Author: Seth Bolen
Team Member: Adrian Alvarez
Professor Beichel
ECE:3360 Embedded Systems
Post-Lab Report 2

1. Introduction

The goal of this lab was to build a hexadecimal up/down counter using the ATmega328P microcontroller in combination with an 8-bit shift register, a 7-segment LED display, and a pushbutton switch.

The basic explanation of what our device does is as follows. When the power is turned on, the 7-segment display will show "0" and the counter will be by default in increment mode. After holding the physically debounced push button for under one second, the 7-segment display will either increment: (0⇨1⇨2⇨etc) or decrement: (0⇨F⇨E⇨etc) by one value. We programmed the button to switch between the 2 modes when it's pressed for more than one, but less than two seconds, and the counter resets back to 0 increment mode when the button is held longer than two seconds.

2. Schematics

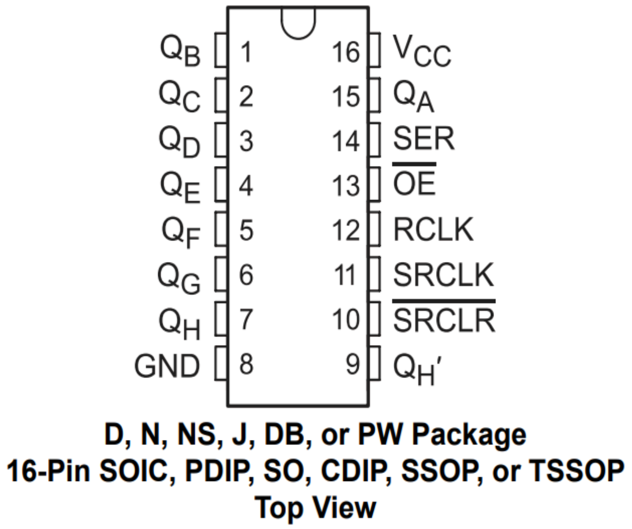

Figure 1: SN74HC595 8-bit shift register

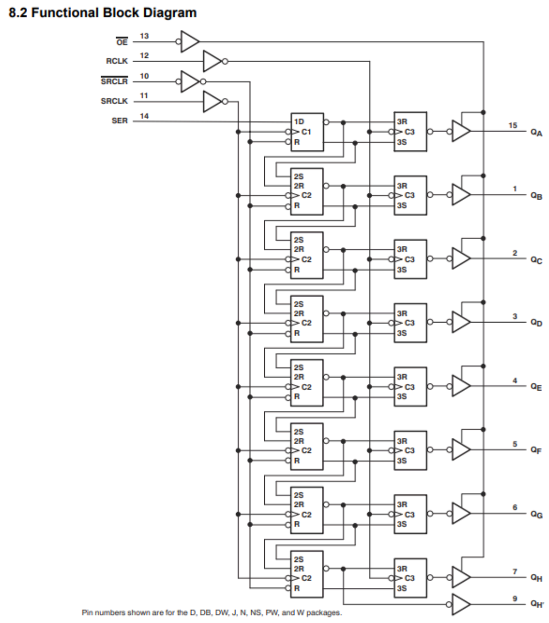
8.2 Functional Block Diagram
OE 13
RCLK 12
SRCLR 10
SRCLK 11
SER 14
15 QA
1 QB
2 QC
3 QD
4 QE
5 QF
6 QG
7 QH
9 QH'
Pin numbers shown are for the D, DB, DW, J, N, NS, PW, and W packages.

Figure 2: SN74HC595 Schematic

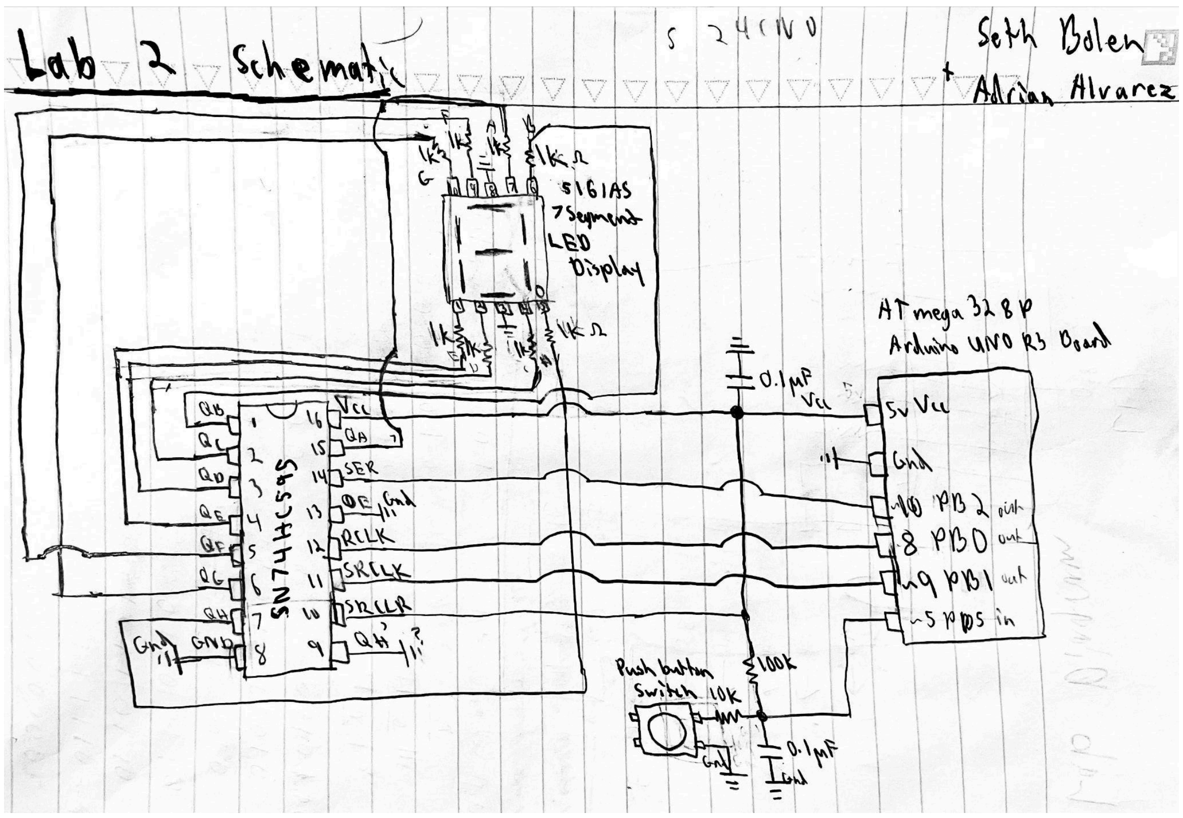

Figure 3: Implemented Circuit Schematic

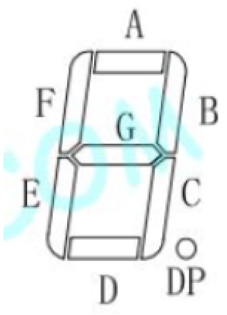

Figure 4: 5161AS 7-segment Display

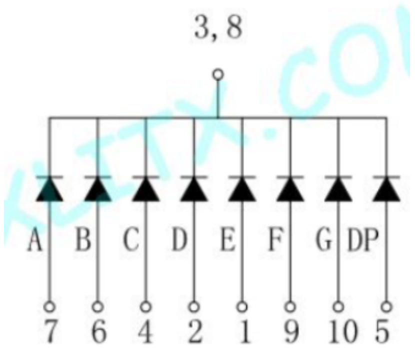

Figure 5: 5161AS Diagram

3. Discussion

Push Button Hardware and Debouncing

The pushbutton was implemented with a hardware-based debouncing circuit using a 10kΩ resistor, a 0.1μF capacitor to ground, and 100kΩ resistor to the Vcc. When buttons are pressed or released, the contacts may “bounce” in which the state transition jumps between high and low at a very high frequency before settling to a defined state. When the button is unpressed, the 5V terminal pushes current through the 100kΩ pull-up resistor, charging the capacitor and setting the output to high. When the button is pressed, the output becomes a voltage division between 100kΩ and 10kΩ, setting the output to 0.5V, which is low. However, because the capacitor was charged prior to being pressed, it discharges and smooths any immediate bouncing. Likewise when the button is released, it first recharges the capacitor, which smooths bouncing in the release and then sets the output back to high. This ensures reliable toggling between increment and decrement modes, as well as accurate counter updates and resets. The debouncing was verified using an oscilloscope.

Counter Functionality

The counter itself is designed to toggle between increment and decrements modes based on the duration of a button press. A short press lasting less than one second results in a counter update, while a longer press lasting between one and two seconds changes the mode from increment to decrement or vice versa. The mode is visually indicated using the decimal point LED on the 7-segment display, which remains off in increment mode and turns on in decrement mode. If the button is held for two or more seconds, the system resets the counter to zero and defaults to increment mode, ensuring a predictable state. Each action only takes effect after the button is released, aligning with best practices for user interface design and avoiding unintended multiple state changes due to holding the button too long.

Display

The 7-segment display is controlled using an SN74HC595 8-bit shift register, which allows the microcontroller to drive the display using fewer GPIO pins. The display updates dynamically based on the counter's current value, which is stored in an internal register and converted into the appropriate 7-segment pattern. Since each segment requires a controlled amount of current, we used 1kΩ resistors to limit the segment current to approximately 5mA, which is within the display's recommended operating limits. This not only protects the hardware components but also ensures consistent brightness across the display.

Code Implementation

Programming the microcontroller in assembly required structuring the code around key functionalities. The main loop continuously monitors input signals from the pushbutton to determine the next action. A dedicated mode selection subroutine detects the duration of a button press and switches between increment and decrements modes accordingly. Another subroutine handles the actual counter update, ensuring that values wrap around correctly at hexadecimal boundaries. For example, if the counter is at 'F' and an increment command is issued, it wraps back to '0', while decrementing from '0' results in 'F'. To ensure smooth operation, debounce handling was integrated within the button press detection subroutine, preventing false triggers.

4. Conclusion

From this lab, we gained more experience with programming in Assembly, handling timing, and understanding hardware implementation, such as wiring the 7-segment display to the microcontroller using a shift register. We developed a deeper understanding of input debouncing through the use of a pushbutton switch and the importance of generating smooth input signals to produce the desired output. Furthermore, we demonstrated an understanding of the 7-segment display and how the LED behaves when certain bits are registered while pressing the button. Additionally, we gained experience resolving timing issues by implementing a delay subroutine after receiving the pushbutton signal. Overall, this lab introduced and reinforced fundamental concepts essential to working with embedded systems.

5. Appendix A: Source Code

```
;
; Lab2.asm
;
; Created: 2/10/2025 5:44:39 PM
; Author : adralvarez + Seth Bolen
; Configure PB1, PB2, and PB3 as output pins.
sbi   DDRB,0         ; PB0 is now output
sbi   DDRB,1       ; PB1 is now output
sbi   DDRB,2       ; PB2 is now output
cbi   DDRD,5       ; PB3 is now input

.equ SHORT_PRESS_TIME = 5   ; about 1 second threshold
.equ LONG_PRESS_TIME  = 12  ; about 2 second threshold
;initialize the equivalent hex to the corresponding number with and without a decimal point
.equ value1= 0x06 .equ value1dec = 0x86  ; 10000110 and 00000110
.equ value2= 0x5B .equ value2dec = 0xDB  ;value for 2 and 11011011 the dec version
.equ value3= 0x4f .equ value3dec = 0xCF  ; 11001111 value for 3
.equ value4= 0x66 .equ value4dec = 0xE6  ; 11100110;value for 4
.equ value5= 0x6d .equ value5dec = 0xED  ; 11101101;value for 5
.equ value6= 0x7d .equ value6dec = 0xFD  ; 11111101;value for 6
.equ value7= 0x07 .equ value7dec = 0x87  ; 10000111;value for 7
.equ value8= 0x7f .equ value8dec = 0xFF  ; 11111111;value for 8
.equ value9= 0x6F .equ value9dec = 0xEF  ; 11101111;value for 9
.equ valueA= 0x77 .equ valueAdec = 0xF7  ; 11110111;value for A
.equ valueB= 0x7C .equ valueBdec = 0xFC  ; 11111100;value for B
.equ valueC= 0x39 .equ valueCdec = 0xB9  ; 10111001;value for C
.equ valueD= 0x5E .equ valueDdec = 0xDE  ; 11011110;value for D
.equ valueE= 0x79 .equ valueEdec = 0xF9  ; 11111001 ;value for E
.equ valueF= 0x71 .equ valueFdec = 0xF1  ; 11110001;value for F
.equ value0= 0x3f .equ value0dec = 0xBF  ; 10111111 ;value for 0
; start main program
; display a digit as the starting digit, 0
ldi R16, value0 ; load pattern to display, sets the display to full
rcall display ; call display subroutine to display it
;MAIN LOOP, RECURSES UNTIL BUTTON PRESS
loop1:
    ldi R20, 0         ; Reset the press-length counter for good measure
    ; Wait here until the button is pressed.
    ; Since the button is active low, when NOT pressed PD5 = 1.
    sbic PIND,5        ; Skip next instruction if PD5 is clear (bit is 0) (i.e. button pressed)
    rjmp loop1         ; If button is not pressed (bit is 1), loop here.
    rjmp button_pressed; When button is pressed, PD5 becomes 0, so SBIC skips the rjmp and we rjmp to
button_pressed subroutine instead.

button_pressed:    ; In this loop, if the button is still pressed (active low: PD5 = 0), then SBIC will
skip the next
                                    ; instruction. If the button is released (PD5 = 1), SBIC will NOT
skip, and will go rjmp to do_something_loop
    sbic PIND,5            ; Skip next instruction if PD5 is clear (button unpressed)
    rjmp do_something_loop ; If not skipped (button released), exit loop.
    inc R20                ; Increment the press-duration counter to use to compare with later
    call delay             ; Delay a fixed interval (about 0.2 of a second)
    rjmp button_pressed    ; loop back and continue while button is pressed

do_something_loop:  ;do_something_loop subroutine decides what the button does based off of how long it
was pressed (the number stored in r20)
        cpi R20, SHORT_PRESS_TIME ;compares R20 and value stored as SHORT_PRESS_TIME
    brlo increment_number   ; If R16 < SHORT_PRESS_TIME (1s), go to increment_number subroutine, else
check if less than long press

    cpi R20, LONG_PRESS_TIME ;compares R20 and value stored as LONG_PRESS_TIME
    brlo switch_mode_code    ; If R16 < LONG_PRESS_TIME (2s) , go to switch_mode_code subroutine

    rjmp long_press_RESET  ; since neither were true, the  R16 must be >= LONG_PRESS_TIME, so we go to
long_press_RESET subroutine

switch_mode_code: ;switch_mode_code subroutine will always switch the 7th bit of whatever R16 is from 1
to 0 or 0 to 1
        sbrs R16, 7  ; Skip next instruction if bit 7 is SET (1)
        rjmp bit_is_clear  ; If skipped, bit 7 was 1; otherwise, jump means it was 0
        andi R16, 0x7F  ; should remove the decimal from the number
        continue: ;continue is from bit_is_clear runoff subroutine. A little messy but it works
        rcall display ;update display
        ldi R20, 0    ; Reset the press-length counter for good measure
        rjmp loop1    ;jump back to base loop1 to be ready to button again
bit_is_clear:
        ori R16, 0x80 ;should keep the same number but give it a decimal
        rjmp continue

increment_number: ;The subroutine increment number actually has functionality to both increment AND
decrement
        ;FIRST IS INCREMENT MODE WITH NON-DECIMAL POINT NUMBERS
        cpi R16, value0   ; Compare R16  with MY_VALUE
        brne skip_next      ; If R16 1= MY_VALUE, skip next line
        ldi R16, value1   rcall display rjmp loop1   ; This line is skipped if R16 == MY_VALUE
        skip_next:

        cpi R16, value1   ; Compare R16  with MY_VALUE
        brne skip_next1      ; If R16 != MY_VALUE, skip next line
        ldi R16, value2   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next1:

        cpi R16, value2   ; Compare R16  with MY_VALUE
        brne skip_next2      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, value3   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next2:

        cpi R16, value3   ; Compare R16  with MY_VALUE
        brne skip_next3      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, value4   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next3:

        cpi R16, value4   ; Compare R16  with MY_VALUE
        brne skip_next4      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, value5   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next4:

        cpi R16, value5   ; Compare R16  with MY_VALUE
        brne skip_next5      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, value6   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next5:

        cpi R16, value6   ; Compare R16  with MY_VALUE
        brne skip_next6      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, value7   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next6:

        cpi R16, value7   ; Compare R16  with MY_VALUE
        brne skip_next7      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, value8   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next7:

        cpi R16, value8   ; Compare R16  with MY_VALUE
        brne skip_next8      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, value9   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next8:

        cpi R16, value9   ; Compare R16  with MY_VALUE
        brne skip_next9      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, valueA   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next9:

        cpi R16, valueA   ; Compare R16  with MY_VALUE
        brne skip_nextA      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, valueB   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextA:

        cpi R16, valueB   ; Compare R16  with MY_VALUE
        brne skip_nextB      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, valueC   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextB:

        cpi R16, valueC   ; Compare R16  with MY_VALUE
        brne skip_nextC      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, valueD   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextC:

        cpi R16, valueD   ; Compare R16  with MY_VALUE
        brne skip_nextD      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, valueE   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextD:

        cpi R16, valueE   ; Compare R16  with MY_VALUE
        brne skip_nextE      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, valueF   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextE:
```

```
        cpi R16, valueF   ; Compare R16  with MY_VALUE
        brne skip_nextF      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next line
        ldi R16, value0   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextF:

        ;THIS IS THE END OF CHECKING TO INCREMENT, NOW I WILL START TO THE CHECKS FOR DECREMENT, THIS
MIGHT BE OVER COMPLICATED BUT ITS MINE AND IT WORKS

        cpi R16, value3dec   ; Compare R16  with MY_VALUE
        brne skip_next3dec      ; If R16 1= MY_VALUE, skip next line
        ldi R16, value2dec   rcall display rjmp loop1   ; This line is skipped if R16 == MY_VALUE
        skip_next3dec:

        cpi R16, value2dec   ; Compare R16  with MY_VALUE
        brne skip_next2dec      ; If R16 != MY_VALUE, skip next line
        ldi R16, value1dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next2dec:

        cpi R16, value1dec   ; Compare R16  with MY_VALUE
        brne skip_next1dec      ; If R16 != doesnt equal MY_VALUE, skip incrementing to 3 on the next
line
        ldi R16, value0dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next1dec:

        cpi R16, value0dec   ; Compare R16  with MY_VALUE
        brne skip_next0dec      ; If R16 != MY_VALUE, skip next line
        ldi R16, valueFdec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next0dec:

        cpi R16, valueFdec   ; Compare R16  with MY_VALUE
        brne skip_nextFdec      ; If R16 != MY_VALUE, skip next line
        ldi R16, valueEdec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextFdec:

        cpi R16, valueEdec   ; Compare R16  with MY_VALUE
        brne skip_nextEdec      ; If R16 != MY_VALUE, skip next line
        ldi R16, valueDdec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextEdec:

        cpi R16, valueDdec   ; Compare R16  with MY_VALUE
        brne skip_nextDdec      ; If R16 != MY_VALUE, skip next line
        ldi R16, valueCdec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextDdec:

        cpi R16, valueCdec   ; Compare R16  with MY_VALUE
        brne skip_nextCdec      ; If R16 != MY_VALUE, skip next line
        ldi R16, valueBdec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextCdec:

        cpi R16, valueBdec   ; Compare R16  with MY_VALUE
        brne skip_nextBdec      ; If R16 != MY_VALUE, skip next line
        ldi R16, valueAdec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextBdec:

        cpi R16, valueAdec   ; Compare R16  with MY_VALUE
        brne skip_nextAdec      ; If R16 != MY_VALUE, skip next line
        ldi R16, value9dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_nextAdec:
```

```
        cpi R16, value9dec   ; Compare R16  with MY_VALUE
        brne skip_next9dec      ; If R16 != MY_VALUE, skip next line
        ldi R16, value8dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next9dec:

        cpi R16, value8dec   ; Compare R16  with MY_VALUE
        brne skip_next8dec      ; If R16 != MY_VALUE, skip next line
        ldi R16, value7dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next8dec:

        cpi R16, value7dec   ; Compare R16  with MY_VALUE
        brne skip_next7dec      ; If R16 != MY_VALUE, skip next line
        ldi R16, value6dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next7dec:

        cpi R16, value6dec   ; Compare R16  with MY_VALUE
        brne skip_next6dec      ; If R16 != MY_VALUE, skip next line
        ldi R16, value5dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next6dec:

        cpi R16, value5dec   ; Compare R16  with MY_VALUE
        brne skip_next5dec      ; If R16 != MY_VALUE, skip next line
        ldi R16, value4dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next5dec:

        cpi R16, value4dec   ; Compare R16  with MY_VALUE
        brne skip_next4dec      ; If R16 != MY_VALUE, skip next line
        ldi R16, value3dec   rcall display    rjmp loop1    ; This line is skipped if R16 == MY_VALUE
        skip_next4dec:
                                                                                  ;THE END OF COMPARING
YES THIS IS LONG BUT IT WORKS GREAT
        rcall display ;update display
        ldi R20, 0 ;reset button press length time for good measure
        rjmp loop1 ;jump back to base loop1 to be ready to button again

long_press_RESET: ;long_press_RESET subroutine resets the display to 0
        ldi R16, 0x3f ; load pattern to display, sets the display to 0
        rcall display ; call display subroutine
        ldi R20, 0  ;reset button press length time for good measure
        rjmp loop1 ;jump back to base loop1 to be ready to button again

display: ;display pushes the R16 value to shift register
 ; 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,2
 rjmp end
set_ser_in_1:
 ; put code here to set SER to 1
 sbi PORTB,2
end:
 ; put code here to generate SRCLK pulse
 sbi PORTB,1
 cbi PORTB,1
 dec R17
 brne loop
 ; put code here to generate RCLK pulse
 sbi PORTB,0
 cbi PORTB,0
 ; restore registers from stack
 pop R17
 out SREG, R17
 pop R17
 pop R16
 ret

; === Delay Subroutines === basically makes the program run through junk loops for about 0.2 seconds
delay:
    ; Outer loop counter in R22
    ldi R22, 50
outer_loop:
    ; Middle loop counter in R23
    ldi R23, 50
middle_loop:
    ; Inner loop counter in R24
    ldi R24, 200
inner_loop:
    dec R24
    brne inner_loop
    dec R23
    brne middle_loop
    dec R22
    brne outer_loop
    Ret
```

6. Appendix B: References

"SN54HC595." *SN54HC595 Data Sheet, Product Information and Support | TI.Com*, Texas Instruments, 2021, <www.ti.com/product/SN54HC595> .

XLITX. "5161AS." 5161AS Datasheet, Apr. 2019, http://www.xlitx.com/datasheet/5161AS.pdf