Cole Brooks

Thomas Butler

55:036 Embedded Systems

Post-Lab Report 3

Introduction

The goal of this lab was to gain some experience using the advanced timer/counter functionality of the ATMega328P specifically relating to pulse width modulation (PWM), interrupts, and LCDs.

In this lab, we will construct a brightness adjustable LED light source. For user interaction, the device will utilize an LCD, a rotary pulse generator (RPG), and a pushbutton switch (PBS). The device will adjust the duty cycle of a transistor which will lead to an adjustment in the brightness of the LEDs. To adjust the brightness of the LEDs, the user will rotate the RPG. In order to increase the brightness, the user will rotate the RPG clockwise. In order to decrease the brightness, the user will rotate the RPG counterclockwise. The user will also be able to use the PBS to turn off or turn on the LED. Also, the device will display information about the current state of the LEDs on an LCD screen.

Schematic

Discussion

The Arduino used in the circuit was programmed with AVR assembly source code. That source code is included in the Source Code section of this lab report. An sn74hc595 shift register was used, in order to increase the outputs of the Arduino.

To complete our project, we verified the following requirements from the lab description:

|  |  |
| --- | --- |
| **Requirement** | **Status:** |
| When LEDs are on and the rotary pulse generator is spun counterclockwise, the Duty Cycle increases |  |
| When LEDs are on and the rotary pulse generator is spun counterclockwise, the Duty Cycle decreases |  |
| When the PBS is pressed the LEDs are toggled on/off |  |
| When initially turning the device on, the default DC is set to 50% |  |
| After first use, the device should default to the previously set DC |  |
| The duty cycle must be adjustable between 5% - 100% in 5% increments |  |
| The duty cycle must only be adjustable if the LEDs are on |  |
| A pulse width modulation signal for Q1 must be generated by the microcontroller with one of the 8-bit timers/counters using the waveform generation functionality. |  |
| The pulse width modulation frequency must be 1.25 kHz (fixed) |  |
| At all times, the state of the device must be displayed on the LCD display  The state includes:   * Duty cycle * Status of LEDs |  |

The planning for this lab occurred entirely online. Like the past labs, our primary mode of communication was an online messaging platform, Discord. We were short on time in the previous lab, so we elected to start and complete this circuit on our own timeline opposed to simply keeping up with the deadlines imposed by the professor.

A preliminary step for this lab was to test our LCD components with the given LCD test program. This was an important step because it proved that our hardware would behave as intended during development of the circuit. It also gave us a reference to work off of.

The first step for this lab was to get a circuit set up properly, and simple source code developed such that the LCD display showed proper formatting for the circuit. As with previous labs, we felt it was important to get the hardware correct first, as the software is typically easier and less time intensive to change and debug. For this reason, we tried to keep the software component for this circuit as simple as possible to confirm that the hardware behaved as expected. The plan for doing so was to implement code which would display the proper format on the LCD, allowing the user to change the duty cycle with the RPG and toggle the status of the LEDs with the PBS. These changes utilized simple delays like those used in labs 1 and 2 instead of using the peripheral timer/counter delays which are needed for proper functionality of the circuit. We elected to use such delays because they are familiar to us and quite simple to implement quickly. Note that user input did not change the LEDs at this point in the lab. The LCD screen included in this circuit was unfamiliar to us, and as such we thought it prudent to plan extra time for ensuring the circuit is set up properly. This step of the lab was to be completed by Tuesday, March 30th.

On March 31st, we had our first lab check-in. The requirement for this check in was to demonstrate that our program could successfully display different strings on the LCD. At this point in time our circuit was functioning well enough to do so.

The next step for this lab was to update the simple source code to allow for more complete functionality of the circuit. In this step, we consulted with TA’s in office hours to ensure that we were using timers properly, as that was a challenge in the previous lab and converting from delays of nearly identical lengths should theoretically not impact the circuit. After doing so, we implemented mechanisms to allow for actual control of the LEDs. This step of the lab was to be done by Friday, April 2.

The last, and possibly most crucial step for this lab was to debug the circuit to bring functionality up to the standards set forth in the lab description (found in the table above). We found that in lab 2, we worked down to the last second because we did not plan adequate time for debugging the circuit after typical development had completed, and as such the stability of the circuit was questionable. We planned for 4 days of debug time which we thought was enough to get the circuit prepared for a final release.

After confirmation that the circuit behaved as intended, we met with a TA in office hours and had our final check-in.

Calculations

Conclusion

Appendix A: Source Code

; REGISTERS:

; R16 - used in binary. 8 bits map to the 8 leds on the 7-seg

.def s\_seg = r16

; R17 - used to test 8 bits in the display function

.def displayret = r18 ; if 1, go to incorrect

; R19 - counter register to keep track of counter so that display doesn't go out of range

.def counter = r19

; R20 - keeps track of current state

.def state = r20

; R21 - keeps track of last state

.def old\_state = r21

; R22

; R23->R25 - Buffer registers used in delay loops and other temp storage

.def tmp1 = r23

.def tmp2 = r24

; R26 - counter for pushbutton debounce

; R27 - holds number of values entered

; R28 - holds values of correct code

.include "m328Pdef.inc"

.cseg

.org 0

; define macros

.macro SET\_STATE

.set count = 10 ; 2ms

ldi r30, low(count)

ldi r31, high(count)

rcall delay\_TC0

ldi old\_state, 0

OR old\_state, state

ldi state, 0

sbis PIND, 2

sbr state, (1<<0)

sbis PIND, 3

sbr state, (1<<4)

.endmacro

; code to configure I/O lines

sbi DDRB,0 ; pin 8 as output -> SER

sbi DDRB,1 ; pin 9 as output -> SRCLK

sbi DDRB,2 ; pin 10 as output -> RCLK

sbi DDRB,5 ; turn led L off

; need to configure input lines?

; PIND,2 -> A

; PIND,3 -> B

; PIND,4 -> Button

;ldi s\_seg, 0b10000000 ; s\_seg is the display register. Whatever is loaded here will be displayed on the 7-seg

ldi counter, 0xff ; negative, needs to display '-' to start

ldi r27,0 ; start with 0 entries

ldi r28,0 ; will be loaded with code values as code is entered

ldi state, 0x00 ; state starts in normal (no input) state

ldi old\_state, 0x00 ; old\_state should start at normal state, too

ldi displayret,0

rcall display;

; configure TC0

ldi tmp1,0x00; normal mode

out TCCR0A, tmp1

ldi tmp1,0x03 ; 1/256 prescale

out TCCR0B,tmp1

ldi tmp1,0x00

out TCNT0,tmp1

display:

; backup used registers on stack

push s\_seg

push R17

in R17, SREG

push R17

ldi R17, 8 ; test all 8 bits

loop:

rol s\_seg ; rotate left

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

cbi PORTB,0 ; set SER to 0 (correct?)

rjmp end

set\_ser\_in\_1:

; code to set SER to 1

sbi PORTB,0 ; set SER to 1 (c orrect?)

end:

; code to generate SRCLK pulse (correct?)

cbi PORTB,1

nop

sbi PORTB,1

dec R17

brne loop

; code to generate RCLK pulse (correct?)

cbi PORTB,2

nop

sbi PORTB,2

; restore registers from stack

pop R17

out SREG, R17

pop R17

pop s\_seg

rjmp main ; return to main loop

main:

;debug

/\*

rcall incorrect\_delay

;end debug \*/

; check if we need to delay due to entry

cpi displayret,1

breq correct\_delay

cpi displayret,2

breq incorrect\_delay

; check for no input first - set state back to normal

rcall state\_machine

; check for button push

sbic PIND,5 ; skip if button not pressed

rjmp buttonpush

bounce:

rjmp set\_display

display\_ready:

rjmp wait\_for\_zero

zeroed:

rjmp display ; call display again, leads to infinite loop

wait\_for\_zero:

SET\_STATE

cpi state, 0x00

breq zeroed

rjmp wait\_for\_zero

buttonpush:

ldi R26,0 ; count presses

debounce:

rcall delay\_100\_ms

sbic PIND,5

inc R26

sbic PIND,5

rjmp debounce ; continue looping while pressed

cpi R26,30 ; not actually 3 seconds -- fix delays

brsh reset ; pushed for longer than 3s -> reset

cpi R26, 2 ; 200 ms

brlt bounce

rjmp acceptval

reset:

ldi counter, 0xff

ldi r27,0

rjmp set\_display

acceptval:

ADD r28, counter ; add value of counter to r28

inc r27

cpi r27,4 ; 4 entries

breq attemptunlock

rjmp main

attemptunlock:

ldi r27,0 ; reset entries

cpi r28,32 ; correct code 5+3+f+8 = 31 + 1 starts at -1

breq correct

cpi r28,32

brne incorrect

rjmp main

correct:

;ldi counter, 0xff

ldi s\_seg, 0b00000001

ldi displayret,1

rjmp display

incorrect:

ldi s\_seg, 0b0100000

ldi displayret,2

rjmp display

correct\_delay:

sbi PORTB,5 ; turn led L on

;.set count = 25000 ; 5s

;ldi r30, low(count)

;ldi r31, high(count)

;rcall delay\_TC0

sbi PORTB,5

ldi r30, 5

start\_delay:

rcall delay\_1s

dec r30

brne start\_delay

rjmp entry\_done

incorrect\_delay:

.set count = 60000 ; 12s

ldi r30, low(count)

ldi r31, high(count)

rcall delay\_TC0

rjmp entry\_done

entry\_done:

cbi PORTB,5

ldi displayret,0

state\_machine:

; check state. Only move forward in state machine if state = 0x11

SET\_STATE

cpi state, 0x01

breq increment

cpi state, 0x10

breq decrement

sm\_done:

ret

increment:

cpi counter, 0x10

breq sm\_done

inc counter

rjmp sm\_done

decrement:

cpi counter, 0x00

breq sm\_done

dec counter

rjmp sm\_done

delay\_TC0:

rcall delay\_helper\_TC0

sbiw Z,1

brne delay\_TC0

ret

delay\_helper\_TC0:

in tmp2,TCNT0 ; Get counter value

cpi tmp2,0x00 ; is it zero?

brne delay\_helper\_TC0 ; no --> wait some more

ret ; yes --> continue on

delay\_1s: ; actually 1s??

ldi r23,100 ; r23 <-- Counter for outer loop

onesec1: ldi r24,200 ; r24 <-- Counter for level 2 loop

onesec2: ldi r25,199 ; r25 <-- Counter for inner loop

onesec3: dec r25

nop ; no operation

brne onesec3

dec r24

brne onesec2

dec r23

brne onesec1

ret

delay\_100\_ms: ; is this actually 100 ms??

ldi r23,1 ; r23 <-- Counter for outer loop

hms1: ldi r24,200 ; r24 <-- Counter for level 2 loop

hms2: ldi r25,199 ; r25 <-- Counter for inner loop

hms3: dec r25

nop ; no operation

brne hms3

dec r24

brne hms2

dec r23

brne hms1

ret

start:

ldi s\_seg, 0b00000001

rjmp done

zero:

ldi s\_seg, 0b01111110

rjmp done

one:

ldi s\_seg, 0b01001000 ;load new display into s\_seg

rjmp done ;jump to the end of the case statement

two:

ldi s\_seg, 0b00111101

rjmp done

three:

ldi s\_seg, 0b01101101

rjmp done

four:

ldi s\_seg, 0b01001011

rjmp done

five:

ldi s\_seg, 0b01100111

rjmp done

six:

ldi s\_seg, 0b01110111

rjmp done

seven:

ldi s\_seg, 0b01001100

rjmp done

eight:

ldi s\_seg, 0b01111111

rjmp done

nine:

ldi s\_seg, 0b01101111

rjmp done

A:

ldi s\_seg, 0b01011111

rjmp done

B:

ldi s\_seg, 0b01111111

rjmp done

C:

ldi s\_seg, 0b00110110

rjmp done

D:

ldi s\_seg, 0b01111110

rjmp done

E:

ldi s\_seg, 0b00110111

rjmp done

F:

ldi s\_seg, 0b00010111

rjmp done

set\_display:

cpi counter, 0xff

breq start

cpi counter, 0x00

breq zero

cpi counter, 0x01

breq one

cpi counter, 0x02

breq two

cpi counter, 0x03

breq three

cpi counter, 0x04

breq four

cpi counter, 0x05

breq five

cpi counter, 0x06

breq six

cpi counter, 0x07

breq seven

cpi counter, 0x08

breq eight

cpi counter, 0x09

breq nine

cpi counter, 0x0A

breq A

cpi counter, 0x0B

breq B

cpi counter, 0x0C

breq C

cpi counter, 0x0D

breq D

cpi counter, 0x0E

breq E

cpi counter, 0x0F

breq F

done:

rjmp display\_ready.

Appendix B: References

sn74hc595.pdf, 3/10/21 < <https://uiowa.instructure.com/files/15017051/download?download_frd=1> >

5161AS.pdf, 3/10/21 < <https://uiowa.instructure.com/files/15017035/download?download_frd=1> >

ATmega328p, 3/10/21 < <https://uiowa.instructure.com/files/14761502/download?download_frd=1> >

AVR\_Assembler.pdf < <https://uiowa.instructure.com/files/14762083/download?download_frd=1> >

ARDUINO\_V2.pdf, 3/10/21 < <https://uiowa.instructure.com/files/14762893/download?download_frd=1> >

AVR\_Instruction\_Set\_Summary.pdf, 3/10/21 < <https://uiowa.instructure.com/files/14761192/download?download_frd=1> >