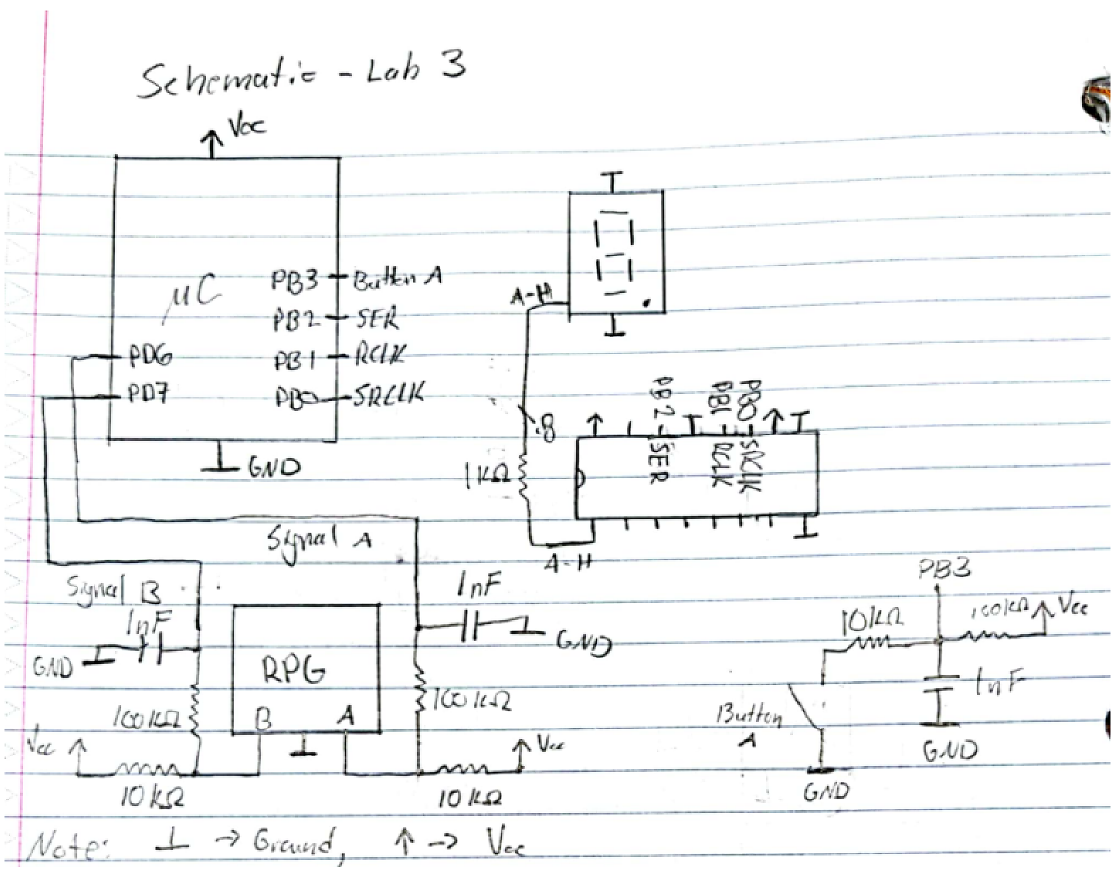
Authors: Brandon Cano, Ian Kuk
Professor Beichel
ECE:3360 Embedded Systems
Post-Lab 3 Report

1. Introduction

The goal of this lab was to create a microcontroller based lock with a passcode utilizing a rotating pulse generator (RPG) in assembly. The RPG has the ability to scroll from 0 – F (0 - 15), without going over or under. A button was also implemented in order to enter in each digit of the passcode on top of being able to reset the entered code. We also needed to be able to match the user's attempt with the correct code and perform an action depending if it was right or wrong. Finally we needed to use the built in timers of the microcontroller to create consistent time delays.

2. Schematic

This image shows a simplified view of how we implemented our circuit. We used a SN74HC595 chip to connect the outputs of the microcontroller to show values on the 7-segment display. The inputs and outputs of the microcontroller are labeled to which pins they go to for all the components. For the display and the chip, there is one wire labeled A-H, denoting how all 8 connections are set up. For both the button and the RPG we used a physical debounce which is shown using 100kΩ pull-up resistors and 1nF capacitors.



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 display. In front of each pin, except for pins three and eight, of the common cathode display there is a 1kΩ 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 kΩ 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.

For the button, a physical debounce was used. The oscilloscope measured a debounce period of twenty nanoseconds so we used a 100kΩ pull up resistor and a 1nF capacitor. This roughly creates a period of 100 ns of debounce. We choose these values for the components because it's an extremely fast time to humans, and it more than ensures there will be no unnecessary bouncing.

To debounce the RPG we looked to the data sheet where it recommended a 10kΩ pull up resistor with a .01µF capacitor. We encountered a problem of not having a capacitor at that capacitance. To counteract this problem we used a 1nF capacitor in place and replaced the pullup resistor with a 100kΩ resistor instead. By decreasing the capacitor and increasing the resistor by the same magnitude we were able to have the factory recommended debounce solution.

For the software, we first had to start with getting a single 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 value 0 - F, dot (.), underscore (_), and dash (-), we needed to set up the registers to send the data to the display. 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. After that setup, we then manually tested each value, to determine that we had all the correct symbols to be shown.

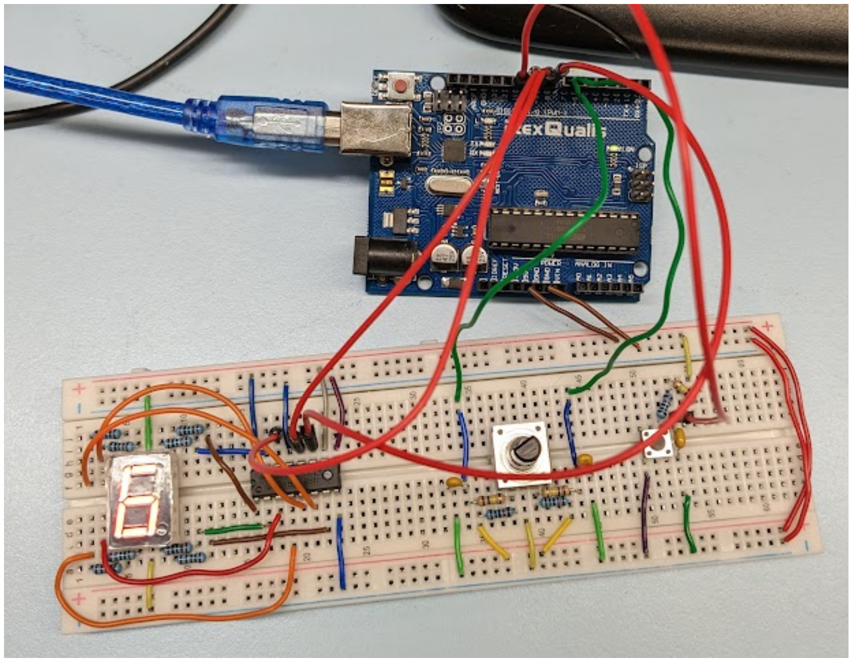
In order to update the seven-segment display with different values we had to utilize an RPG which can rotate through all the different values we can select from. To start, we enter an infinite loop, and check the input from PIND into the register R18, this checks the values from all 8 pins in its section, but only two of them are being used, PD6 and PD7. To ensure we have only the values we need, an 'andi' operation is used with the value 0b11000000 to only keep the two important bits. For the proper matching, the most recently obtained values need to be the two rightmost bits, also known as the least significant bits. The 'lsr' operation is used six times to move the newly recorded bits. We then compare our temporary value in R22 to R18, if a change is recorded, meaning not equal, then we branch off to compare the sequence. If nothing changed, meaning equal, we go back to the start of the loop. If a change is detected, we then move the value in R18 to R22. R23 is used to hold the entire sequence, so we then shift the values in R23 over by two bits, to make room for our new values. Then an 'or' operation is used to combine the two values into R23, which now has a sequence started. At this point there are two comparison checks, one for a clockwise turn (0b01001011) and one for a counter-clockwise turn (0b10000111). If the current sequence in R23 does not match, jump back to the main loop and continue to grab values as the rotation is taking place. If one of the values does match, the respective subroutines are called to change the value that will show on the seven-segment display.

To update the value on the seven-segment display, we had a counter R29 which would either be incrementing or decrementing depending on which direction the RPG was turning. Once a change in R29 occurred, then we had a subroutine with branches that compares all the numerical values and would match the value then call an additional subroutine which would load and display the values.

We also had to add button functionality to the circuit. In order to do this we had to set up a pin as an input. In the main loop we check for both the RPG rotation and a button press. Once a button is pressed it will start waiting until a release occurs, checking with 'sbic' and 'sbis' respectively. If the button has been released then the value will be saved into the attempt to unlock the code. If the button is held for at least 2 seconds then is released the whole thing essentially resets by going back to the dash and the user can restart the password attempt. To track the two seconds, during the time frame in which the button is being held, we use a 10ms delay and a counter. Every 10ms interval we increment the counter register, R19. Once R19 reaches a hexadecimal value of 0xC8, then the 2 second mark has been reached and the comparison will become true for branching. However, a problem was encountered when doing this, if the button was held down for too long (~4 seconds) then an overflow happens in R19, which prevents the user from resetting the passcode attempt. In order to overcome this issue, we had to add a check which would stop incrementing the counter, once it reached the value. We went with this fix because we knew that once the 2 second mark had been hit, we didn't need to count anymore, since we knew the condition was met and can't be changed.

In the lab we needed to implement a five second timer if the password entered was correct, and a nine second timer if the password was incorrect. The problem with this was that the micro-controllers internal timer cannot do times that high. To solve this problem we used the micro-controllers internal timer to make a 10ms delay. This was achieved by setting TCCR0B to 0x05, this set the timer's prescaler to 1024. The next step was to find the value to output into TCNT0. The first step was to find the tmax, this was found by taking the microcontrollers frequency multiplying it by the prescaler and 256. The microcontrollers frequency is 16mHz so the equation looked like (1/16,000,000)*1024*256, resulting in a tmax of 16.384 ms. The next step is to find n, to find n we divide the time we want the timer to be by tclk. To find tclk we divide tmax by 256. The equation for tclk is 16.384ms/256 giving 64 µs. With tclk found we divided the desidered time of 10ms by 64µs to give a value of 156.25. This value has to be rounded so we chose to round up, giving n the value of 157. The final step was to load 256 minus n,which is 99, into TCNT0. With TCNT0's value set we simply repeatedly loop a subroutine that checks its value till it is 0, thus creating a 10ms timer. To create the five and nine second timers we created loops that would call the 10ms delay 500 and 900 times respectively.

In order to match the password detection, we had to be able to save the 5 digit password attempt and be able to compare it to the correct code. For the correct values, 5 variables were set for the five digit password in order named as {number}_digit. To store the values there were five registers used, R24 - R28. Every time a button is pressed to add a digit to the passcode, a subroutine is called which checks each register starting from the first one to see if the initial value of 0x10 has been changed. This value was chosen because it is one integer higher than the max value allowed on the selection. If the value has not been modified then a subroutine is called to store the currently selected value into that register, which then will not be able to be further modified until a reset happens. After all five digits are set, then a comparison happens for all five registers. The moment a value does not match, a fail case subroutine is called, which shows an underscore on the display, and after 9 seconds the program resets back for the user to try again. If all five values match then a success case is called, which will show a dot and turn on a LED on the microcontroller for 5 seconds, then reset. After either subroutine call, the registers will reset and be able to be used again.



4. Conclusion

From this lab, we both learned how to match the sequence of a RPG in order to determine the direction someone is twisting it and how to perform an action unique to the direction. On top of that, we learned how to also set up an RPG on the hardware side while using a physical debounce approach. Finally we learned how to use the built in timers of theATMega328P

5. Appendix A: Source Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Lab3.asm
;
; Created: 2/25/2023 3:11:30 PM
; Author : Brandon Cano, Ian Kuk
;
; Notes - I/O pins and registers:
;        6   -> PD6 - signal b
;        7   -> PD7 - signal a
;        8   -> PB0 (SRCLK)
;        9   -> PB1 (RCLK)
;        10  -> PB2 (SER)
;        11  -> PB3 (Button)
;
;        R16 -> holds the value for the 7 segment display
;        R17 -> SREG
;        R18
;        R19 -> button press
;        R20 -> timer
;        R21 -> delay counters
;        R22 -> store current RPG value
;        R23 -> store current RPG sequence
;        R24 -> 1st hex number
;        R25 -> 2nd hex number
;        R26 -> 3rd hex number
;        R27 -> 4th hex number
;        R28 -> 5th hex number
;        R29 -> display value counter
;        R30
```

```
;         R31
;
;   Code - Group 12 - 7E0E2
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.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 DDRB, 3                 ; PB3 is now input
sbi DDRB, 5                 ; PB5 is now output

cbi DDRD, 6                 ; PD6 is now input
cbi DDRD, 7                 ; PD7 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 a = 0x77
.equ b = 0x7c
.equ c = 0x39
.equ d = 0x5e
.equ e = 0x79
.equ f = 0x71
.equ dot = 0x80
.equ dash = 0x40
.equ off = 0x00
.equ underscore = 0x08

; code values - 7e0e2
.equ first_digit = 0x07
.equ second_digit = 0x0e
.equ third_digit = 0x00
.equ fourth_digit = 0x0e
.equ fifth_digit = 0x02

; timer setup
ldi R20, 0x05
out TCCR0B, R20

; start of main program
main:
         ; load the timer counter to 0
         ldi R19, 0x00
         ; load in values for pattern detection with RPG
         ldi R22, 0b00000011
         ldi R23, 0b00000011
         ; load values for registers saving values, set a value above the max
         ldi R24, 0x10
         ldi R25, 0x10
         ldi R26, 0x10
         ldi R27, 0x10
         ldi R28, 0x10
         ; start with the counter being -1 so when the CW turn occurs it goes to 0
         ldi R29, -0x01
         ldi R16, dash
         rcall display
```

```
; main loop for checking for inputs from the RPG and the button
main_loop:
        in R18, PIND                                ; read entire input
        andi R18, 0b11000000                        ; keep only the two MSBs
        lsr R18                                     ; shift the input from MSBs to LSBs
        lsr R18
        lsr R18
        lsr R18
        lsr R18
        lsr R18
        cp R18, R22                                 ; compare to the previously stored value
        brne compare                                ; if the value is not the same, we branch since we moved onto the
next value in the sequence

        wait_for_button:
                sbic PINB, 3                ; wait for button press
                rjmp main_loop              ; go back to main loop if not pressed
                rjmp button_pressed         ; if pressed go to next routine
                rjmp main_loop
        button_pressed:
                rcall delay_10ms            ; start timer counting
                cpi R19, 0xc9               ; compare to one value above the 2 second mark
                breq skip_increment         ; if equal we don't want to count anymore, this prevents an overflow error
                inc R19                     ; otherwise, we continue to increment the counter
                skip_increment: nop         ; used to skip the incrementing
                sbis PINB, 3                ; detects button release
                rjmp button_pressed         ; jump back to button press subroutine if not released
                cpi R19, 0xc8               ; compare the register to the immediate value to determine ~2 seconds
                brsh main                   ; if held for 2 seconds or more, reset
                rcall save_value            ; otherwise go and perform the button action
                ldi R29, 0x00               ; reset back to zero as an identifier to know something happened
                rcall update_display        ; update display
        rjmp main_loop                      ; go back to main loop
/*
 * This block will compare the current sequence that is recorded and depending if there is
 * a match it will know which direction the RPG is turning and perform the correct action.
 */
compare:
        mov R22, R18                                ; R22 <- R18
        lsl R23                                     ; move the previous value to the left
        lsl R23
        or R23, R18                                 ; add in the current part of the pattern
        cpi R23, 0b01001011                         ; clockwise pattern
        breq increase_counter
        cpi R23, 0b10000111                         ; counter clockwise pattern
        breq decrease_counter
        rjmp main_loop                              ; if no match go back to main loop and continue sequence
increase_counter:
        ldi R23, 0b00000011         ; reset sequence register
        inc R29                     ; increment counter register
        rcall update_display        ; update display
        rcall display
        rjmp main_loop              ; jump back to main loop
decrease_counter:
        ldi R23, 0b00000011         ; reset sequence register
        cpi R16, dash               ; check to see if the display is showing a dash
        brne decrease               ; if it is not, decrease as usual
        ldi R29, -0x01              ; otherwise, set the counter to -1, so when a right turn happens it goes to 0
        rjmp main_loop              ; jump back to the start of the loop
decrease:
        dec R29                     ; decrement counter register
        rcall update_display        ; update display
        rcall display
        rjmp main_loop              ; jump back to main loop
/*
 * This block contains the subroutines for checking which value the seven segment should be set to
 * before the display subroutine is called.
```

```
 * We do this by comparing the counter to the immediates to determine which value to show.
 */
update_display:
        ; zero
        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
        ; a
        cpi R29,0x0a
        breq show_a
        ; b
        cpi R29,0x0b
        breq show_b
        ; c
        cpi R29,0x0c
        breq show_c
        ; d
        cpi R29,0x0d
        breq show_d
        ; e
        cpi R29,0x0e
        breq show_e
        ; f
        cpi R29,0x0f
        breq show_f
        ; prevent from counting above 15(f)
        cpi R29, 0x0f
        brge upper_count
        ; prevent from counting below 0
        cpi R29, 0x00
        brlt lower_count
        ret
upper_count:
        ldi R29, 0x0f
        ret
lower_count:
        ldi R29, 0x00
        ret
; load and display each possible value needed
show_zero:
        ldi R16, zero
        rcall display
```

```
        ret
show_one:
        ldi R16, one
        rcall display
        ret
show_two:
        ldi R16, two
        rcall display
        ret
show_three:
        ldi R16, three
        rcall display
        ret
show_four:
        ldi R16, four
        rcall display
        ret
show_five:
        ldi R16, five
        rcall display
        ret
show_six:
        ldi R16, six
        rcall display
        ret
show_seven:
        ldi R16, seven
        rcall display
        ret
show_eight:
        ldi R16, eight
        rcall display
        ret
show_nine:
        ldi R16, nine
        rcall display
        ret
show_a:
        ldi R16, a
        rcall display
        ret
show_b:
        ldi R16, b
        rcall display
        ret
show_c:
        ldi R16, c
        rcall display
        ret
show_d:
        ldi R16, d
        rcall display
        ret
show_e:
        ldi R16, e
        rcall display
        ret
show_f:
        ldi R16, f
        rcall display
        ret
/*
 * This block of code will store any values after the button press and
 * will also check the comparison of all 5 digits once they are all selected
 */
save_value:
        ldi R19, 0x00                               ; reset button hold counter back to zero
        cpi R24, 0x10                               ; check register one
```

```
        breq set_register_one                          ; if equal, then we set a value to the register
        cpi R25, 0x10                                  ; check register two
        breq set_register_two                          ; if equal, then we set a value to the register
        cpi R26, 0x10                                  ; check register three
        breq set_register_three                        ; if equal, then we set a value to the register
        cpi R27, 0x10                                  ; check register four
        breq set_register_four                         ; if equal, then we set a value to the register
        cpi R28, 0x10                                  ; check register five
        breq set_register_five                         ; if equal, then we set a value to the register
        ret
set_register_one:
        mov R24, R29                         ; move over the number from the counter
        ret
set_register_two:
        mov R25, R29                         ; move over the number from the counter
        ret
set_register_three:
        mov R26, R29                         ; move over the number from the counter
        ret
set_register_four:
        mov R27, R29                         ; move over the number from the counter
        ret
set_register_five:
        mov R28, R29                         ; move over the number from the counter
compare_values:                              ; after register five has been set, go right into comparing the value
        ; we check each value one at a time, if any are not equal then a fail case will occur
        cpi R24, first_digit
        brne fail_case
        cpi R25, second_digit
        brne fail_case
        cpi R26, third_digit
        brne fail_case
        cpi R27, fourth_digit
        brne fail_case
        cpi R28, fifth_digit
        brne fail_case
        rjmp success_case                    ; if all comparisons are equal, then a success case is called
fail_case:
        ldi R16, underscore                  ; show '_'
        rcall display
        rcall delay_nine                     ; wait 9 seconds
        rjmp main                            ; reset back to main
success_case:
        ldi R16, dot                         ; show '.'
        rcall display
        sbi PORTB, 5                         ; turn the LED on the microcontroller on
        rcall delay_five                     ; wait five seconds
        cbi PORTB, 5                         ; turn off the LED
        rjmp main                            ; reset back to main
/*
 * display - updates the seven segment display
 */
display:
        push R16                             ; add the display
        push R17
        in R17, SREG
        push R17
        ldi R17, 8                           ; loop --> test all 8 bits, both displays
loop:
        rol R16                              ; rotate left through 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 R16
        ret

/*
 * delay_10ms - uses the timer in order to count 10ms
 * delay_five - uses the 10ms for a 5 second delay
 * delay_nine - uses the both the 10ms and 5s delays for a 9 second delay
 */
delay_10ms:
        ldi R20, 0x63
        out TCNT0, R20
wait:
        in R20, TCNT0
        cpi R20, 0x00
        brne wait
        ret

delay_five:
        ldi R21, 0xea
loopFiveOne:
        rcall delay_10ms
        dec R21
        cpi R21, 0x00
        brne loopFiveOne
        ldi R21, 0xea
loopFiveTwo:
        rcall delay_10ms
        dec R21
        cpi R21, 0x00
        brne loopFiveTwo
        ret

delay_nine:
        rcall delay_five
        ldi R21, 0xff
loopNineOne:
        rcall delay_10ms
        dec R21
        cpi R21, 0x00
        brne loopNineOne
        ldi R21, 0x8c
loopNineTwo:
        rcall delay_10ms
        dec R21
        cpi R21, 0x00
        brne loopNineTwo
        ret
.exit
```