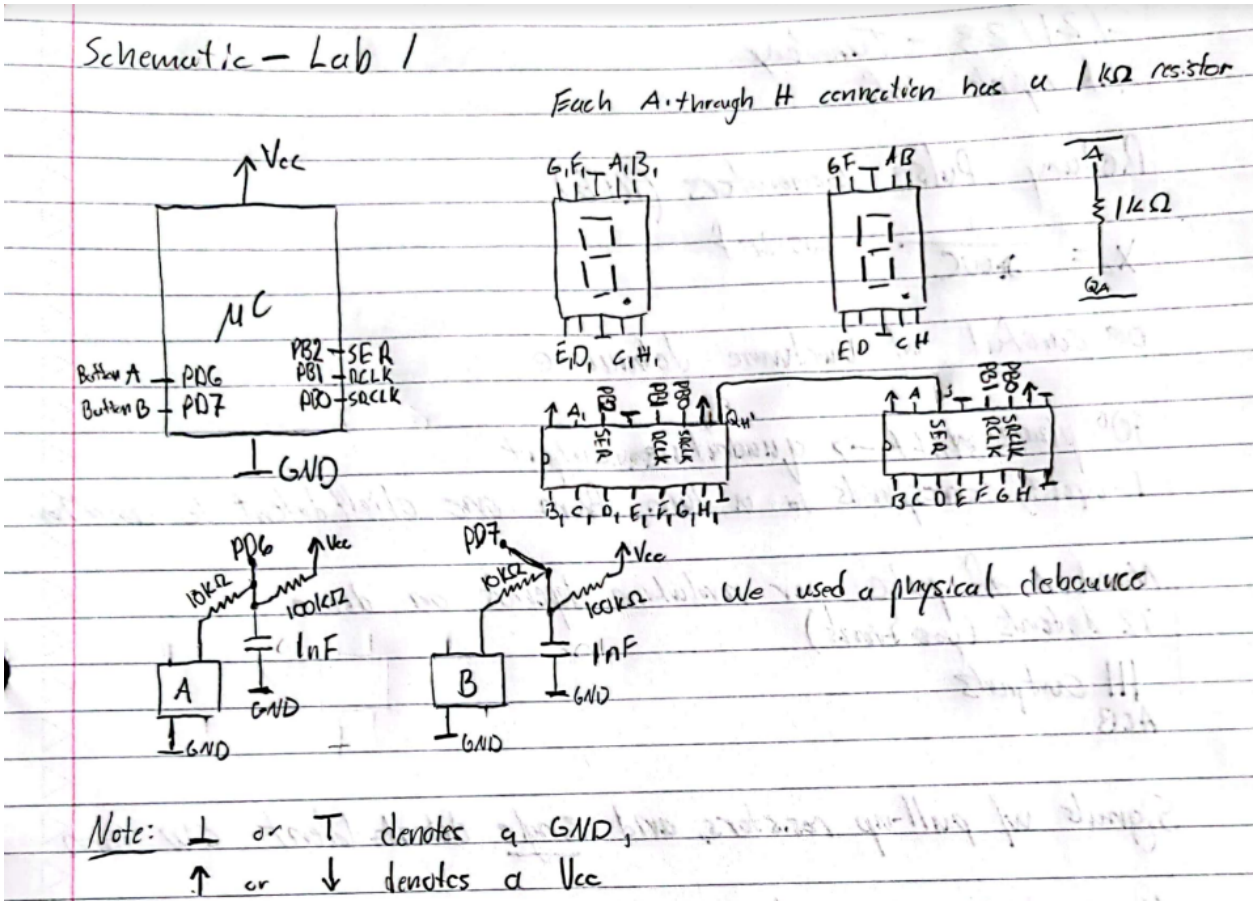
Authors: Brandon Cano, Ian Kuk
Professor Beichel
ECE:3360 Embedded Systems
Post-Lab 1 Report

1. Introduction

The goal of this lab was to create a microcontroller based countdown timer using assembly. We needed to have a button which could increment the timer to a desired number between 0 and 25, with the inability to increment over 25. We also needed to have that same button be able to reset the timer back to zero by holding it for over one second. Then we needed to use a second button which would start the timer and accurately countdown to zero with an animation indicating that it is done, reset, and is ready to go again.

2. Schematic

This image shows a simplified view of how we implemented our circuit. We used the SN74HC595 chips to connect the outputs of the microcontroller to properly light up the LCD 7-segment displays. The inputs and outputs of the microcontroller are labeled to which pins they go to on either the buttons or the chips. The A-H labels on the chips correspond to the A-H labels on the 7-segment display as well, and each one of those connections had a 1kOhm resistor attached to it. For the buttons we used a physical debounce which is shown using a 100kOhm pull-up resistor and a 1nF capacitor.



3. Discussion

When designing the project it was split off into two categories, the hardware and the software. We first started with getting the hardware built starting with the LCD 7-segment displays. In front of each pin, except for pins three and eight, of the common cathode display there is a 1kOhm resistor. The LED's inside optimal current should be around 5mA. Since we are sending in 5V, we can use Ohm's law to find that 5V / 5mA = 1 kOhm and this will get 5mA of current to each diode. Pins three and eight are connected to ground. The wiring of the shift registers followed this pattern. Each pin letter of the register would be wired to the correlating letter on the diode. So Qa on the register would be wired to pin seven of the display, because that pin lights up section a. Qb would be wired to b and so on. The buttons used a physical debounce. The oscilloscope measured a debounce period of twenty nanoseconds so we used a 100kOhm pull up resistor and a 1nF capacitor. This roughly creates a period of 100 ns of debounce. It is a little overkill, but it's an extremely fast time to humans, and it more than insures there will be no unnecessary bouncing.

For the software, we first had to start with getting the display to show a specific pattern that we could configure from a hexadecimal value determined by the hardware setup. Once we had the hexadecimal values for every number, we needed to set up the registers to send the data to the displays. Using the SRCLK to send each bit of information then using RCLK to set the data in the registers can be seen in the display subroutine below.

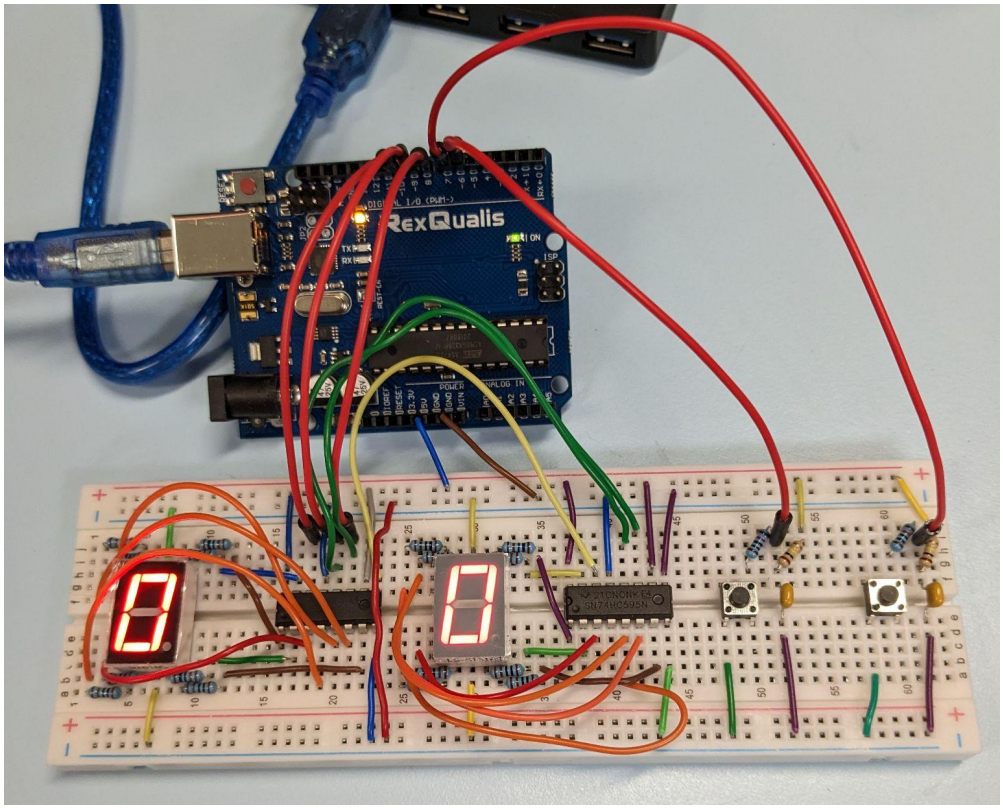
We also had to add button functionality to the circuit. In order to do this we had to set up pins as inputs, then we start an infinite loop that is constantly checking for button presses from either button. During this waiting period we oscillate between the two buttons, constantly checking for a button press so that they can work independently. Once a button is pressed it will detect the one being held and start waiting until a release occurs or until button A is held for over one second, checking with 'sbic' and 'sbis' respectively. Then upon release, or the case of button A being held for over a second, a specific subroutine for the respective button will then compute its task before returning back to the loop to check for another button press.

In order to know which number to set the displays to we set up a few subroutines. Upon each action that requires an update to the displays we change the value in register R29. Register 29 will increase by one for every button press to increase the display, and will reset to zero when the button is held to reset the display. For each of the values 0-25 we have a subroutine that loads the values to registers R16 and R19, to the hexadecimal values for the left and right displays respectively. To determine which of the display subroutines to call we run through a separate subroutine with several branch statements that compare an immediate value to the value in register R29. When it finds a match it branches off to the specific display subroutine that will display the updated value. With having to limit the display to 25, if the end of the branch statements is reached we set the register to the value of 25 to prevent the counter from being able to go above 25. This helps greatly with the timer feature, because when we decrement the timer it uses the same display subroutines.

For the delays we have a few different subroutines all building from the bottom up. We designed a nested looping subroutine and debugged the values in order to get roughly 10 ms. Then we added a 100ms, 500ms, and 1 second delay subroutine which all use the smaller delay to create their own larger delay.

Utilizing the delays we were able to properly implement counting down and resetting the timers. To count down, we have a subroutine that decrements the value in the register R29. Which constantly checks if R29 is greater than or equal to one. We used one instead of zero because if the start timer button

was pressed on zero, it would hit negative values and never break out of the loop. Once the value zero is hit then another subroutine is called to display the flashing dashes to indicate it has finished counting down. To reset the timer we had to use the 10 ms delay in order to count how long the button was held for. Once the button is pressed and is awaiting release, we call the delay subroutine. The idea here is that while it's in this delay routine we increment another register R20 by 1. We compare the value in R20 to 100, since 100 * 10 ms = 1 second. If this value is hit before the button is released then a new subroutine is called, if not then R20 is set back to zero. The subroutine reset_counter that is called sets the values in R20 and R29 to zero, and sets the values in R16 and R19 to the hexadecimal value that displays zero. The display subroutine is then called to produce 0 0 on the displays.



4. Conclusion

From this lab, we both gained experience in coding in assembly and being able to be able to measure values on an oscilloscope to determine what values of resistors or capacitors we needed to use. We also learned how to physically debounce buttons, and how to generate and send clock signals to shift registers so they can take necessary data. On top of that, we learned how to debug both the code and the hardware that we were making.

5. Appendix A: Source Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Lab2.asm
;
; Created: 2/5/2023 1:49:12 PM
; Author : Brandon Cano, Ian Kuk
;
; Notes - I/O pins and registers:
;        8        -> PB0 (SRCLK)
;        9        -> PB1 (RCLK)
;        10       -> PB2 (SER)
;        6        -> PD6 (Button A) - left
;        7        -> PD7 (Button B) - right
;
;        R16 -> left 7 segment display
;        R17 -> SREG
;        ...
;        R19 -> right 7 segment display
;        R20 -> hold to reset counter
;        ...
;        R26 -> full second delay counter
;        R27 -> half second delay counter
;        R28 -> tenth second delay counter
;        R29 -> 7 segment display counter
;        R30 -> outer loop delay counter
;        R31 -> inner loop delay counter
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.include "m328Pdef.inc"
.cseg
.org 0

sbi DDRB, 0                ; PB0 is now output
sbi DDRB, 1                ; PB1 is now output
sbi DDRB, 2                ; PB2 is now output

cbi DDRD, 7                ; PD7 is now input
cbi DDRD, 6                ; PD6 is now input

; display values for seven segment display
.equ zero = 0x3f
.equ one = 0x06
.equ two = 0x5b
.equ three = 0x4f
.equ four = 0x66
.equ five = 0x6d
.equ six = 0x7d
.equ seven = 0x07
.equ eight = 0x7f
.equ nine = 0x6f
.equ dash = 0x40
.equ off = 0x00

; start of main program
ldi R29, 0x00                                          ; counter
ldi R20, 0x00                                          ; hold to reset counter
ldi R16, zero                                          ; left 7-segment
ldi R19, zero                                          ; right 7-segment
rcall display                                          ; call display subroutine
rcall btnloop                                          ; wait for buttons

; subroutines to display values 0-15 and the dashes
show_zero:
         ldi R16, zero
         ldi R19, zero
         rcall display
         ret
```

```
show_one:
         ldi R16, zero
         ldi R19, one
         rcall display
         ret
show_two:
         ldi R19, two
         rcall display
         ret
show_three:
         ldi R19, three
         rcall display
         ret
show_four:
         ldi R19, four
         rcall display
         ret
show_five:
         ldi R19, five
         rcall display
         ret
show_six:
         ldi R19, six
         rcall display
         ret
show_seven:
         ldi R19, seven
         rcall display
         ret
show_eight:
         ldi R19, eight
         rcall display
         ret
show_nine:
         ldi R16, zero
         ldi R19, nine
         rcall display
         ret
show_ten:
         ldi R16, one
         ldi R19, zero
         rcall display
         ret
show_eleven:
         ldi R19, one
         rcall display
         ret
show_twelve:
         ldi R19, two
         rcall display
         ret
show_thirteen:
         ldi R19, three
         rcall display
         ret
show_fourteen:
         ldi R19, four
         rcall display
         ret
show_fifthteen:
         ldi R19, five
         rcall display
         ret
show_dash:
         ldi R16, dash
         ldi R19, dash
         rcall display
         ret
```

```
reset_counter:
         ldi R29, -0x01              ; set value to negative 1 to account for the extra increment
         ldi R20, 0x00               ; reset hold timer
         rcall show_zero             ; reset displays
         ret
display_counter:
         ; compare the counter to the immediate to determine which value to show on the displays
         cpi R29,0x00
         breq show_zero
         ; one
         cpi R29,0x01
         breq show_one
         ; two
         cpi R29,0x02
         breq show_two
         ; three
         cpi R29,0x03
         breq show_three
         ; four
         cpi R29,0x04
         breq show_four
         ; five
         cpi R29,0x05
         breq show_five
         ; six
         cpi R29,0x06
         breq show_six
         ; seven
         cpi R29,0x07
         breq show_seven
         ; eight
         cpi R29,0x08
         breq show_eight
         ; nine
         cpi R29,0x09
         breq show_nine
         ; ten
         cpi R29,0x0a
         breq show_ten
         ; eleven
         cpi R29,0x0b
         breq show_eleven
         ; twelve
         cpi R29,0x0c
         breq show_twelve
         ; thirteen
         cpi R29,0x0d
         breq show_thirteen
         ; fourteen
         cpi R29,0x0e
         breq show_fourteen
         ; fifthteen
         cpi R29,0x0f
         breq show_fifthteen
         ; branch for 16-25 to avoid "relative branch out of reach error"
         cpi R29, 0x10
         brge to_twentyfive
         ; go back to waiting
         ret

; this subroutine will check button presses
btnloop:
         wait_for_button:
                  sbic PIND, 6                                ; check for button press on the increment/reset button
                  rjmp wait_for_button_countdown              ; continue to wait if not pressed
                  rcall button_inc_pressed                    ; call button pressed routine when pressed
                  rjmp btnloop                                ; restart loop
         button_inc_pressed:
```

```
                rcall delay                         ; start delay to check in on time for button hold
                inc R20                             ; increment the counter for every 10ms the button is held for
                cpi R20, 0x64                       ; compare if it has been held for a 1 second (100, 10ms cycles)
                brge reset_counter                  ; call this to reset the counter if held for a second
                sbis PIND, 6                        ; check for button release
                rjmp button_inc_pressed             ; continue to wait for button release
                ldi R20, 0x00                       ; reset hold timer
                inc R29                             ; increment the counter
                rcall display_counter               ; update display
                rjmp wait_for_button                ; continue waiting for button
        wait_for_button_countdown:
                sbic PIND, 7                        ; check for the right button to be pressed
                rjmp wait_for_button                ; wait for button otherwise
                rcall button_countdown_pressed      ; call the pressed subroutine
                rjmp btnloop                        ; reset loop
                ret
        button_countdown_pressed:
                sbis PIND, 7                        ; check for button release
                rjmp button_countdown_pressed       ; continue to wait for button release
                rcall delay_tenth_second            ; add a slight delay so the change doesn't seem immediate
                rcall countdown_timer               ; start countdown upon release
                rjmp wait_for_button                ; resume waiting
ret

countdown_timer:
        dec R29                                     ; decrement from where the timer was set to
        rcall display_counter                       ; update display
        rcall delay_second                          ; wait roughly a second
        ; compare to a value of 1, this prevents breaking the circuit when pressing the countdown on 0
        cpi R29, 0x01
        brge countdown_timer                        ; while its greater than or equal to 1, keep decrementing
        rcall end_animation                         ; upon completion show the end animation
        ret

; another subroutine to display 16-25, this prevents the relative branch out of reach error
to_twentyfive:
        ; sixteen
        cpi R29,0x10
        breq show_sixteen
        ; seventeen
        cpi R29,0x11
        breq show_seventeen
        ; eighteen
        cpi R29,0x12
        breq show_eighteen
        ; nineteen
        cpi R29,0x13
        breq show_nineteen
        ; twenty
        cpi R29,0x14
        breq show_twenty
        ; twenty one
        cpi R29,0x15
        breq show_twentyone
        ; twenty two
        cpi R29,0x16
        breq show_twentytwo
        ; twenty three
        cpi R29,0x17
        breq show_twentythree
        ; twenty four
        cpi R29,0x18
        breq show_twentyfour
        ; twenty five
        cpi R29,0x19
        breq show_twentyfive
        ; prevent further incrementing
        ldi R29, 0x19
```

```
          ret
; show values 16-25, and nothing
show_sixteen:
          ldi R19, six
          rcall display
          ret
show_seventeen:
          ldi R19, seven
          rcall display
          ret
show_eighteen:
          ldi R19, eight
          rcall display
          ret
show_nineteen:
          ldi R16, one
          ldi R19, nine
          rcall display
          ret
show_twenty:
          ldi R16, two
          ldi R19, zero
          rcall display
          ret
show_twentyone:
          ldi R19, one
          rcall display
          ret
show_twentytwo:
          ldi R19, two
          rcall display
          ret
show_twentythree:
          ldi R19, three
          rcall display
          ret
show_twentyfour:
          ldi R19, four
          rcall display
          ret
show_twentyfive:
          ldi R19, five
          rcall display
          ret
show_nothing:
          ldi R16, off
          ldi R19, off
          rcall display
          ret

; this subroutine will display a value on the 7-segment display
display:
          push R16                              ; add the left display
          push R19                              ; add the right display
          push R17
          in R17, SREG
          push R17
          ldi R17, 16                           ; loop --> test all 16 bits, both displays
loop:
          rol R16                               ; rotate left through left display carry
          rol R19                               ; rotate left through right display carry
          BRCS set_ser_in_1                     ; branch if Carry is set
          cbi PORTB,2
          rjmp end
set_ser_in_1:
          ; set SER to 1
          sbi PORTB,2
end:
```

```
          ; generate SRCLK pulse
          sbi PORTB,0
          cbi PORTB,0
          dec R17
          brne loop
          ; generate RCLK pulse
          sbi PORTB,1
          cbi PORTB,1
          ; restore registers from stack
          pop R17
          out SREG, R17
          pop R17
          pop R19
          pop R16
          ret

end_animation:
          ; the animation needs to flash 4 times over a course of 2 seconds
          rcall show_flash
          rcall show_flash
          rcall show_flash
          rcall show_flash

          rcall show_zero             ; reset display
          ldi R29, 0x00               ; reset count to zero (prevents an incorrect value from showing)
          ret

show_flash:
          rcall show_dash             ; turn on
          rcall delay_half_second     ; wait
          rcall show_nothing          ; turn off
          rcall delay_half_second     ; wait
          ret

; The following subroutines below are all delays of different lengths
; each delay is built upon the 10ms delay
; this is roughly 10ms (9.98428 ms)
delay:
          ldi R30, 0xd0               ; outer loop
d1:
          ldi R31, 0xff               ; inner loop
d2:
          dec R31                     ; decrement inner loop
          brne d2                     ; when inner loop is not zero branch back to d2
          nop
          dec R30                     ; decrement outer loop
          brne d1                     ; if outer loop is not zero branch back to d1 to reset inner loop
          ret

; this is roughly 100ms (99.977 ms)
delay_tenth_second:
          ldi R28, 0x0a
d10:
          nop
          rcall delay
          nop
          dec R28
          brne d10
          ret

; this is roughly 500ms (499.891 ms)
delay_half_second:
          ldi R27, 0x05
d50:
          nop
          rcall delay_tenth_second
          nop
          dec R27
```

```
          brne d50
          ret

; this is roughly 1000ms (999.784 ms)
delay_second:
          ldi R26, 0x02
d100:
          nop
          rcall delay_half_second
          nop
          dec R26
          brne d100
          ret
.exit
```