Author: Adrian Alvarez
Team Member: Seth Bolen
Professor Beichel
ECE:3360 Embedded Systems
Post-Lab Report 3

1. Introduction

The goal of this lab was to construct a simple electronic door lock system capable of accepting a 5-digit hexadecimal code. The device utilizes a rotary pulse generator (RPG), an 8-bit shift register, a pushbutton switch, and a 7-segment LED display. The code is entered by utilizing the RPG 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 "-" to indicate that the system is ready for code entry. The RPG is used to select a value for each hexadecimal digit with clockwise rotation incrementing the displayed value ("-" → "0" → "1" → ... → "F") and counter-clockwise rotation decrementing the displayed value ("F" → "E" → ... → "0" → "-"). To confirm a selection and move to the next digit the user presses the push button for less than 1 second. Once all five digits have been entered the system compares the entered code to the pre-programmed unlock code (54393). If the entered code is correct the yellow LED on the Arduino board is activated for 4 seconds, during which the display shows “.”. Afterwards the system resets to display “-” to accept a new code entry. If the code is incorrect the display shows “_” and resets and accepts a new entry after seven seconds. The user can reset the entered code at any point by pressing the push button for more than 2 seconds, which will return the display to “-” to begin a new code entry.

2. Schematics

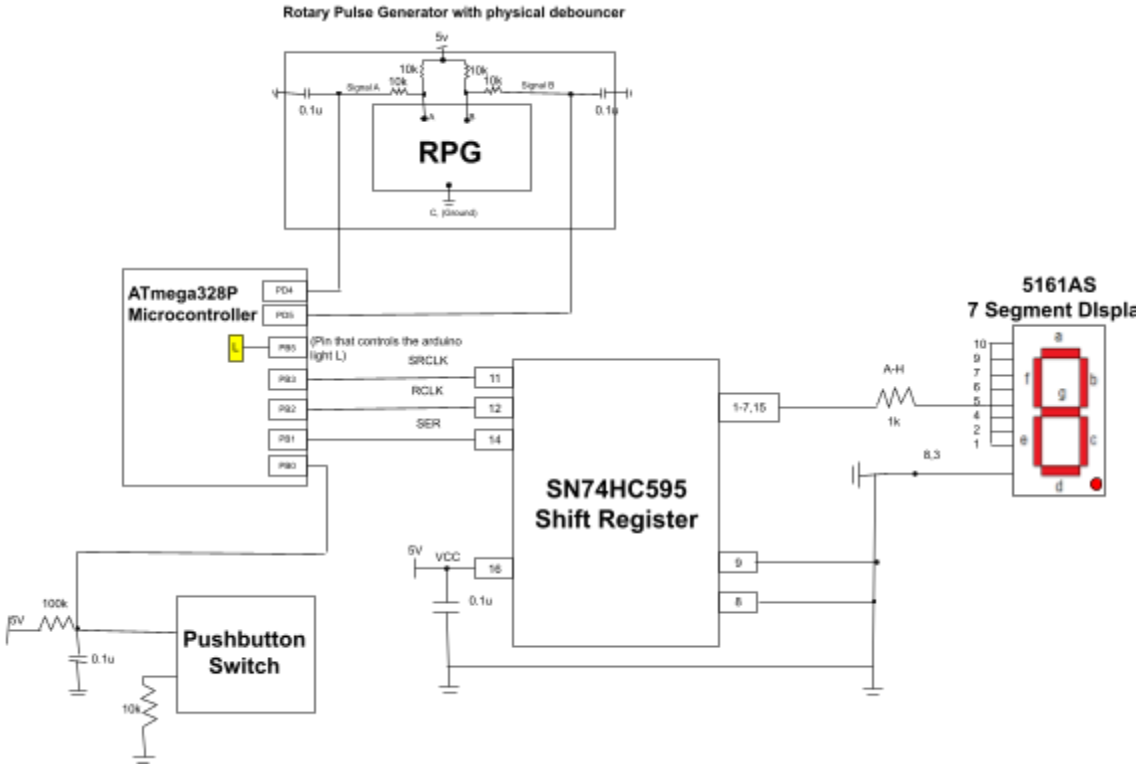

Figure 1: Implemented Circuit Schematic

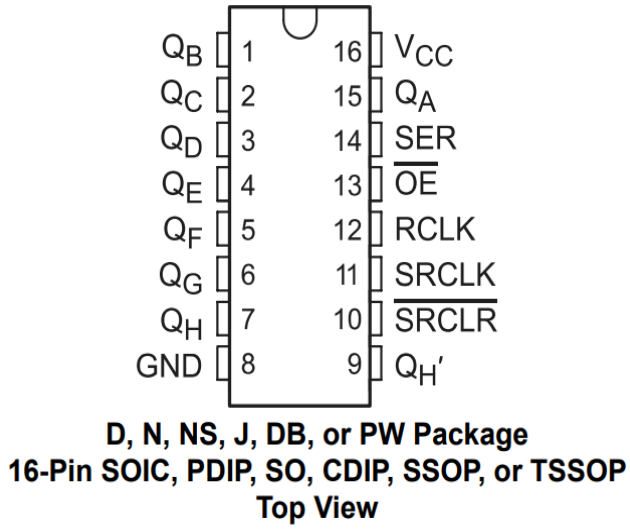

Figure 2: SN74HC595 8-bit Shift Register

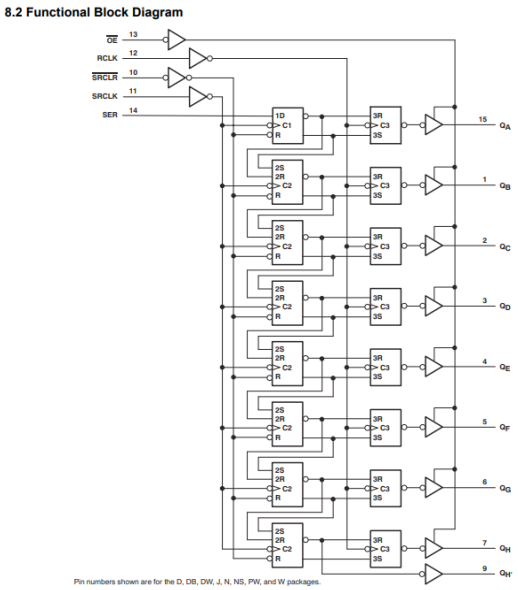

Figure 3: SN74HC595 Schematic

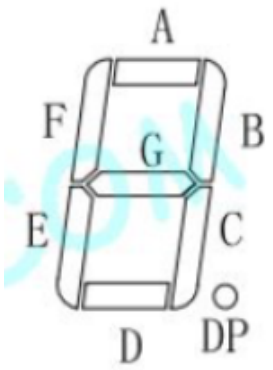

Figure 4: 5161AS 7-segment Display

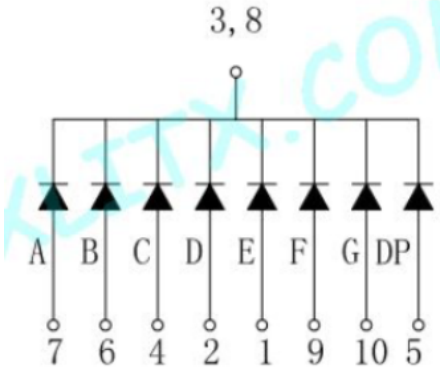

Figure 5: 5161AS Diagram

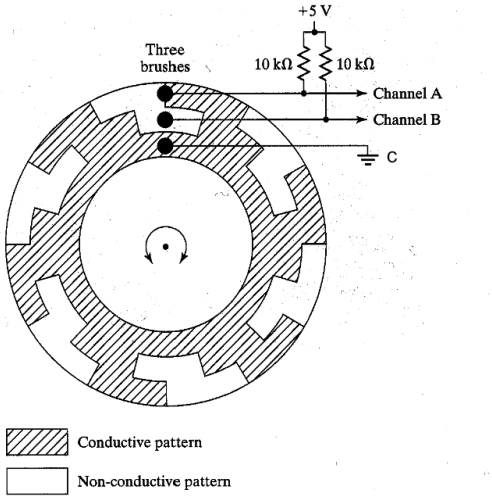

Figure 6: RPG Schematic

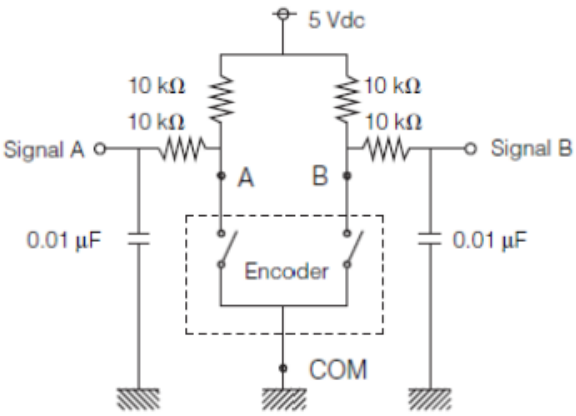

Figure 7: RPG Debounce Schematic

3. Discussion

Rotary Pulse Generator (RPG) Hardware and Decoding

The RPG was implemented with a hardware-based debouncing circuit using two 10kΩ to the Vcc, another two 10kΩ, and two 0.01µF to ground. We used two input pins to read the signals from the RPG, with pull-up resistors to ensure a stable signal. The RPG produces two signals, A and B, that are 90 degrees out of phase allowing us to determine both the direction and amount of rotation.

The RPG algorithm monitors both signals and detects changes in their states. When the RPG is rotated clockwise signal A leads signal B, while counter-clockwise rotation causes signal B to lead signal A. By monitoring the sequence of these signals, we can accurately determine the direction of rotation. We implemented a state machine approach to track the current and previous states of both signals which allowed us to determine the direction of rotation and filter out any bounce in the signals.

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, similar to our previous lab. When buttons are pressed or released, the contacts may "bounce," causing multiple state transitions before settling to a defined state. Our debouncing circuit ensures reliable operation for both short presses (less than 1 second) used for digit confirmation and long presses (more than 2 seconds) used for code reset.

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. The capacitor smooths any immediate bouncing during press and release cycles. This ensures reliable operation for digit confirmation and code reset functions.

Digit Entry and Code Verification

The digital entry system is designed to handle five hexadecimal digits sequentially. The user selects each digit by rotating the RPG clockwise or counter-clockwise and confirms the selection with a short button press. We implemented boundary checking to ensure that the display does not change beyond "F" for clockwise rotation or "0" for counter-clockwise rotation. After all five digits have been entered, the system compares the entered code to the pre-programmed unlock code (54393) stored in program memory. The comparison is performed based on increments, and the system determines if the entered code matches the stored code.

Timer Implementation

We utilized the 8-bit Timer/Counter0 hardware to generate all required time sequences, including the 4-second unlock period, the 7-second lockout period after an incorrect code entry, and timing for button press duration detection. The timer was configured in normal mode with a prescaler of 1024, resulting in a clock frequency of approximately 15.625 kHz (16MHz / 1024). This allowed us to generate accurate timing intervals for all the required functions. The timer was used to measure button press duration by starting the timer when the button is pressed and checking its value when the button is released. This enables the system to differentiate between short presses (less than 1 second) for digit confirmation and long presses (more than 2 seconds) for code reset.

Display Control

The 7-segment display is controlled using an SN74HC595 8-bit shift register. We implemented a lookup table in program memory to convert hexadecimal values (0-9, A-F) to the appropriate 7-segment pattern for display. Each segment requires a

controlled amount of current, so we used 1kΩ resistors to limit the segment current to approximately 5mA, which is within the display's recommended operating limits and below the required 6mA maximum. This protects the hardware components and ensures consistent brightness across the display.

4. Conclusion

From this lab we gained experience with digital I/O, timers, complex timing issues, and rotary pulse generators. We developed a deeper understanding of debouncing techniques for both push buttons and rotary encoders, as well as implementing complex state machines for user interface management. We successfully implemented a five-digit electronic door lock system using assembly language with all the required functionality including digit selection, code entry, code verification, and appropriate feedback for correct and incorrect codes. The system demonstrates robust handling of user input through the RPG and push button, with appropriate debouncing and timing controls. We also gained valuable experience with Timer/Counter0 hardware for generating precise timing sequences without using interrupts. This approach required careful management of the timer value and system state to ensure accurate timing for all required functions.

Overall, this lab enhanced our understanding of embedded system design, particularly in the areas of user interface implementation, state machine design, and hardware timing control. These skills are essential for developing more complex embedded systems in the future.

5. Appendix A: Source Code

```
;
; Lab3.asm
;
; Created: 3/3/2025 9:40:14 PM
; Author : Seth Bolen & Adrian Alvarez
;

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

; ----------
; -- CODE --
; ----------
;   54393

; Use pin 4 and 5 for A and B outputs from RPG
.equ A_BIT = 4
.equ B_BIT = 5

; lookup table
lookup_table:
        .db 0x3F, 0x06, 0x5B, 0x4F ; 0-3 on the display
        .db 0x66, 0x6D, 0x7D, 0x07 ; 4-7 on the display
```

```
        .db 0x7F, 0x6F, 0x77, 0x7C ; 8-B on the display
        .db 0x39, 0x5E, 0x79, 0x71 ; C-F on the display


; configure I/O lines as output & connected to SN74HC595
sbi DDRB, 1 ; PB1 is now output to SER, PIN 14 on SR
sbi DDRB, 2 ; PB2 is now output to SRCLK,
sbi DDRB, 3 ; PB3 is now output to RCLCK
cbi DDRB, 0 ; PB0 IS INPUT FROM BUTTON!
sbi DDRB,5 ;output yellow LED
ldi R20, 0

; set pins PB4 and PB5 as input lines for outputs A and B from rotary device
cbi DDRD, A_BIT
cbi DDRD, B_BIT

; timing
.equ SHORT_PRESS_TIME = 5 ; about 1 second threshold
.equ LONG_PRESS_TIME  = 12 ; about 2 second threshold

; states
.equ STATE_00 = 0 ; both A and B are low
.equ STATE_01 = 1 ; A is high, B is low
.equ STATE_11 = 2 ; both A and B are high
.equ STATE_10 = 3 ; A is low, B is high

; set names for registers
.def display_value = R16 ; value for displaying
.def temp = R21 ; temp
.def timerINC = R20 ; for timer
.def curr_state = R24 ; current state of the state machine
.def new_state = R25 ; new state to transition to

.def count = R18 ; keeps track of what number is being displayed
.def inputIndex = R22 ; keeps track of how many inputs have already been input so we can easier compare
.def willUnlock = R23 ; will unlock will act as a boolean, we will set it to true if all input numbers
match the combination.

ldi display_value, 0x40 ; set the first thing when power to "-"
rcall display
cbi PORTB, 5 ; led reset
ldi R18, 0 ; load default value 0 into R18
ldi willUnlock, 0 ; load 0 so we can compare if its 5 later on
ldi inputIndex, 0 ; load 0 do determine which combination index we are on

;Using R16, R17, R18, R20, R21, R22, R23, R24, R27, R30, R31
main_loop:
    ; read current value of A and B pins
    in temp, PIND
    andi temp, (1 << A_BIT) | (1 << B_BIT) ; mask out all bits except A and B

    ; determine the new state based on pin readings
    ldi new_state, STATE_00 ; default to 00
    cpi temp, 0x00 ;check if both pins are low
    breq state_detected
    ldi new_state, STATE_01 ; set to 01
    cpi temp, 0x10 ; check if A is high, B is low
    breq state_detected
    ldi new_state, STATE_11 ; set to 11
```

```
    cpi temp, 0x30 ; check if both pins are high
    breq state_detected
    ldi new_state, STATE_10 ; set to 10
    cpi temp, 0x20 ; check if A is low, B is high
    breq state_detected

    ; if we get here, the reading was invalid (shouldn't happen)
    rjmp check_button

state_detected:
    ; if the state hasn't changed, nothing to do
    cp new_state, curr_state
    breq check_button ; no change in state, check button instead

    ; check for clockwise rotation (00->01->11->10->00)
    cpi curr_state, STATE_00
    brne not_cw_from_00 ; if not in state 00, check next state
    cpi new_state, STATE_01
    breq update_state ; valid transition, but rotation not complete
    rjmp check_ccw ; not a valid CW transition, check CCW

not_cw_from_00:
    cpi curr_state, STATE_01
    brne not_cw_from_01 ; if not in state 01, check next state
    cpi new_state, STATE_11
    breq update_state ; valid transition, but rotation not complete
    rjmp check_ccw ; not a valid CW transition, check CCW

not_cw_from_01:
    cpi curr_state, STATE_11
    brne not_cw_from_11 ; if not in state 11, check next state
    cpi new_state, STATE_10
    breq update_state ; valid transition, but rotation not complete
    rjmp check_ccw ; not a valid CW transition, check CCW

not_cw_from_11:
    cpi curr_state, STATE_10
    brne check_ccw ; if not in state 10, check CCW
    cpi new_state, STATE_00
    brne check_ccw ; if not transitioning to 00, check CCW

    ; clockwise rotation completed
    cpi count, 15 ; check if max value (F)
    brge stay_at_F ; if max, dont increment
    inc count
    rcall update_display ; update display with new count
    rjmp update_state ; update state and continue

check_ccw:
    ; check for counter-clockwise rotation (00->10->11->01->00)
    cpi curr_state, STATE_00
    brne not_ccw_from_00 ; if not in state 00, check next state
    cpi new_state, STATE_10
    breq update_state ; valid transition, but rotation not complete
    rjmp update_state ; not a valid transition

not_ccw_from_00:
    cpi curr_state, STATE_10
    brne not_ccw_from_10; if not in state 10, check next state
```

```
    cpi new_state, STATE_11
    breq update_state ; valid transition, but rotation not complete
    rjmp update_state ; not a valid transition

not_ccw_from_10:
    cpi curr_state, STATE_11
    brne not_ccw_from_11 ; if not in state 11, just update state
    cpi new_state, STATE_01
    breq update_state ; valid transition, but rotation not complete
    rjmp update_state ; not a valid transition

not_ccw_from_11:
    cpi curr_state, STATE_01
    brne update_state ; if not in state 01, just update state
    cpi new_state, STATE_00
    brne update_state ; if not transitioning to 00, just update state

    ; counter-clockwise rotation completed
    cpi count, 0 ; check if at minimum value (0)
    breq stay_at_0 ; if min, dont increment
    dec count
    rcall update_display ; update display with new count

update_state:
    ; update the current state
    mov curr_state, new_state

check_button:
    ; Button code
    ; Since the button is active low, when NOT pressed PB0 = 1.
    sbis PINB, 0      ; skip next instruction if PB0 is set (bit is 1) (i.e. button not pressed)
    rjmp button_pressed ; when button is pressed, PB0 becomes 0, so we rjmp to button_pressed subroutine

    cpi inputIndex, 5 ; checks if the number combination indexs is 5
    breq combinationCheck ; if true go to combinationCheck

    rjmp main_loop    ; continue looping if no change

; button is pressed
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
        sbi PORTB, 5 ; light up yellow led on board and decimal for 4 seconds
        rcall delay_10ms ; ea 1 sec
        cbi PORTB, 5 ; turn off yellow led

    sbic PINB,0 ; skip next instruction if PD5 is clear (button unpressed)
    rjmp do_something_loop ; if not skipped (button released), exit loop.
    inc timerINC ; increment the press-duration counter to use to compare with later
    rcall delay_10ms ; delay a fixed interval (about .1 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 timerINC, 100
        brlo checkInput ; iF timer increment is less than .1 ms * 100 = 10s, jump to storing or
checking if the input matches the combination
        rjmp reset_display
```

```
reset_display:
        ldi timerINC, 0
    ldi display_value, 0x40 ; Load "-" character
    rcall display           ; Call display update
    ldi count, 0            ; Reset count
    ldi inputIndex, 0       ; Reset input index
    ldi willUnlock, 0       ; Reset unlock flag
    rjmp main_loop          ; Return to main loop

; makes sure display doesnt overflow over F
stay_at_F:
    ; Stay at "F"
    ldi display_value, 0x71 ; displays "F"
    rcall display
    rcall delay
    rjmp main_loop ; keep looping

; makes sure display doesnt overflow over 0
stay_at_0:
    ; Stay at "0"
    ldi display_value, 0x3F ; displays "0"
    rcall display
    rcall delay
    rjmp main_loop ; keep looping

; checks if the combination is correct
combinationCheck:
        cpi willUnlock, 5
        breq unlockLock ; if combination correct, willUnlock == 5, unlock lock
        rjmp lock ; else keep it locked

; checks the input for the combinaion, based on which index the user is on
checkInput:
        cpi inputIndex, 0 ; checking what digit (place in the combination sequence of code) we will be
comparing the current input/digit
        breq checkInput0
        cpi inputIndex, 1
        breq checkInput1
        cpi inputIndex, 2
        breq checkInput2
        cpi inputIndex, 3
        breq checkInput3
        cpi inputIndex, 4
        breq checkInput4

        rjmp main_loop

; checkInput0 through checkInput4 checks if the input value matches the corresponding passcode value and
increments the code index
; if one of the values doesn't match, the willUnlock value will not increment
checkInput0:
        inc inputIndex ; increments the inputIndex to go to the next input
        cpi count, 5
        breq inc_index ; if first input is equal to 5 go to inc_index
        rjmp main_loop ; go back to main loop
checkInput1:
        inc inputIndex ; increments the inputIndex to go to the next input
        cpi count, 4
```

```
        breq inc_index ; if first input is equal to 4 go to inc_index
        rjmp main_loop
checkInput2:
        inc inputIndex ; increments the inputIndex to go to the next input
        cpi count, 3
        breq inc_index ; if first input is equal to 3 go to inc_index
        rjmp main_loop
checkInput3:
        inc inputIndex ; increments the inputIndex to go to the next input
        cpi count, 9
        breq inc_index ; if first input is equal to 9 go to inc_index
        rjmp main_loop
checkInput4:
        inc inputIndex ; increments the inputIndex to go to the next input
        cpi count, 3
        breq inc_index ; if first input is equal to 3 go to inc_index
        rjmp main_loop

; increments willUnlock
inc_index:
        inc willUnlock
        rjmp main_loop

; unlocks lock
unlockLock:
        ; the three snippets of code resets all the registers so the user may try another combination
        ldi count, 0 ; resets the count
        ldi inputIndex, 0 ; resets the inputIndex
        ldi willUnlock, 0 ; resets the willUnlock
        ldi display_value, 0x80 ; displays "."
        rcall display
        sbi PORTB, 5 ; light up yellow led on board and decimal for 4 seconds
        rcall delay ; ea 1 sec
        rcall delay
        rcall delay
        rcall delay
        cbi PORTB, 5 ; turns off yellow led
        ldi display_value, 0x40 ; reset display to "-"
        rcall display
        rjmp main_loop ; goes back to main_loop for user to try again

; keeps lock locked
lock:
        ; the three snippets of code resets all the registers so the user may try another combination
        ldi count, 0 ; resets the count
        ldi inputIndex, 0 ; resets the inputIndex
        ldi willUnlock, 0 ; resets the willUnlock
        ldi display_value, 0x08 ; displays "_"
        rcall display
        rcall delay
        rcall delay
        rcall delay
        rcall delay
        rcall delay
        rcall delay
        rcall delay
        rcall delay
        rcall delay
        ldi display_value, 0x40 ; reset display to "-"
```

```
        rcall display
        rjmp main_loop ; goes back to main_loop for user to try again

; uses the lookup table to update whats displayed
update_display:
        ; Use the Z pointer to look up the display pattern from program memory
    ldi ZL, LOW(2*lookup_table) ; load low byte of table address with the 2* multiplier for word
addressing
    ldi ZH, HIGH(2*lookup_table) ; load high byte of table address

    mov temp, count ; move count to temp register
    add ZL, temp ; add offset to Z pointer
    ldi temp, 0 ; clear temp for carry
    adc ZH, temp ; add carry to high byte

    lpm display_value, Z ; load pattern from program memory at Z pointer

    rcall display ; call display routine
        rcall delay_10ms
        rcall delay_10ms rcall delay_10ms
        rcall delay_10ms
    rjmp main_loop ; return to main loop


display:
        ; backup used registers on stack
        push display_value
        push R17
        in R17, SREG
        push R17

        ldi R17, 8 ; loop --> test all 8 bits
loop:
        rol display_value ; rotate left trough Carry
        BRCS set_ser_in_1 ; branch if Carry is set
        cbi PORTB,1 ; set SER to 0

        rjmp end
set_ser_in_1:
        sbi PORTB,1 ; set SER to 1

end:
; generate SRCLK pulse, sets pins
        sbi PORTB,2 ; set SRCLK to 1
        nop ; delay
        nop
        nop
        nop
        cbi PORTB,2 ; set back to 0

        dec R17
        brne loop

        ; generate RCLK pulse, parallel output 8 bits in register
        sbi PORTB,3
        nop
        nop
        nop
        cbi PORTB,3
```

```
        ; restore registers from stack
        pop R17
        out SREG, R17
        pop R17
        pop display_value

        ret

delay:
        delay_1s:
        ldi r27, 100
        delay_1s_cont:
        cpi r27, 0
        rcall delay_10ms
        dec r27
        brne delay_1s_cont
        ret
;delay for 10ms
        delay_10ms:
        ldi r24, 0b00000101 ; Set prescaler to 1024
        out TCCR0B, r24
        ldi r24,100
        out TCNT0, r24 ; Set timer count to 100
; Wait for TIMER0 to roll over.
        delay_cont:
; Stop timer 0.
        in r30,TCCR0B ; Save configuration
        ldi r31,0x00 ; Stop timer 0
        out TCCR0B,r31
; Clear overflow flag.
        in r31,TIFR0 ; tmp <-- TIFR0
        sbr r31,1<<TOV0 ; Clear TOV0, write logic 1
        out TIFR0,r31
; Start timer with new initial count
        out TCNT0,r24 ; Load counter
        out TCCR0B,r30 ; Restart timer
        wait:
        in r31,TIFR0 ; tmp <-- TIFR0
        sbrs r31,TOV0 ; Check overflow flag
        rjmp wait
        ret
```

6. Appendix B: References

Beichel, Reinhard. Arduino_V2.pdf, Spring 2025. The University of Iowa. 03/10/2025. <https://uiowa.instructure.com/courses/248357/files/29320694?module_item_id=804231>

Beichel, Reinhard. Rotary Pulse Generators & Lab 3 used in ECE:3360 Embedded Systems, Spring 2025. The University of Iowa. 03/10/2025. <https://uiowa.instructure.com/courses/248357/files/29778930?module_item_id=816215>

Beichel, Reinhard. Lecture 8: Timers used in ECE:3360 Embedded Systems, Spring 2025. The University of Iowa. 03/10/2025. <https://uiowa.instructure.com/courses/248357/files/29853139?module_item_id=816571>

“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