**Lab 2 Report**

Sage Marks, Matt Krueger

3360:0001 - Embedded Systems

Professor Beichel

University of Iowa, College of Engineering

## Introduction

The lab builds a hexadecimal up/down counter. It uses an ATmega328P microcontroller, an 8-bit shift register, a 7-segment LED display, and a pushbutton switch. The 7-segment display shows "0" at power on, and the counter increments by default. The push button controls mode selection, increment/decrement, and reset. A push button press lasting for less than one second will increment or decrement the count based on your current mode. Pressing for one to two seconds will switch between increment and decrement modes. A press that exceeds two seconds will reset the count to 0 and you will once again be in increment mode. All actions are executed when the button has been released.

## Schematic

A computer screen shot of a circuit board

AI-generated content may be incorrect.

Figure 1: Electrical circuit schematic created using KiCAD

**Materials List**

|  |  |  |
| --- | --- | --- |
| Hardware | Quantity | Description |
| Atmega 328P µC | 1 | Programmable µC |
| 74HC595 Shift Register | 1 | Storage of hex codes for 7-Segment display |
| 5161AS 7-Segment Display | 1 | Display current counter |
| Enable Low Push Button | 1 | Enables user interaction with 7-Segment display |
| 560Ω Resistor | 8 | Resist current into 7-Segment display LEDs |
| 10KΩ Resistor | 1 | Pull up resistor for push button |
| 100KΩ Resistor | 1 | Pull up resistor for push button |
| 0.1µF Capacitor | 2 | Decoupling capacitors for button and µC |

Figure 2: Materials List

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

The design of this circuit successfully utilizes the ATmega328P and a 7-segment display with the use of a shift register. There are four I/O lines configured in the schematic, PB0 is configured as an output and as SER, PB1 as an output for RCLK, PB2 as an output or SRCLK, and PB3 as an input for a low signal from the button press. This is an active low push button meaning that when the push button is pressed the logic level goes to 0.

Our design utilizes a hardware-based debounce approach for the push button. This approach consists of a pull-up resistor (10KΩ) to keep the node at a defined state when the button is not pressed and an RC low pass filter with a 0.1 microfarad capacitor and a 10KΩ resistor. The low pass filter helps with possible oscillation that can occur when there is a button press, so the Arduino can recognize a press correctly and not increment/decrement when it is not supposed to.

The SRCLR and the OE lines of the shift register were taken to VCC and GND respectively. To wire the shift register to the 7-segment display for correct functionality it was important to know that the display is common cathode. This led us to tying pins 3 and 8 to ground. The QA line went to port A on the display, Qb went to port b, and continued until QH went to the decimal point.

It was important for us to have current limiting resistors in between the shift register and the display so we did not burn out the LED’s inside the display. To find the value of these resistors we consulted the data sheet for the 5161AS 7-segment display. The design specifications stated the current should not exceed 6mA. In the data sheet it states that the forward voltage drop associated with the LED is 1.8V. So to find the resistor value to get 6mA current we can use Ohm’s law defined as,

The voltage being (5-1.8) V and the current being 6 x 10^-3 A. Solving for R you get 533.33Ω. We rounded this value up to remain less than or equal to 6mA and utilized 560Ω resistors as stated in figure 2.

## 3. Discussion

**Functionality**

As stated in the abstract, the 7-Segment display is controlled by user via interacting with a push button. There are three modes, and their descriptions are as follows:

i) Increment Mode

- Press button < 1 second: Increment count by 1

- When display shows "F": Next increment rolls over to "0"

ii) Decrement Mode

- Press button < 1 second: Decrement count by 1

- When display shows "0": Next decrement rolls over to "F"

- DP LED on to indicate decrement mode

iii) Reset

- Press and hold button ≥ 2 seconds: Reset to "0" and enter increment mode

#TODO – ADD IMAGES OF EACH MODE

## 4. Conclusion

## 5. Appendix A: Source Code

; ---- main.asm (Embedded Systems Lab 2 - Spring 2025)

;

; Purpose:

; This Assembly file contains functionality for a 7-Segment display controlled by a 74hc595 shift register IC.

; Additional functionality is implemented via an active-low push button. The 7-Segment displays a sequence of hexidecimal numbers.

;

; Functionality of the 7-Segment display:

; 1. increment count (0,1,..,e,f)

; 2. decrement count (f,e,..,1,0)

; 3. reset count (show 0)

;

; Authors:

; - Sage Marks

; - Matt Krueger

; ---- I/O Configuration

;

; port assignments:

; SER <- PB0 (output)

; RCLK <- PB1 (output)

; SRCLK <- PB2 (output)

; PBTN -> PB3 (input)

sbi DDRB, 0

sbi DDRB, 1

sbi DDRB, 2

cbi DDRB, 3

ldi R21, 0; this register keeps track of the number being displayed in decimal

ldi R22, 0; this register keeps track of if the register is in increment or decrement mode (0=increment) (1=decrement)

ldi R26, 0xe8; this and register 29 are used to keep track of amount of time button is pressed for (initialized to decimal 1000 together)

ldi R27, 0x03; this and register 28 are used to keep track of amount of time button is pressed for (initialized to decimal 1000 together)

IncNumberCheck: ;Main loop that handles checking and displaying numbers in increment mode

rcall ButtonCheck; check if the button is being pressed

sbrc R22, 0; check if the first bit in R22 (mode checker) is clear

rjmp DecNumberCheck; if the bit is not clear we should jump to decrement mode

cpi R21, 0x00; checks if number tracker is set to 0

breq disp0;

cpi R21, 0x01; checks if number tracker is set to 1

breq disp1;

cpi R21, 0x02; checks if number tracker is set to 2

breq disp2;

cpi R21, 0x03; checks if number tracker is set to 3

breq disp3

cpi R21, 0x04; checks if number tracker is set to 4

breq disp4;

cpi R21, 0x05; checks if number tracker is set to 5

breq disp5;

cpi R21, 0x06; checks if number tracker is set to 6

breq disp6;

cpi R21, 0x07; checks if number tracker is set to 7

breq disp7;

cpi R21, 0x08; checks if number tracker is set to 8

breq disp8;

cpi R21, 0x09; checks if number tracker is set to 9

breq disp9;

cpi R21, 0x0a; checks if number tracker is set to a

breq dispA;

cpi R21, 0x0b; checks if number tracker is set to b

breq dispb;

cpi R21, 0x0c; checks if number tracker is set to c

breq dispC;

cpi R21, 0x0d; checks if number tracker is set to d

breq dispd;

cpi R21, 0x0e; checks if number tracker is set to e

breq dispE;

cpi R21, 0x0f; checks if number tracker is set to f

breq dispf;

rjmp IncNumberCheck; jumps back to start of the loop

;load corresponding pattern for hex number into R16

;jump to display function and then back to loop

disp0:

ldi R16, 0x3f; 0

rjmp IncDisp;

disp1:

ldi R16, 0x06; 1

rjmp IncDisp;

disp2:

ldi R16, 0x5b; 2

rjmp IncDisp;

disp3:

ldi R16, 0x4f; 3

rjmp IncDisp;

disp4:

ldi R16, 0x66; 4

rjmp IncDisp;

disp5:

ldi R16, 0x6d; 5

rjmp IncDisp;

disp6:

ldi R16, 0x7d; 6

rjmp IncDisp;

disp7:

ldi R16, 0x07; 7

rjmp IncDisp;

disp8:

ldi R16, 0x7f; 8

rjmp IncDisp;

disp9:

ldi R16, 0x6f; 9

rjmp IncDisp;

dispA:

ldi R16, 0x77; A

rjmp IncDisp;

dispb:

ldi R16, 0x7c; b

rjmp IncDisp;

dispC:

ldi R16, 0x39; C

rjmp IncDisp;

dispd:

ldi R16, 0x5e; d

rjmp IncDisp;

dispE:

ldi R16, 0x79; E

rjmp IncDisp;

dispf:

ldi R16, 0x71; f

rjmp IncDisp;

IncDisp: ;displays the value for increment numbers and then jumps back to increment loop

rcall display;

rjmp IncNumberCheck;

DecNumberCheck: ;Main loop that handles checking and displaying decrement mode numbers

rcall ButtonCheck; call subroutine to check button press

sbrs R22, 0; skip if the 0 bit is set (we are in decrement mode)

rjmp IncNumberCheck; If the 0 bit is not set (is 0) we are in increment mode and we go to increment loop

cpi R21, 0x00; checks if number tracker is at 0

breq disp0Dec;

cpi R21, 0x01; check if number tracker is at 1

breq disp1Dec;

cpi R21, 0x02; check if number tracker is at 2

breq disp2Dec;

cpi R21, 0x03; check if number tracker is at 3

breq disp3Dec;

cpi R21, 0x04; check if number tracker is at 4

breq disp4Dec;

cpi R21, 0x05; check if number tracker is at 5

breq disp5Dec;

cpi R21, 0x06; check if number tracker is at 6

breq disp6Dec;

cpi R21, 0x07; check if number tracker is at 7

breq disp7Dec;

cpi R21, 0x08; check if number tracker is at 8

breq disp8Dec;

cpi R21, 0x09; check if number tracker is at 9

breq disp9Dec;

cpi R21, 0x0a; check if number tracker is at a

breq dispADec;

cpi R21, 0x0b; check if number tracker is at b

breq dispbDec;

cpi R21, 0x0c; check if number tracker is at c

breq dispCDec;

cpi R21, 0x0d; check if number tracker is at d

breq dispdDec;

cpi R21, 0x0e; check if number tracker is at e

breq dispEDec;

cpi R21, 0x0f; check if number tracker is at f

breq dispfDec;

rjmp DecNumberCheck;

disp0Dec:

ldi R16, 0xbf; 0 with decimal

rjmp DispDec;

disp1Dec:

ldi R16, 0x86; 1 with decimal

rjmp DispDec;

disp2Dec:

ldi R16, 0xdb; 2 with decimal

rjmp DispDec;

disp3Dec:

ldi R16, 0xcf; 3 with decimal

rjmp DispDec;

disp4Dec:

ldi R16, 0xe6; 4 with decimal

rjmp DispDec;

disp5Dec:

ldi R16, 0xed; 5 with decimal

rjmp DispDec;

disp6Dec:

ldi R16, 0xfd; 6 with decimal

rjmp DispDec;

disp7Dec:

ldi R16, 0x87; 7 with decimal

rjmp DispDec;

disp8Dec:

ldi R16, 0xff; 8 with decimal

rjmp DispDec;

disp9Dec:

ldi R16, 0xef; 9 with decimal

rjmp DispDec;

dispADec:

ldi R16, 0xf7; A with decimal

rjmp DispDec;

dispbDec:

ldi R16, 0xfc; b with decimal

rjmp DispDec;

dispCDec:

ldi R16, 0xb9; C with decimal

rjmp DispDec;

dispdDec:

ldi R16, 0xde; d with decimal

rjmp DispDec;

dispEDec:

ldi R16, 0xf9; E with decimal

rjmp DispDec;

dispfDec:

ldi R16, 0xf1; f with decimal

rjmp DispDec;

DispDec:; Displays the value for decrement numbers and then jumps back to the loop

rcall display;

rjmp DecNumberCheck;

ButtonCheck:

sbic PINB, 3; skip if button is pressed (if line is low skip) (a button press makes the line low)

ret; (button not pressed jump back to display loop)

ButtonPressLoop:

rcall Delay; call the 1 milisecond delay function (means we are checking the button for a press ~1 ms intervals)

sbiw R27:R26, 1; subtract 1 from the registers that hold a value of 1000ms (1 second)

breq OneToTwo; If the button is pressed for long enough that register is cleared (1 second has passed) branch to 1 to 2 second

sbis PINB, 3; skip if the button has been released (line is back to high)

rjmp ButtonPressLoop; keep looping for checking if button is released

cpi R22, 1; Check if we are in decrement mode

breq DecButtonCheck; branch to dec button check

IncButtonCheck:

inc R21; increment register that is tracking display number

cpi R21, 0x10; compare if register value is 16 (f+1)

breq rolloverInc; if it is go to rollover increment logic

rjmp WaitForRelease;

rolloverInc:

ldi R21, 0x00; load 0 into the register tracking value (when we increment f it goes back to 0)

rjmp WaitForRelease

DecButtonCheck:

dec R21; decrement register that is tracking the display number

cpi R21, 0xff; compare if the register value is 255 (when we decrement 0 the register has value of 255)

breq rolloverDec; if equal go to rollover decrement logic

rjmp WaitForRelease;

rolloverDec:

ldi R21, 0x0f; load f into register tracking value (when we decrement 0 we go back to 0)

rjmp WaitForRelease;

OneToTwo:

ldi R26, 0xe8; resets counter to 1000 low byte (1 second limit)

ldi R27, 0x03; resets counter to 1000 high byte (1 second limit)

OneToTwoLoop:

rcall Delay; call delay function (1 ms)

sbiw R27:R26, 1; subtract 1 from register with 1000 value (this occurs every milisecond)

breq Reset; if the register value reaches 0 the button has been pressed for more than two seconds

sbis PINB, 3; skip if the button has been released (line is back to high)

rjmp OneToTwoLoop; keep looping to check for button release

cpi R22, 1; if we are in decrement mode (button was pressed for 1 to 2 seconds)

breq IncMode; branch to switch to increment mode

ldi R22, 1; switch to decrement mode (because we are in increment mode)

rjmp WaitForRelease;

Reset:

ldi R22, 0; reset, we are in increment mode

ldi R21, 0; display 0

rjmp WaitForRelease;

IncMode:

ldi R22, 0; function for switching to increment mode (1 to 2 second button press)

WaitForRelease:

sbis PINB, 3; skip if the button has been released (line is back to high)

rjmp WaitForRelease; loops so that action does not occur until the button is released

ldi R26, 0xe8; resets counter to 1000 low byte (1 second limit)

ldi R27, 0x03; resets counter to 1000 high byte (1 second limit)

ret;

; ---- Display

;

; Output hexidecimal bit representation to 7-Segment display utilizing stack and 74hc595 shift register for storage

display:

push R16; put registers on the stack (last in first out system)

push R17

in R17, SREG

push R17

ldi R17, 8; Load 8 for 8 bits

; Loop

loop:

rol R16; rotate left with carry (covers each bit)

BRCS set\_ser\_in\_1; branch if the carry is set

cbi PORTB, 0; sets serial line to 0 (0 is the value of the bit being sent)

rjmp end

; Set SER Input High

set\_ser\_in\_1:

sbi PORTB, 0; sets the serial line to 1 (1 is the value of the bit being sent)

; End

end:

sbi PORTB, 2; sets SRCLK to high (cycles bit through the shift register)

cbi PORTB, 2; sets SRCLK to low (only cycles one bit at a time then it checks the carry bit)

dec R17; decrement R17 until it is 0

brne loop; branch to loop checking carry bit

sbi PORTB, 1; sets RCLK to high (puts bit values on output that goes to display)

cbi PORTB, 1; sets RCLK to low (so we can change 7 segment display when we call display again)

pop R17; take registers off the stack

out SREG, R17

pop R17

pop R16

ret;

;This delay loop lasts for ~1ms (check button every ms)

Delay:

ldi r30, 0x5d ; r31:r30 <-- load a 16-bit value into counter register for outer loop

ldi r31, 0x00;

d1:

ldi r29, 0x2a ; r29 <-- load a 8-bit value into counter register for inner loop

d2:

nop ; no operation

dec r29 ; r29 <-- r29 - 1

brne d2 ; branch to d2 if result is not "0"

sbiw r31:r30, 1 ; r31:r30 <-- r31:r30 - 1

brne d1 ; branch to d1 if result is not "0"

ret;

## 6. Appendix B: References

Need to include data sheets for:

Shift register

7-segment display

AVR microcontroller

ADD references to slides and course webpage