ME 586: Lab 3 Report
STM32 Serial Communications

Harsh Savla & TJ Wiegman

2022-09-27

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 convert binary numbers into decimal and then ASCII format, as well as configure a USART and then act on serial input and produce serial output through two GPIO pins on the device.

1 Lab Checkpoints

1.1 Simulation: Test Decimal Conversion and Serial Output

Overview

The first task was to compile and test assembly programs from Homework 3 (Problems 2 and 3) using the simulated debugger.

Procedure

This section tested the `bindec` and `shownum` routines given in Appendix A.2.1 with flow charts given in Figures 4 and 5, respectively. The code was compiled and simulated using Keil µVision. Both a positive number and a negative number were used as inputs, and the outputs were compared against the expected values.

Results and Discussion

Initially there was trouble with the serial connection. After much debugging and careful code review, it was discovered that a value had not been properly set in one of the configuration registers. After correcting that error, this task was completed successfully.

1.2 Simulation: Serial Input and Project Integration

Overview

The next task was to compile and test assembly programs from Homework 3 (Problem 4) using the simulated debugger.

Procedure

This section tested the `keycount` program given in Appendix A.2.2 with flow chart given in Figure 6. The code was compiled and simulated using Keil µVision. Sample data values were injected using the integrated serial window and the output was compared with the expected behavior.

Results and Discussion

The counter accurately incremented upon receiving 's'from serial input window, and decremented upon receiving 'd'. The program ended when 'ESC'was received. One issue cropped up, however, in that the last character written to the serial output register would never display. This issue was solved by "flushing" the output buffer with extra characters–a line feed and a carriage return were chosen for their inconspicuous and aesthetically pleasing appearance when eventually written to the serial connection.

1.3 Hardware: Test the Keycount Program

Overview

The next task was to compile and test assembly programs from Homework 3 (Problem 4) using physical STM32 hardware.

Procedure

This section tested the `keycount` program given in Appendix A.2.2 with flow chart given in Figure 6. The code was compiled with Keil µVision and executed on an STM32F100RB board. Serial inputs were injected using a PC serial terminal called "Tera Term" and the output was compared with the expected behavior.

Results and Discussion

The counter accurately incremented upon receiving an 's' serial input, and decremented upon receiving a 'd' input. The program ended when 'ESC' was received, as expected. However, one issue encountered was that the intial counter value was not always 0, due to random initialization in the RAM that is not modeled in the simulator. The main program was changed to include a step explicitly initializing the counter to zero, and the issue was resolved.

2 Conclusion

This lab taught us how to properly configure several peripherals at once, including the USART system. We learned how serial communications are performed at the assembly level, as well as how to work with ASCII characters.

The most important practice we learned is to carefully review all code for initializing new services on the microcontroller. A small error when, for example, setting up the configuration registers, can cause much confusion and many problems when testing the associated peripheral later on.

A Appendix

A.1 Flow Charts

Figure 1: Flowchart for the `initcom` routine given in Appendix A.2.1.

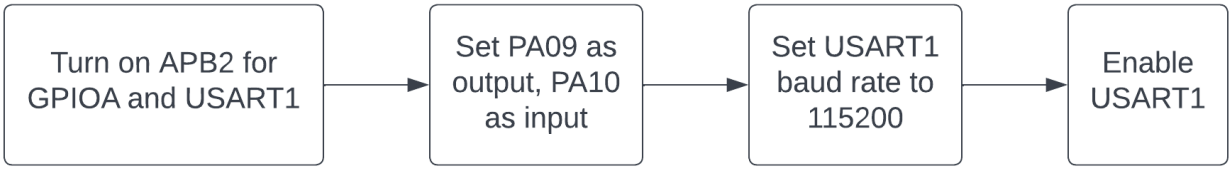


Figure 2: Flowchart for the `checkcom` routine given in Appendix A.2.1.

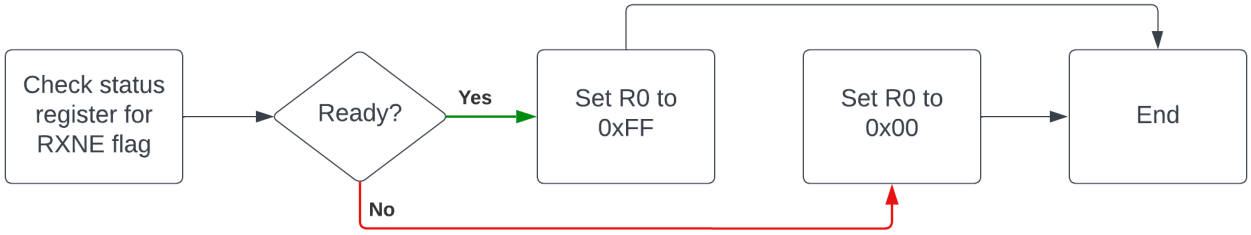


Figure 3: Flowchart for the `showchar` routine given in Appendix A.2.1.

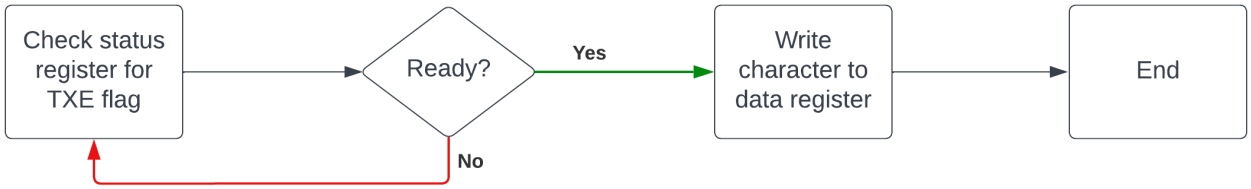


Figure 4: Flowchart for the `bindec` routine given in Appendix A.2.1.

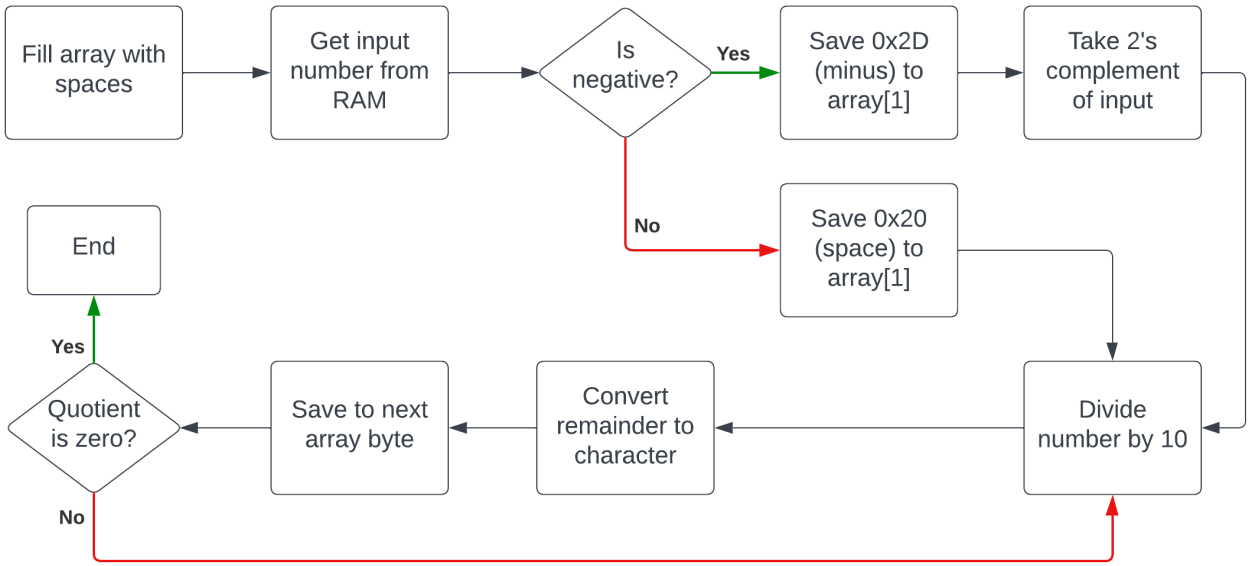


Figure 5: Flowchart for the `shownum` routine given in Appendix A.2.1.

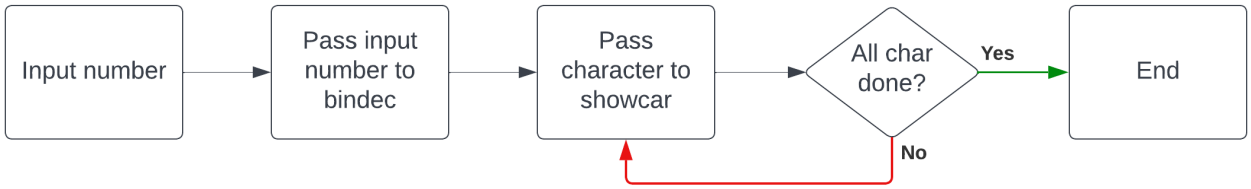
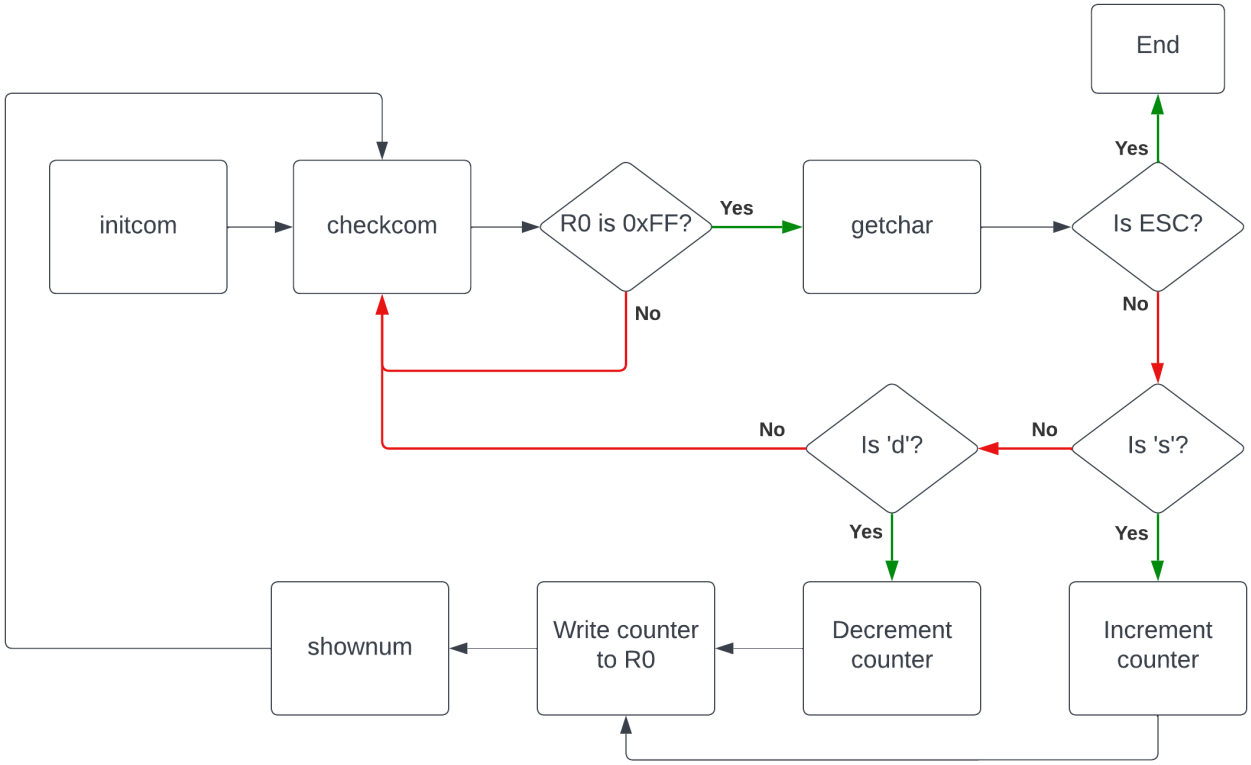


Figure 6: Flowchart for the `keycount` routine given in Appendix A.2.2.



A.2 Code

A.2.1 serialIO.s

```
; Harsh Savla & TJ Wiegman
; ME 58600
; 2022-09-19
; serialIO.s

RCC_APB2ENR EQU 0x40021018 ; enable APB2 clock for USART1
GPIOA_CRH EQU 0x40010804 ; configure PA09 and PA10 for tx/rx
USART1_BRR EQU 0x40013808 ; configure USART1 baud rate
USART1_CR1 EQU 0x4001380C ; enable USART1, set parity, mode
USART1_SR EQU 0x40013800 ; USART1 satus register
USART1_DR EQU 0x40013804 ; USART1 data register

USART1baud EQU 0x00D0 ; hex fraction for setting baud rate

; allocate some RAM for bindec
       AREA MyData, DATA, READWRITE
num3 SPACE 2
array3 SPACE 6

; program code
       AREA ARMex, CODE, READONLY
       ENTRY
initcom PROC ; initializes serial channel 1 for asynchronous communcations
       EXPORT initcom
       ; push LR to stack
       push {LR}

       ; turn on APB2 perhipheral clock
       ldr R3, =RCC_APB2ENR
       ldr R1, [R3] ; save current APB2 state
       orr R1, #0x4000
       orr R1, #0x0005
       str R1, [R3]

       ; configure port A - PA09 output, PA10 input
       ldr R3, =GPIOA_CRH
       ldr R1, =0x444444B4
       str R1, [R3]

       ; configure USART1 baud rate
       ldr R3, =USART1_BRR
       mov R1, #USART1baud ; as close as possible to 115,200 -- impossible to get exact at 24MHz
       str R1, [R3]

       ; enable USART1 for 8 data bits. disable parity and interrupts
       ldr R3, =USART1_CR1
       mov R1, #0x200C
       str R1, [R3]

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

checkcom PROC ; checks to see if a character is in the data receive register
                  ; writes 0xFF to R0 if available, 0x00 otherwise
       EXPORT checkcom
```

```
58          ; push LR to stack
59          push {LR}
60
61          ; check status register
62          ldr R3, =USART1_SR
63          ldr R1, [R3]
64          and R1, #32 ; mask out unneeded flags
65          cmp R1, #32 ; check if RXNE flag is set
66          beq ready
67
68          ; set R0 to 0x00 if not ready
69          mov R0, #0x00
70          b chEnd
71
72          ; set R0 to 0xFF if ready
73  ready mov R0, #0xFF
74
75          ; End subroutine and go back to caller
76  chEnd pop {LR}
77          bx LR
78          ENDP
79
80  getchar PROC ; fetches character from serial channel 1 and writes it as ASCII to R0
81          EXPORT getchar
82          ; push LR to stack
83          push {LR}
84
85          ldr R3, =USART1_DR
86          ldrb R0, [R3]
87
88          ; End subroutine and go back to caller
89          pop {LR}
90          bx LR
91          ENDP
92
93  showchar PROC ; checks that TXE is set, then outputs ASCII character in R0 to serial channel 1
94          EXPORT showchar
95          ; push LR to stack
96          push {LR}
97
98          ; check status register
99  shWait ldr R3, =USART1_SR
100         ldr R1, [R3]
101         and R1, #128 ; mask out unneeded flags
102         cmp R1, #128 ; check if TXE flag is set
103         beq write
104
105         ; if DR is not ready yet
106         b shWait
107
108         ; if DR is ready
109 write ldr R3, =USART1_DR
110         strb R0, [R3]
111
112         ; End subroutine and go back to caller
113         pop {LR}
114         bx LR
115         ENDP
116
117 bindec PROC ; converts a 16-bit signed binary number into five decimal characters (digits)
118             ; preceded by either a space or a dash (minus), depending on sign
```

```
119         EXPORT bindec
120         ; push LR to stack
121         push {LR}
122 
123         ; fill array with spaces
124         mov R1, #1
125         ldr R3, =array3
126         mov R0, #0x20 ; ascii character for " "
127 clrloop strb R0, [R3], #1
128         add R1, #1
129         cmp R1, #7
130         bne clrloop
131 
132         ; get input number from RAM, put into R0
133         ldr R3, =num3
134         ldrh R0, [R3]
135 
136         mov R1, R0
137         lsr R1, #15 ; shift first digit down to LSB
138         cmp R1, #1 ; is negative?
139         beq neg1
140 
141         ; is positive
142         mov R1, #0x20 ; ascii character for " "
143         ldr R3, =array3
144         strb R1, [R3], #5
145         b binloop
146 
147         ; is negative
148 neg1 mov R1, #0x2D ; ascii character for "-"
149         ldr R3, =array3
150         strb R1, [R3], #5
151         sxth R0
152         sub R0, #1
153         eor R0, #0xFFFFFFFF
154 
155         ; divide by 10
156 binloop mov R4, #10
157         udiv R1, R0, R4 ; R1 holds quotient
158         mul R2, R1, R4
159         sub R2, R0, R2 ; R2 holds remainder
160 
161         ; convert to ASCII and store
162         add R2, #0x30
163         strb R2, [R3], #-1
164         mov R0, R1
165         cmp R0, #0
166         bne binloop
167 
168         ; End subroutine and go back to caller
169         pop {LR}
170         bx LR
171         ENDP
172 
173 shownum PROC ; takes binary half-word from R0 and outputs decimal over serial
174         EXPORT shownum
175         ; push LR to stack
176         push {LR}
177 
178         ; Load R0 half-word to RAM for bindec to use it
179         ldr R3, =num3
```

```
        strh R0, [R3]
        bl bindec

        ; Loop through decimal characters until all printed
        mov R2, #0
sloop ldr R3, =array3
        add R3, R2
        ldrb R0, [R3]
        bl showchar
        add R2, #1
        cmp R2, #6
        bne sloop

        ; Write newline afterwards, forces buffer empty
        mov R0, #0x0A
        bl showchar
        mov R0, #0x0D
        bl showchar

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

A.2.2 keycount.s

```
; Harsh Savla & TJ Wiegman
; ME 58600
; 2022-09-19
; keycount.s

; allocate some RAM for bindec
        AREA MyData, DATA, READWRITE
counter FILL 2, 0x00

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

        ; Initialize serial communications
        bl initcom

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

        ; If received, fetch character into R0
        bl getchar

        ; If escape character, end program
        cmp R0, #0x1B ; ASCII for ESC
        beq done
```

```
34
35          ; If 's' goto incrementer
36          cmp R0, #0x73 ; ASCII for 's'
37          beq incr
38          ; If 'd' goto decrementer
39          cmp R0, #0x64 ; ASCII for 'd'
40          beq decr
41          ; Else check next character
42          b chloop
43
44          ; Increment counter
45  incr ldr R3, =counter
46          ldrh R0, [R3]
47          add R0, #1
48          strh R0, [R3]
49          b out0
50
51          ; Decrement counter
52  decr ldr R3, =counter
53          ldrh R0, [R3]
54          sub R0, #1
55          strh R0, [R3]
56
57          ; Output decimal of R0 (counter) over serial
58  out0 bl shownum
59          b chloop
60
61  done b done
62          ENDP
63      END
```