checkcom

CMP R0, #0xFF

BEQ receive

B next ; if nothing received, branch to "next"

receive bl getchar ; load received ascii number to R0

CMP R0, #0x1B ; check whether it's 'ESC'

BEQ done ; if it's 'ESC', quit

B loop

next CMP R11, #0 ; check if R11 is 0

BEQ loop ; if R11 is zero

; if R11 is nonzero

mov r0, #0x0D ; clear window

bl showchar ; clear window

bl shownum ; display R11(outclicks)

B loop

done

b done

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

EXTI1\_IRQHandler PROC

EXPORT EXTI1\_IRQHandler

ldr R0, =EXTI\_PR

mov r1, #EXTI1\_PReset ; reset interrupt

str r1,[r0]

add R10, R10, #1

bx lr

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SysTick\_Handler PROC

EXPORT SysTick\_Handler

push {lr}

mov R11, R10

mov R10, #0

pop {lr}

bx lr

ENDP

END

**inittime.s:**

RCC\_APB2ENR EQU 0x40021018 ; enable APB2 clock for USART1 peripheral

GPIOA\_CRL EQU 0x40010800 ; configure port A pin 1

EXTI\_IMR EQU 0x40010400 ; enable external interrupt 1

EXTI\_FTSR EQU 0x4001040C ; configure EXTI1 for falling edge trigger

EXTI\_PR EQU 0x40010414 ; EXTI pending register

NVIC\_ISER0 EQU 0xE000E100 ; enable EXTI1 on NVIC

STK\_CTRL EQU 0xE000E010 ; enable SysTick timer interrupt

STK\_LOAD EQU 0xE000E014 ; configure period for SysTick timer

period EQU 100

periodcnt EQU 24000000/8\*period/1000-1

AREA MyData, DATA, READWRITE ; set memory for uninitialized data

AREA ARMex, CODE, READONLY

ENTRY

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

inittime PROC

EXPORT inittime

push {lr}

PUSH {R0}

PUSH {R1}

ldr R0, =STK\_CTRL

mov R1, #0x3

str R1, [R0]

ldr R0, =STK\_LOAD

ldr R1, =periodcnt

str R1, [R0]

POP {R1}

POP {R0}

pop {lr}

bx lr

ENDP

END

**initint.s:**

RCC\_APB2ENR EQU 0x40021018 ; enable APB2 clock for USART1 peripheral

GPIOA\_CRL EQU 0x40010800 ; configure port A pin 1

EXTI\_IMR EQU 0x40010400 ; enable external interrupt 1

EXTI\_FTSR EQU 0x4001040C ; configure EXTI1 for falling edge trigger

EXTI\_PR EQU 0x40010414 ; EXTI pending register

NVIC\_ISER0 EQU 0xE000E100 ; enable EXTI1 on NVIC

AREA MyData, DATA, READWRITE ; set memory for uninitialized data

AREA ARMex, CODE, READONLY

ENTRY

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

initint PROC

EXPORT initint

push {lr}

PUSH {R0}

PUSH {R1}

; configure APB2ENR, enable APB2 clock for USART1 peripheral

LDR R0, =RCC\_APB2ENR

LDR R1, [R0]

ORR R1, #0x0004

STR R1, [R0]

; configure port A

LDR R0, =GPIOA\_CRL

LDR R1, =0x00000040

STR R1, [R0]

; configure EXTI interrupt mask register

LDR R0, =EXTI\_IMR

LDR R1, =0x0002

STR R1, [R0]

; configure EXTI falling trigger selection register

LDR R0, =EXTI\_FTSR

LDR R1, =0x0002

STR R1, [R0]

; configure NVIC set-enable register

LDR R0, =NVIC\_ISER0

LDR R1, =0x0080

STR R1, [R0]

POP {R1}

POP {R0}

pop {lr}

bx lr

ENDP

END

**serialIO.s:**

numOfChar EQU 6

spaceAscii EQU 0x20

minusAscii EQU 0x2D

RCC\_APB2ENR EQU 0x40021018 ;enable APB2 clock for USART1 peripheral

GPIOA\_CRH EQU 0x40010804 ;configure port A

USART1\_BRR EQU 0x40013808 ;configure USART1 baud rate

USART1\_CR1 EQU 0x4001380C ;enable USART1; set parity & mode

USART1\_SR EQU 0x40013800 ;USART1 status register; register empty flags

USART1\_DR EQU 0x40013804 ;USART1 data register

RXNE EQU 0x0020

TXE EQU 0X0080

AREA MyData, DATA, READWRITE ; set memory for uninitialized data

array SPACE numOfChar\*2 ; allocate 10 bytes for data array

AREA ARMex, CODE, READONLY

ENTRY

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

initcom PROC

EXPORT initcom

push {lr} ;store link register on stack

PUSH {R0}

PUSH {R1}

; configure APB2ENR, enable APB2 clock for USART1 peripheral

LDR R0, =RCC\_APB2ENR

LDR R1, [R0]

ORR R1, #0x4000

ORR R1, #0x0005

STR R1, [R0]

; configure port A

LDR R0, =GPIOA\_CRH

LDR R1, =0x000004B0

STR R1, [R0]

; configure baud rate

; integer:13, fraction:0

LDR R0, =USART1\_BRR

LDR R1, =0xD0

STR R1, [R0]

; configure control register

LDR R0, =USART1\_CR1

LDR R1, =0x200C

STR R1, [R0]

POP {R1}

POP {R0}

pop {lr} ;get link value

bx lr

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

checkcom PROC

EXPORT checkcom

push {lr}

PUSH {R1}

LDR R0, =USART1\_SR

LDR R1, [R0]

ANDS R1, #RXNE

BEQ nochar

MOV R0, #0xFF

B checkdone

nochar MOV R0, #0

checkdone

POP {R1}

pop {lr} ;get link value

bx lr

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

getchar PROC

EXPORT getchar

push {lr}

PUSH {R1}

LDR R1, =USART1\_DR

LDR R0, [R1] ; get data from data register

POP {R1}

pop {lr}

bx lr

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

showchar PROC

EXPORT showchar

push {lr}

PUSH {R2}

PUSH {R1}

LDR R1, =USART1\_SR

wait LDR R2, [R1]

ANDS R2, #TXE ; check both TXE and TC

BEQ wait

LDR R1, =USART1\_DR

STRB R0, [R1] ; send data to data register

POP {R1}

POP {R2}

pop {lr} ;get link value

;b nextSC

bx lr

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bindec PROC

EXPORT bindec

push {lr}

push {R6}

PUSH {R3}

PUSH {R2}

PUSH {R1}

; check positive or negative

ANDS R1, R0, #0x8000

LDR R3, =array

BEQ positive

;if negegative

SXTH R1, R0 ; perform sign extension to 32 bits

RSB R2, R1, #0xFFFFFFFF ; reverse

ADD R0, R2, #1 ; take two's complement of number

MOV R2, #minusAscii

STRB R2, [R3]

B doneCheck

positive

MOV R2, #spaceAscii

STRB R2, [R3]

doneCheck

LDR R3, =array

ADD R3, #5 ; move array pointer to last bit

MOV R11, #5

loop BL getReminder

MOV R5, #0x30

ORR R1, R5

MOV R2, R1

STRB R2, [R3]

SUB R3, #1

SUBS R11, #1

BNE loop

POP {R1}

POP {R2}

POP {R3}

POP {r6}

pop {lr}

bx lr

; bx lr ;return to calling program

ENDP

getReminder PROC

EXPORT getReminder

push {lr}

PUSH {R3}

PUSH {R4}

PUSH {R11}

; input: R0

; output: R1-reminder

; R0-rest

MOV R4, #10

UDIV R3, R0, R4

MUL R11, R3, R4

SUB R1, R0, R11

MOV R0, R3

POP {R11}

POP {R4}

POP {R3}

pop {lr}

bx lr

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

asc2num PROC

push {lr}

PUSH {R11}

; input: R0

; output: R0

MOV R11, #0x0F

AND R0, R11

POP {R11}

pop {lr}

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

num2asc PROC

push {lr}

PUSH {R11}

; input: R0

; output: R0

MOV R11, #0x30

ORR R0, R11

POP {R11}

pop {lr}

ENDP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

shownum PROC

EXPORT shownum

push {lr}

PUSH {R5}

push {r3}

push {r4}

push {r0}

push {r11}

MOV R0, R11

MOV R2, #numOfChar

bl bindec

LDR R3, =array

loopSN LDRB R4, [R3], #1

; MOV R5, #0x30

; ORR R4, R5

MOV R0, R4

BL showchar

SUBS R2, #1

BNE loopSN

pop {r11}

pop {r0}

pop {r4}

pop {r3}

POP {R5}

pop {lr}

bx lr

ENDP

END

**flowchart:**

