1. Introduction

The goal of Lab 2 was to create a counter with decrement, increment, and reset functionality. We accomplished this by using a 8 bit shift register, and a 7-segment LCD. We sampled the time that the button was pushed using a combination of subroutines for delays and register increments.

2. Schematic

Figure 1 shows the provided circuit design that we used to blink two LEDs.

Figure 1: Circuit with two LEDs on PB2 and PB1

3. Discussion

Most time was spent setting up our board and AVR drivers, but upon our first successful flash, we set to work in calculating our needed clock cycle count. To start, we enabled breakpoints to reveal the total cycle count of an unedited loop. We then used that cycle count to derive a formula made up of 3 variables (X1, X 2 , and X 3) to represent the increment counters of our nested loop registers (r23, r24, r25). After incorporating the cycle count of the “nop”, “brne”, “dec”, and “ldi” operations (each taking 4 counts aside from their final execution) we ended up with the following formula:

By using the default values of 41, 24, and 20, our C output was 2608690, which accurately represented the number given in simulation. We changed our variables around freely until we landed on a C value that was incredibly close to the required .2484s.

Our final values were r23 = 11, r24 = 226, and r25 = 249, as shown below from our formula calculation.

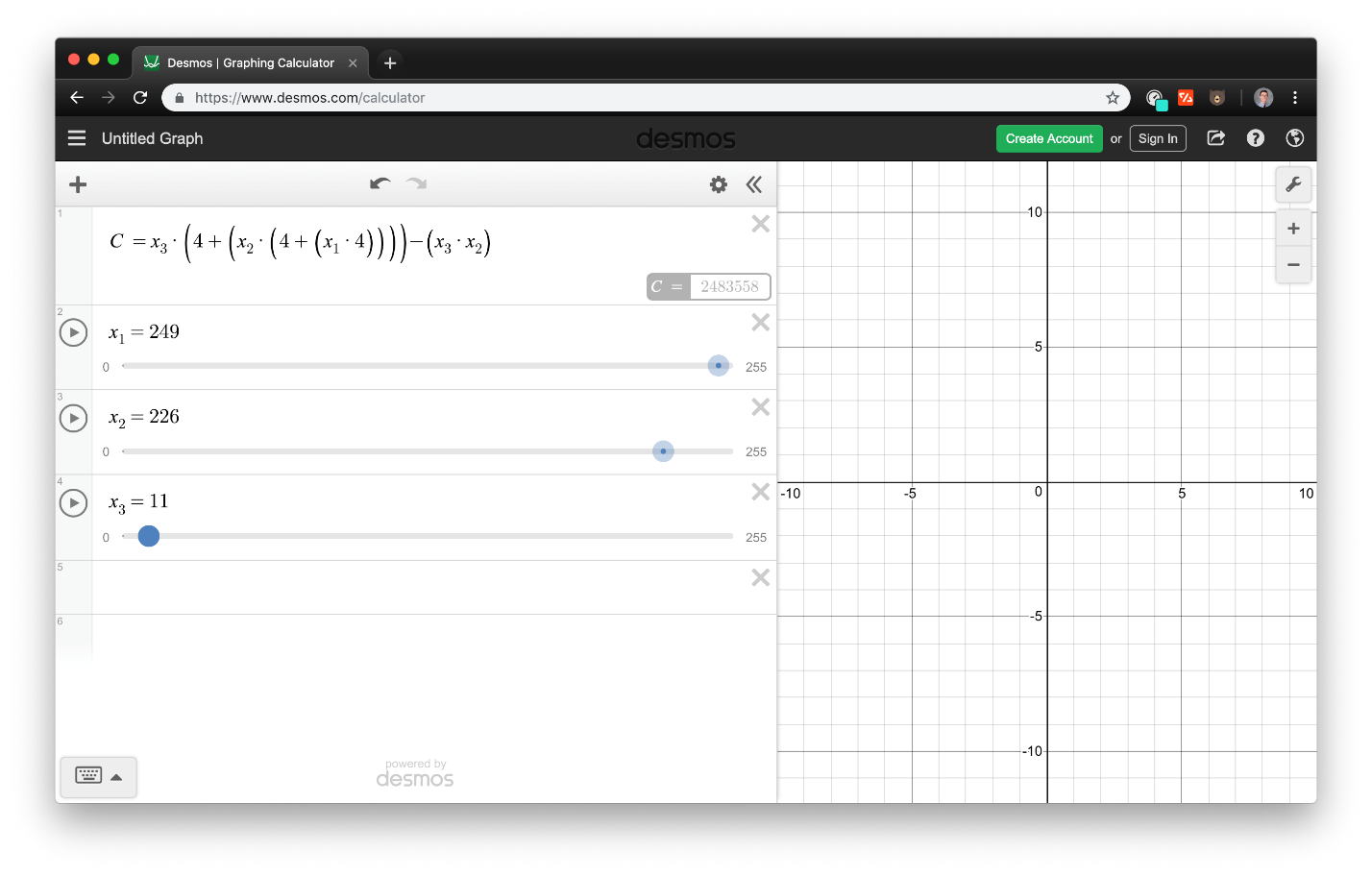


Figure 2: Our resultant calculation

4. Conclusion

In this lab, I grew comfortable calculating cycle counts of individual methods, as well as deploying and simulating assembly code on my own laptop. I also learned proper soldering methods for the first time, which was empowering to have confidence on when headed into future projects.

5. Appendix A: Source Code

A-1: main.asm

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Assembly language file for Lab 2 in ECE:3360 (Embedded Systems)

; Spring 2018, The University of Iowa.

; Desc: runs the control logic for a button controlled counter

; B. Mitchinson, A. Powers

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.include "tn45def.inc"

.include "disp\_values.inc" ; file with hex constants for 7-segment display

.dseg

.org 0x0060 ;Tiny45, Start in SRAM

disp\_table: .byte 16

.cseg

.org 0x0000

; initialize SRAM

ldi r20, ZERO\_DISP

sts disp\_table, r20

ldi r20, ONE\_DISP

sts disp\_table+1, r20

ldi r20, TWO\_DISP

sts disp\_table+2, r20

ldi r20, THREE\_DISP

sts disp\_table+3, r20

ldi r20, FOUR\_DISP

sts disp\_table+4, r20

ldi r20, FIVE\_DISP

sts disp\_table+5, r20

ldi r20, SIX\_DISP

sts disp\_table+6, r20

ldi r20, SEVEN\_DISP

sts disp\_table+7, r20

ldi r20, EIGHT\_DISP

sts disp\_table+8, r20

ldi r20, NINE\_DISP

sts disp\_table+9, r20

ldi r20, A\_DISP

sts disp\_table+10, r20

ldi r20, B\_DISP

sts disp\_table+11, r20

ldi r20, C\_DISP

sts disp\_table+12, r20

ldi r20, D\_DISP

sts disp\_table+13, r20

ldi r20, E\_DISP

sts disp\_table+14, r20

ldi r20, F\_DISP

sts disp\_table+15, r20

clr r20

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.equ SER\_IN = 0

.equ SRCK = 1

.equ RCK = 2

.equ PUSH\_BUTTON = 4

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.DEF DISP\_REG = r16

.DEF DISP\_INS\_REG = r17

.DEF PRESS\_TIME\_REG = r18

.DEF DEC\_REG = r19

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; ATiny - Register Relations

; PB0 = SER\_IN

; PB1 = SRCK

; PB2 = RCK

; PB4 = button input

; Configure PB0, PB1, and PB2 as output pins.

; Configure PB4 as input pins

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

sbi DDRB,0

sbi DDRB,1

sbi DDRB,2

cbi DDRB,4

ldi ZL, low(disp\_table) ;r30

ldi ZH, high(disp\_table) ;r31

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; main method -- infinite loop to keep the controller responding to input

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

rcall reset\_routine ; reset display state

main:

nop

sbis PINB, PUSH\_BUTTON ; if button pushed, next line will execute

rcall button\_pressed ; react to the button press

rjmp main

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

button\_pressed:

clr PRESS\_TIME\_REG ; set initial state of counter to zero

rcall count\_press

rjmp update

; count press sets certain register values associated with the length of button press

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

count\_press:

rcall sample\_delay

cpi PRESS\_TIME\_REG, 255

brlo increment\_time

sbis PINB, PUSH\_BUTTON ; if button is still pressed, execute next line

rjmp count\_press ; rjmp to this method

ret

increment\_time:

inc PRESS\_TIME\_REG

sbis PINB, PUSH\_BUTTON

rjmp count\_press

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; sample\_delay of 80000 cycles

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

sample\_delay:

ldi r23,20 ; r23 <-- Counter for outer loop

my\_d1: ldi r24,24 ; r24 <-- Counter for level 2 loop

my\_d2: ldi r25,41 ; r25 <-- Counter for inner loop

my\_d3: dec r25

nop ; no operation

brne my\_d3

dec r24

brne my\_d2

dec r23

brne my\_d1

ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; update the state based on the contents of PRESS\_TIME\_REG

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

update:

cpi PRESS\_TIME\_REG, 200

brsh reset\_routine

cpi PRESS\_TIME\_REG, 100

brsh toggle\_routine

rjmp move\_routine

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

reset\_routine:

nop

ldi ZL, 0x60

ld DISP\_REG, Z

rcall toggle\_dec\_off

rjmp main

toggle\_routine:

nop

cpi DEC\_REG, 0x00

breq toggle\_dec\_on

cpi DEC\_REG, 0x00

brne toggle\_dec\_off

rjmp main

move\_routine:

nop

; branching:

; if dec -> if overflow: edit r30 (or) else dec

; if inc -> if overflow: edit r30 (or) else inc

cpi r30, 0x6F

breq pos\_overflow

cpi r30, 0x60

breq neg\_overflow

cpi DEC\_REG, 0x01

brne pos\_counter

cpi DEC\_REG, 0x00

brne neg\_counter

rjmp main ; saftey

neg\_overflow:

cpi DEC\_REG, 0x01

breq bottom\_overflow

cpi DEC\_REG, 0x01

brne pos\_counter

rjmp main ; saftey

pos\_overflow:

cpi DEC\_REG, 0x00

breq reset\_routine

cpi DEC\_REG, 0x00

brne neg\_counter

rjmp main

pos\_counter:

ld DISP\_REG, Z+

ld DISP\_REG, Z

rcall display

rjmp main

neg\_counter:

ld DISP\_REG, -Z

ld DISP\_REG, Z

rcall display

rjmp main

bottom\_overflow:

ldi ZL, 0x6F

ld DISP\_REG, Z

rcall display

rjmp main

toggle\_dec\_on:

ldi DEC\_REG, 0x01

rcall display

rjmp main

toggle\_dec\_off:

ldi DEC\_REG, 0x00

rcall display

rjmp main

; Display subroutine that prints to the LCD the associate hex value in DISP\_REG

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

add\_dp:

sbr DISP\_REG, 8

rjmp dp\_return

remove\_dp:

cbr DISP\_REG, 8

rjmp dp\_return

display:

cpi DEC\_REG, 0x00

brne add\_dp

cpi DEC\_REG, 0x01

brne remove\_dp

dp\_return:

; backup used registers on stack

push DISP\_REG

push DISP\_INS\_REG

in DISP\_INS\_REG, SREG

push DISP\_INS\_REG

ldi DISP\_INS\_REG, 8

; loop --> test all 8 bits

loop:

rol DISP\_REG ;rotate left through Carry

BRCS set\_ser\_in\_1

; branch if Carry set

; put code here to set SER\_IN to 0

cbi PORTB, SER\_IN

rjmp end

set\_ser\_in\_1:

; put code here to set SER\_IN to 1

sbi PORTB, SER\_IN

end:

; put code here to generate SRCK pulse

sbi PORTB, SRCK

nop

nop

cbi PORTB, SRCK

dec DISP\_INS\_REG

brne loop

; put code here to generate RCK pulse

sbi PORTB, RCK

nop

nop

cbi PORTB, RCK

; restore registers from stack

pop DISP\_INS\_REG

out SREG, DISP\_INS\_REG

pop DISP\_INS\_REG

pop DISP\_REG

ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; end of program -- should never be reached due to main infinite loop

.exit