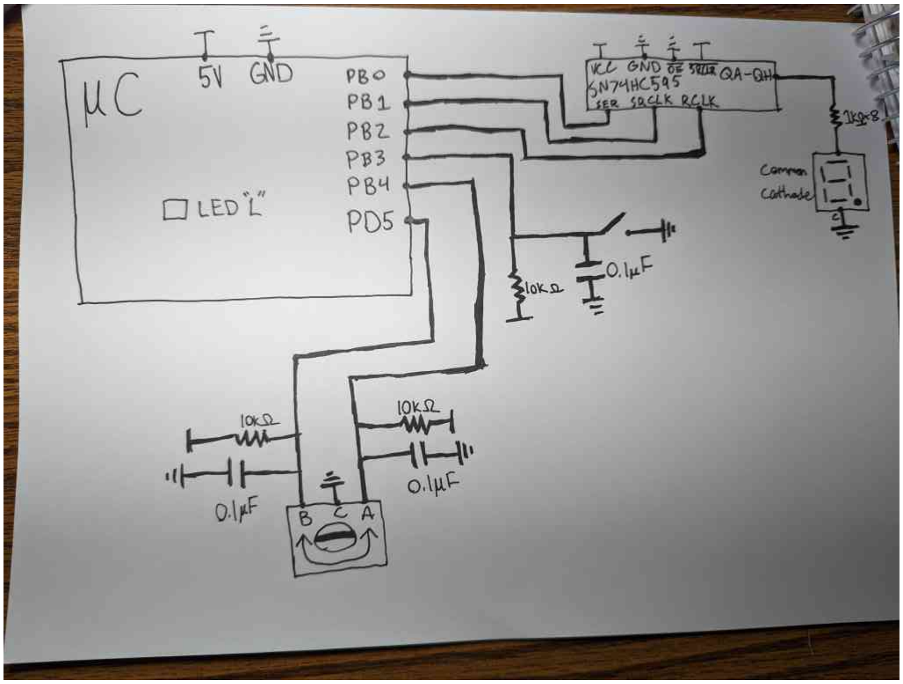
Authors: Max Finch, Tiger Slowinski
Team Members: Max Finch, Tiger Slowinski
ECE:3360 Embedded Systems
Post-Lab Report 3

1. Introduction

The goal of this lab is to create a lock system with a seven-segment display, push-button switch, RPG (Rotary Encoder), shift register, and ATmega328P. The RPG allows a user to increment and decrement from 0-9 and A-F where 9 jumps to A when reached. When a user presses the button the current value represented by the seven-segment display is recorded. The system waits for 5 button presses to evaluate whether the 5 character code entered was correct. If it is correct, the decimal on the seven-segment display will turn on as well as an LED on the Arduino Board for 5 seconds. If the passcode is incorrect an underscore will appear on the seven-segment display for 9 seconds. After either a correct or incorrect passcode's animation has finished the system can accept a 5 character code again starting at a dash. The correct code for this implementation is *E859A*.

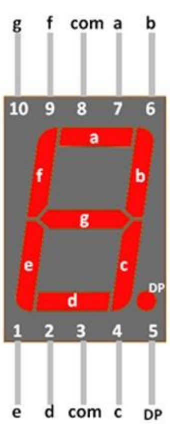
2. Schematic

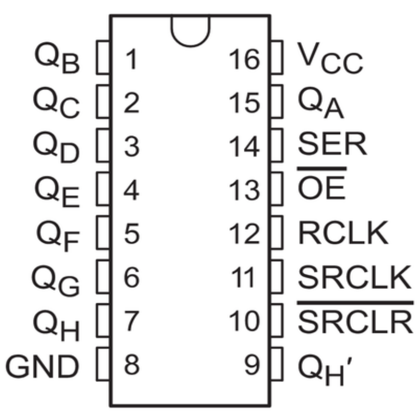


3. Discussion

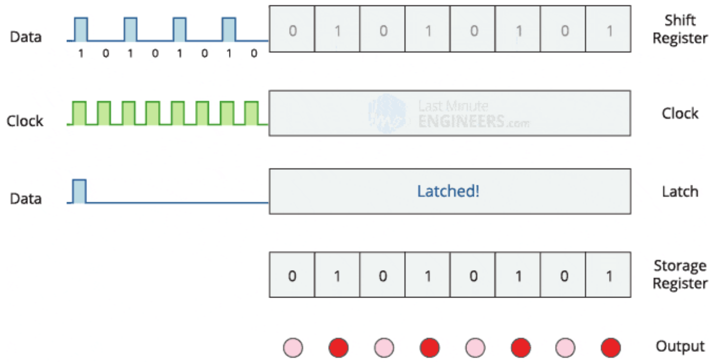
Seven-Segment Display and Shift Register for Data Representation

The key components to get human-recognizable numbers on the screen is with the *74HC595* shift register IC and *Seven-Segment Display* shown below. QA-QH are outputs for single bits. Depending on whether SER is set to 1 or 0, when SRCLK is pulsed, the bit at SER will be shifted into the register and all current values of the register will be shifted to the right. It is only until RCLK is pulsed that the outputs of the register are updated with the current values of the shift register. This can be seen in the timing diagram below. Pins that represent A-G and the decimal point are connected to the register (where A on the seven segment is connected to QA on the shift register, B is connected to QB….DP is connected to QH). Unique hexadecimal values are used to represent the bit pattern to display a character, but the hex values differ depending on how you chose to wire



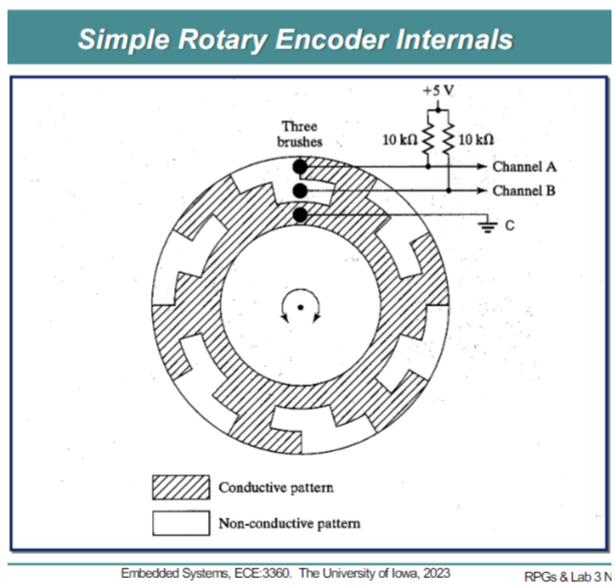


the display to the shift register. For our implementation, a hex value of 0x79 (or 1111001 in binary) represents the letter "E" so A, D, E, F, G are on and B, C and DP are off. **Please note:** **"1111001"** in our implementation corresponds with pins **"(DP)GFEDCBA"** and **"Q: (HGFEDCBA)"** on the Seven-Segment Display and Shift Register respectively. Pin OE is set to low to ENABLE the output since its input is inverted. SRCLR is set high to DISABLE clearing of register data since the input is also inverted. QH' is an output pin used to cascade shift registers which is not used in this lab.



Rotary Pulse Generator (RPG) for User Input

So how do we switch between numbers/characters on the lock? We use an RPG.



The RPG has a series of conductive and non-conductive ridges. To take in the physical input as a signal, the RPG uses 2 channels and one ground pin. Channels A and B act like pushbuttons, where they can either send a 1 or 0 to the Arduino if it makes contact with the conductive or non conductive ridges respectively. As Channels A and B function like pushbuttons, we must decouple the inputs with capacitors (0.01 μF), and set the button unpressed state to either Logic HIGH or Logic LOW with pull up resistors (10KΩ). Our implementation uses pull-up resistors, meaning Channels A and B are always at Logic HIGH when not pressed, and Logic LOW when pressed.

So how do we use this to detect clockwise/counterclockwise turns to determine what value to display? Channels A and B are read at the same time, which means distinct AB patterns can tell us the direction of the turn. For our implementation (due to the pull-up resistors, a clockwise turn has an AB combination of (11 >> 01>>11>>10>>11), and counter-clockwise turn (11 >> 10>>11>>01>>11). Depending on whether a left or right turn is registered, a decimal value will increment or decrement in the software which will then symbolize the new value to display (software details will be in a later section).

Push Button for User Input

As mentioned earlier, the push button can send a 1 or 0 to the Arduino. Depending on how you configured your pull up resistor, the unpressed state of the button can be a 1 or 0. The push button also needs a decoupling capacitor like the RPG to smooth out any possible unwarranted input from the button. In this lab, the button is used to select the current number being visualized by the seven-segment display and, via the software, keep track of the 5-character code being produced (details in a later section).

Timer Used For Delays

***(see lines 373 of the source code to see timer configuration)**

We used TCNT0 (Timer 0), one of the two 8-bit timer registers that the ATmega328P has. TCNT0 is a register that holds the value of where to start counting from to its max value: 256. The smaller the value you put in TCNT0 the longer the delay will be. TCCRB0 is a mode control register that allows you to select a prescaler, set no clock source, and the mode in which the timer operates. In our implementation, setting TCCRB0 with 0b00000101 configures Normal Mode with bit 3 and a prescaler of 1024 with bits 0,1,2 (101).

So why do we need the prescaler and how does the timer give us the delay we want?

The prescaler essentially “slows down” the timer, which normally operates at 16MHz. The period of the clock (T_clock) is: T_clock = (1/(16MHz/1024)) = 6.4 x 10^-5.
Dividing our desired delay by the clock period gives us the amount of clocks needed to simulate our delay. Our implementation uses a delay of 10ms, so the calculation is as follows: (0.01s/6.4 x

10^-5) = 156.25, or approx. 156 clocks. Finally, we must find the value of where to start "counting" to make a 10ms delay. This can be calculated by: Max_Clocks - Clocks_Needed. In our case, 256 - 156 = 100. 100 in hexadecimal is 0x64, and this is the value that can be seen being loaded into TCNT0 in the source code.

How does the clock know when to stop counting?
Register TIFR0 holds the flag bits for all timers on the ATmega328P. We are concerned with bit TOV0, as once this bit is set, that means the timer has finished counting. We check for this change in TOV0 in our code, stop/reset the timer and exit the subroutine.

Software Explanation

Turning the encoder

The "main" subroutine loop contains code that calls the display subroutine and checks if encoder pin A, encoder pin B, or the pushbutton have gone low. If pin A has gone low, three subsequent checks are made in subroutines CW1, CW2, and CW3, which each check for the next state of the turn (pin A goes low, pin B goes low, pin A goes high, and pin B goes back high). If pin B goes low first, subroutines CCW1, CCW2, and CCW3 are similarly called that check for all states of the turn to finish before the turn is considered complete. If a CW turn has been completed, "increment" subroutine is called, and if a CCW turn is completed, "decrement" subroutine is called.

Incrementing and decrementing

Register R20 holds the base 10 value of the state the device is in, from 0 to 15 (corresponding to 0 to F on the display) and will be subsequently referred to as the "display state register". After a turn is complete but before changing the display state register, a check is made to see if the display is in the initial "-" state. If it is, a right turn will set the display state to 0 and a left turn will set the display state to 15. Additionally, if a CW turn is made but the display state is already at 15, no change is made. Similarly, no change is made after a CCW turn if the display state is already at 0. If all checks are passed, the display state register will be incremented or decremented by 1 after a CW turn or CCW turn, respectively.

Updating display

If display was in "-" state, subroutines "dash_to_0" or "dash_to_F" are called that update the display and the program goes back to the main loop. Otherwise, the program loads R19 with numbers from 0 to 15. Each time R19 is loaded with a new number, its value is compared with the display state register, and once a match is found the corresponding bit pattern is loaded into R16 and the program gets sent to main for the display to be updated.

Resetting the display

Once the pushbutton is detected as having gone low in the main loop, a register counts how long the button has been pressed by taking samples. Samples are taken by calling the 10ms delay, checking if the button has gone back high (indicating a button release), and then incrementing the sample count if it hasn't. With a 10ms delay, 200 samples must be counted (10ms x 200 = 2s) for a reset to occur and to go back to "-" state.

Entering the passcode

When a button press shorter than two seconds has occurred, check_digit subroutine is called which checks the value in R24 (the "button press" register), and depending on how many times the button has been pressed sends the program to one of five subroutines which all check whether the right digit has been entered, named check_digit_X (X representing which digit of the passcode is currently being checked). These subroutines then increments R23 (the "correct entries register") if the digit entered is the correct digit, and then sends the program back to the main loop regardless of whether the right digit has been pressed. When the button has been pressed 5 times, the program is sent to the "validate" subroutine, which compares the button press register with the correct entries register. If they match, the passcode is correct and the program call led_code subroutine and led L as well as DP are flashed for 5 seconds, and the program goes back to main loop in "-" display state with all respective registers reset. If the code is incorrect, the "incorrect_code" subroutine is called and "_" is displayed for 9 seconds and similarly goes back to the main loop.

4. Conclusion

This lab provided experience working with I/O registers and problem solving how to output lots of data using shift registers when it is not practical or not possible to directly output all of that data from the ports of the microcontroller. Experience was gained on using multiple devices for user input with the RPG and push-button. Utilizing the timer on the microcontroller gave insight to applications where it can be useful, especially with interrupts. The lab also provided experience with planning and implementing both software and hardware components in incremental steps. "Software design" practices specific to assembly were learned as well, as operations basic to higher level languages (if statements, loops, function calls) must be implemented creatively and efficiently on a case by case basis. Overall, the lab provided a helpful framework for problem solving at the intersection of software and hardware, and with a variety of components.

5. Appendix A: Source Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Assembly Language file for Lab 3 in ECE:3360
; Spring 2023, The University of Iowa
; Author : Max Finch, Tiger Slowinski
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.include "m328Pdef.inc"

.cseg
.org 0

setup:
    ; Configure I/O lines.
    sbi   DDRB,0      ; PB0 is now output (SER)
    sbi   DDRB,1      ; PB1 is now output (SRCLK)
    sbi   DDRB,2      ; PB2 is now output (RCLK)
    cbi   DDRB,3      ; PB3 is now input (button)
    cbi   DDRB,4      ; PB4 is now input (Rotary A)
    cbi   DDRD,5      ; PD5 is now input (Rotary B)
    sbi   DDRB,5      ; PB5 is led L
    cbi   PORTB,5     ; turn off led L

    ldi R16, 0x40     ; "-" is first character to be displayed
    ldi R20, 0        ; R20 holds state of display, 0 to 15 coorresponds to 0 to F
    jmp main



stay_put:
    rcall delay_10ms     ;R22 is sample register
    inc  R22
    cpi R22, 50          ;inner loop of 50
    breq inc_R27_reset_R22
    here:
    cpi R27, 4           ;outer loop of 4: 50 x 4 = 200 loops of 10ms = 2s
    breq set_led_to_dash ;reset
    sbic PINB, 3         ;if button released
    rjmp check_digit     ;Check the digit
    rjmp stay_put

inc_R27_reset_R22:
    inc R27
    ldi R22, 0
    rjmp here

set_led_to_dash:
    ldi R22, 0    ;reset R22 (200->0)
    ldi R27, 0    ;reset R27 (4->0)
    ldi R23, 0    ;reset register that counts correct passcode entry
    ldi R24, 0
    ldi R16, 0x40 ;set leds to dash
    rjmp main
```

```
validate:
    cpse R24, R23                   ;If equal passcode is correct.
    rjmp incorrect_code            ;If not correct go to to underscore and 9s delay
    rjmp led_code

incorrect_code:
    ldi R23, 0    ;reset register that counts correct passcode entry
    ldi R24, 0
    ldi R16, 0x08 ;set to underscore for 9 seconds
    rcall display
    rcall delay_9s
    rjmp set_led_to_dash

led_code:
    ;activate led for 5 seconds
    ldi R16, 0x80 ;set to decimal point
    rcall display
    sbi PORTB,5
    rcall delay_5s
    cbi PORTB,5
    rjmp set_led_to_dash

check_E_code:
        cpi R16, 0x79  ;if the right number
        breq tally  ;tally for a correct number
        inc R24
        rjmp main
check_8_code:
        cpi R16, 0x7F
        breq tally
        inc R24
        rjmp main
check_5_code:
        cpi R16, 0x6D
        breq tally
        inc R24
        rjmp main
check_9_code:
        cpi R16, 0x6F
        breq tally
        inc R24
        rjmp main
check_A_code:
        cpi R16, 0x77
        breq tally
        inc R24
        rjmp validate


main:
    rcall display
    sbis PINB, 4
    rjmp CW_1       ;if A is pulled low first, likely CW turn
    sbis PIND, 5
    rjmp CCW_1      ;if B is pulled low first, likely CCW turn
    sbis PINB, 3    ;button press
    rjmp stay_put
    rjmp main
```

```
increment:
    cpi R16, 0x40        ;Checks for "-" state and increments R20 if R20 < 15
    breq dash_to_zero
    cpi R20, 15
    breq main
    inc R20
    rjmp switch_number

dash_to_zero:
    ldi R16, 0x3F       ;For going from initial "-" to "0" on CW turn
    ldi R20, 0
    rjmp main

decrement:
    cpi R16, 0x40       ;Checks for "-" state and decrements R20 if R20 > 0
    breq dash_to_F
    cpi R20, 0
    breq main
    dec R20
    rjmp switch_number

dash_to_F:
    ldi R16, 0x71       ;For going from initial "-" to "F" on CCW turn, sets state to 15
    ldi R20, 15
    rjmp main

switch_number:
    ldi R19, 0      ;Checks if device is in 0 state, if not checks all other numbers
    cpse R20, R19
    rjmp check_1
    ldi R16, 0x3F
    rjmp main

check_1:
    ldi R19, 1      ;R20 is compared with decimal values 0-15, and when a match is found R16 is update
    cpse R20, R19
    rjmp check_2
    ldi R16, 0x06
    jmp main

check_2:
    ldi R19, 2
    cpse R20, R19
    rjmp check_3
    ldi R16, 0x5B
    jmp main

check_3:
    ldi R19, 3
    cpse R20, R19
    rjmp check_4
    ldi R16, 0x4F
    jmp main
```

```
check_4:
    ldi R19, 4
    cpse R20, R19
    rjmp check_5
    ldi R16, 0x66
    jmp main

check_5:
    ldi R19, 5
    cpse R20, R19
    rjmp check_6
    ldi R16, 0x6D
    jmp main

check_6:
    ldi R19, 6
    cpse R20, R19
    rjmp check_7
    ldi R16, 0x7D
    jmp main

check_7:
    ldi R19, 7
    cpse R20, R19
    rjmp check_8
    ldi R16, 0x07
    jmp main

check_8:
    ldi R19, 8
    cpse R20, R19
    rjmp check_9
    ldi R16, 0x7F
    jmp main

check_9:
    ldi R19, 9
    cpse R20, R19
    rjmp check_A
    ldi R16, 0x6F
    jmp main

check_A:
    ldi R19, 10
    cpse R20, R19
    rjmp check_B
    ldi R16, 0x77
    jmp main

check_B:
    ldi R19, 11
    cpse R20, R19
    rjmp check_C
    ldi R16, 0x7C
    jmp main
```

```
check_C:
    ldi R19, 12
    cpse R20, R19
    rjmp check_D
    ldi R16, 0x39
    jmp main

check_D:
    ldi R19, 13
    cpse R20, R19
    rjmp check_E
    ldi R16, 0x5E
    jmp main

check_E:
    ldi R19, 14
    cpse R20, R19
    rjmp check_F
    ldi R16, 0x79
    jmp main

check_F:
    ldi R16, 0x71
    jmp main

display: ; backup used registers on stack

    push R16
    push R17
    in R17, SREG
    push R17
    ldi R17, 8 ; loop --> test all 8 bits
loop:
    rol R16 ; rotate left trough Carry
    BRCS set_ser_in_1 ; branch if Carry is set
    ; put code here to set SER to 0
    cbi PORTB,0
    rjmp end
set_ser_in_1:
    ; put code here to set SER to 1...
    sbi PORTB,0
end:
    ; put code here to generate SRCLK pulse...
    rcall pulse_clock
    dec R17
    brne loop
    ; put code here to generate RCLK pulse
    rcall pulse_latch
    ; restore registers from stack
    pop R17
    out SREG, R17
    pop R17
    pop R16
    ret
```

```
delay_1s:
    rcall delay_10ms
    inc R22             ;loop 10ms 100 times = 1s delay
    cpi R22, 100
    brne delay_1s
    ldi R22, 0x00
    ret

delay_5s:
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    ret

delay_9s:
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    rcall delay_1s
    ret

pulse_clock:
    sbi   PORTB,1
    cbi   PORTB,1
ret

pulse_latch:
    sbi   PORTB,2
    cbi   PORTB,2
ret

delay_10ms:
    ldi R21, 0x64       ;100 (base 10) is loaded to counter register
    out TCNT0, R21
    ldi R21, 0b00000101 ;starts clock in normal mode, prescaler 1024
    out TCCR0B, R21
again:
    in R21, TIFR0
    sbrs R21, TOV0      ;skip if overflow flag is set
    rjmp again
    ldi R21, 0x00
    out TCCR0B, R21     ;stops timer
    ldi R21, (1<<TOV0)
    out TIFR0, R21      ;reset flag bit
    ret

.exit
```

6. Appendix B: References

Atmel Corporation. *AVR Instruction Set Manual - Microchip Technology*. <https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-0856-AVR-Instruction-Set-Manual.pdf>.

Arduino. *Arduino UNO Rev3 with Long Pins*.<https://docs.arduino.cc/retired/boards/arduino-uno-rev3-with-long-pins>

Last Minute Engineers. “In-Depth: How 74HC595 Shift Register Works & Interface with Arduino.” *Last Minute Engineers*, Last Minute Engineers, 12 Feb. 2023, <https://lastminuteengineers.com/74hc595-shift-register-arduino-tutorial/>.

ES23_Lab02.pdf, and ES23_Lab03.pdf provided by Professor Beichel