Lab 1 Summary

Schuster, Mark

Section 1540

09/12/2017

Pre-Lab Answers:

None

Problems encountered:

None in particular. All documents and resources were clear and informative!

Future Work/Applications:

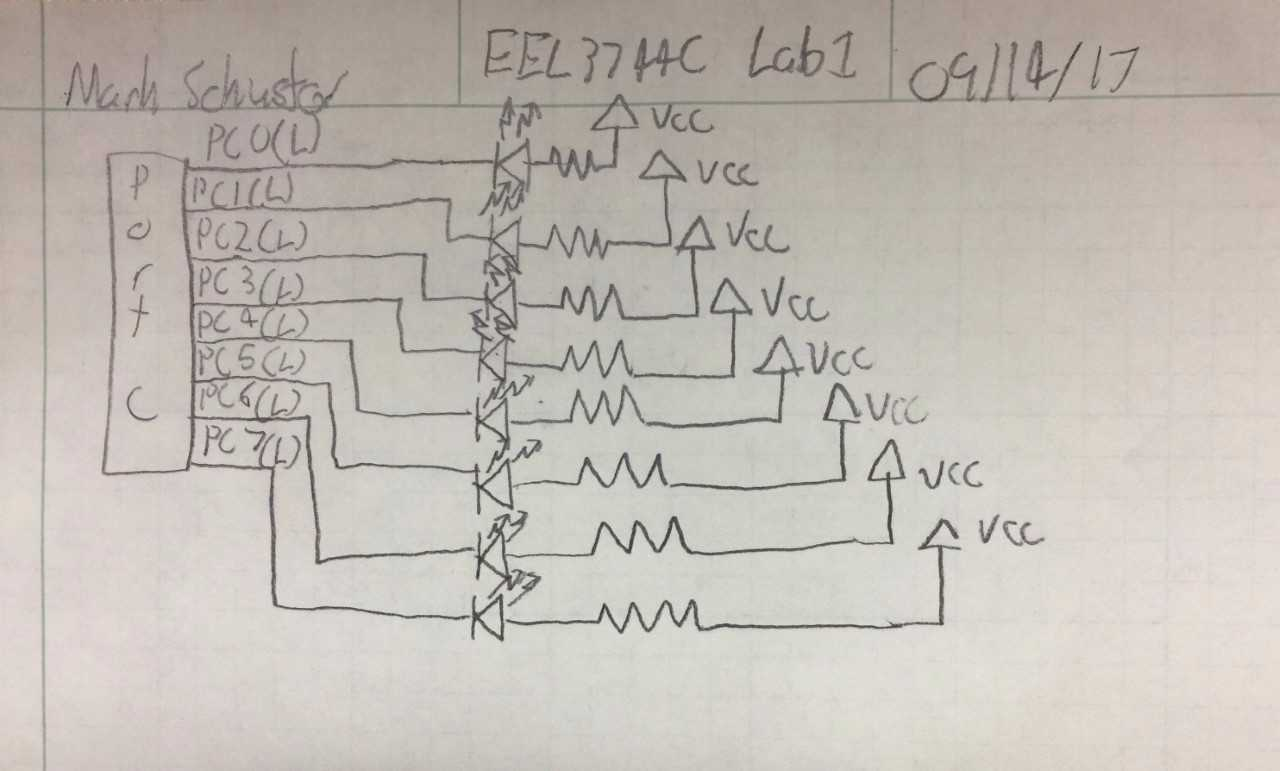
I really enjoyed using the LED backpack and I look forward to using the others.

Schematics:

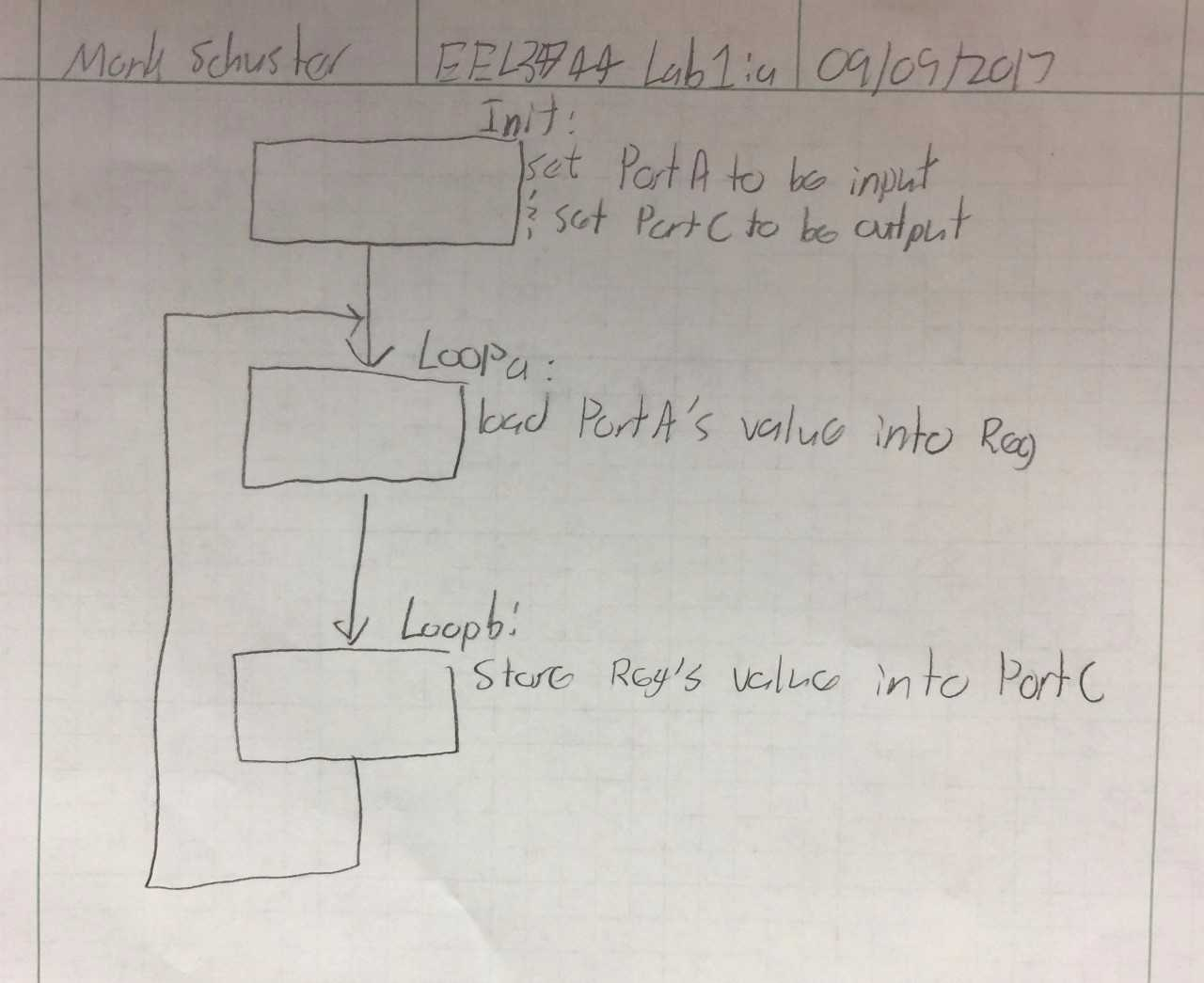
Not applicable for this lab.

Pseudocode/Flowcharts:

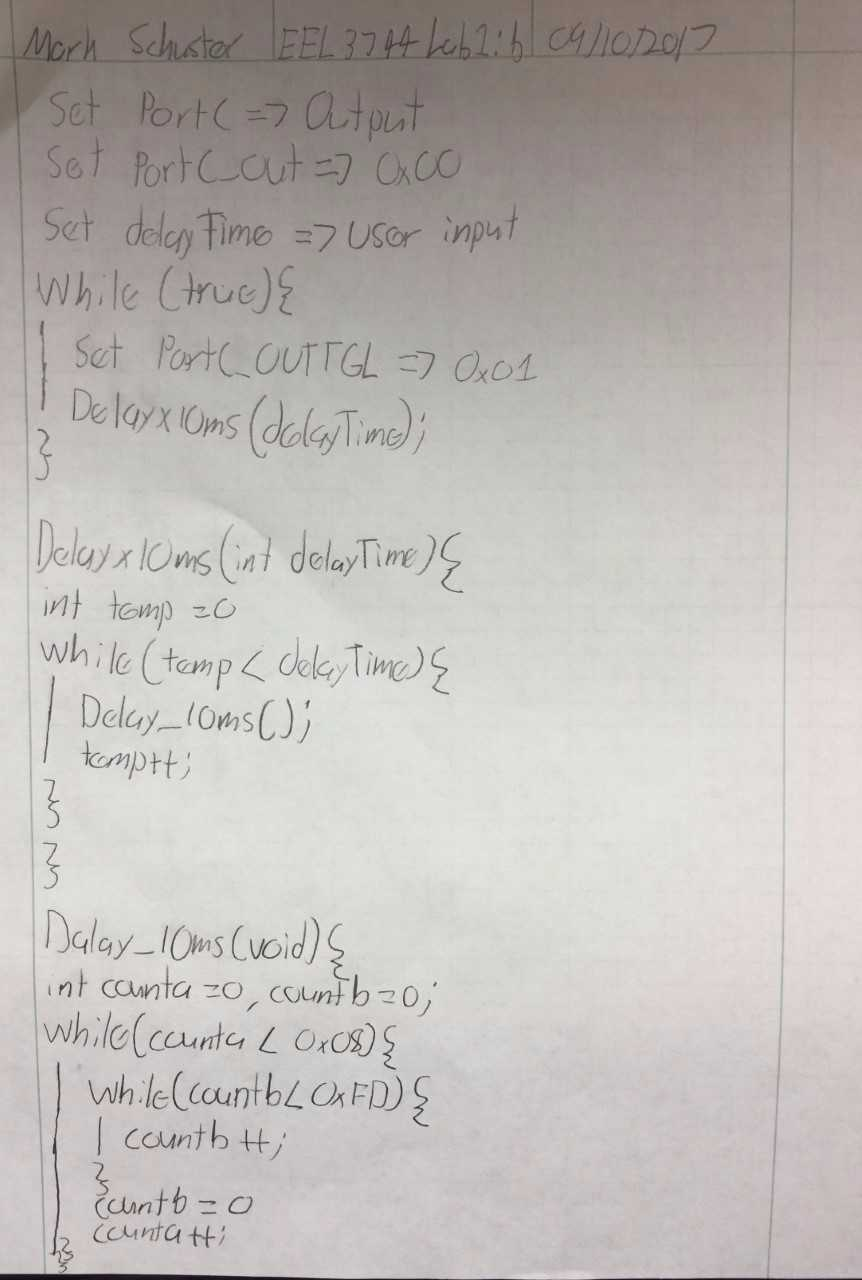
Switch circuit diagram:



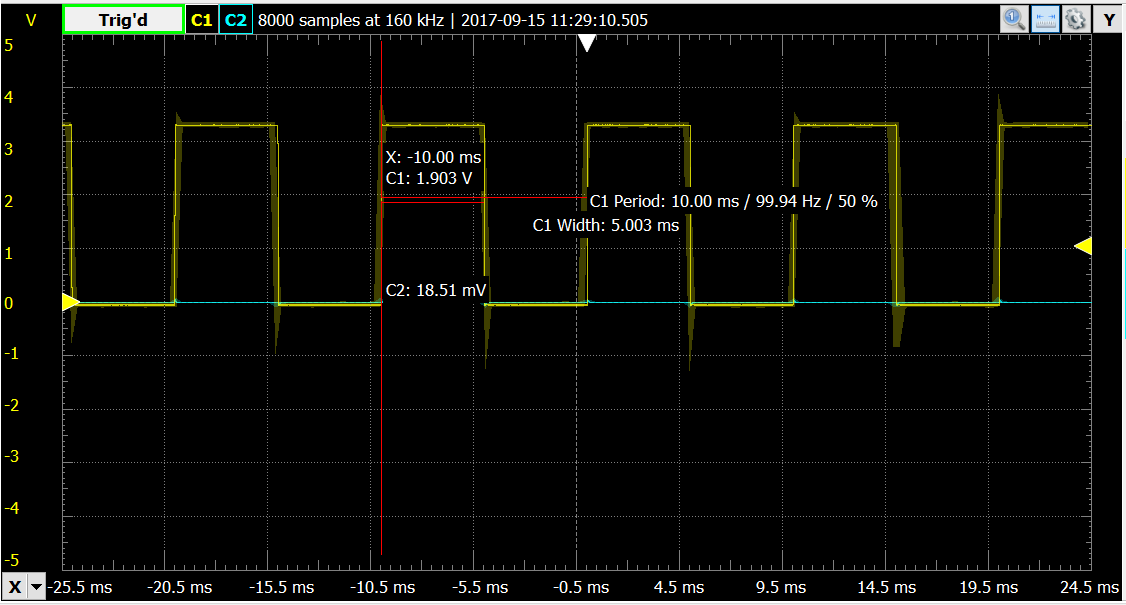
Part A flow diagram:



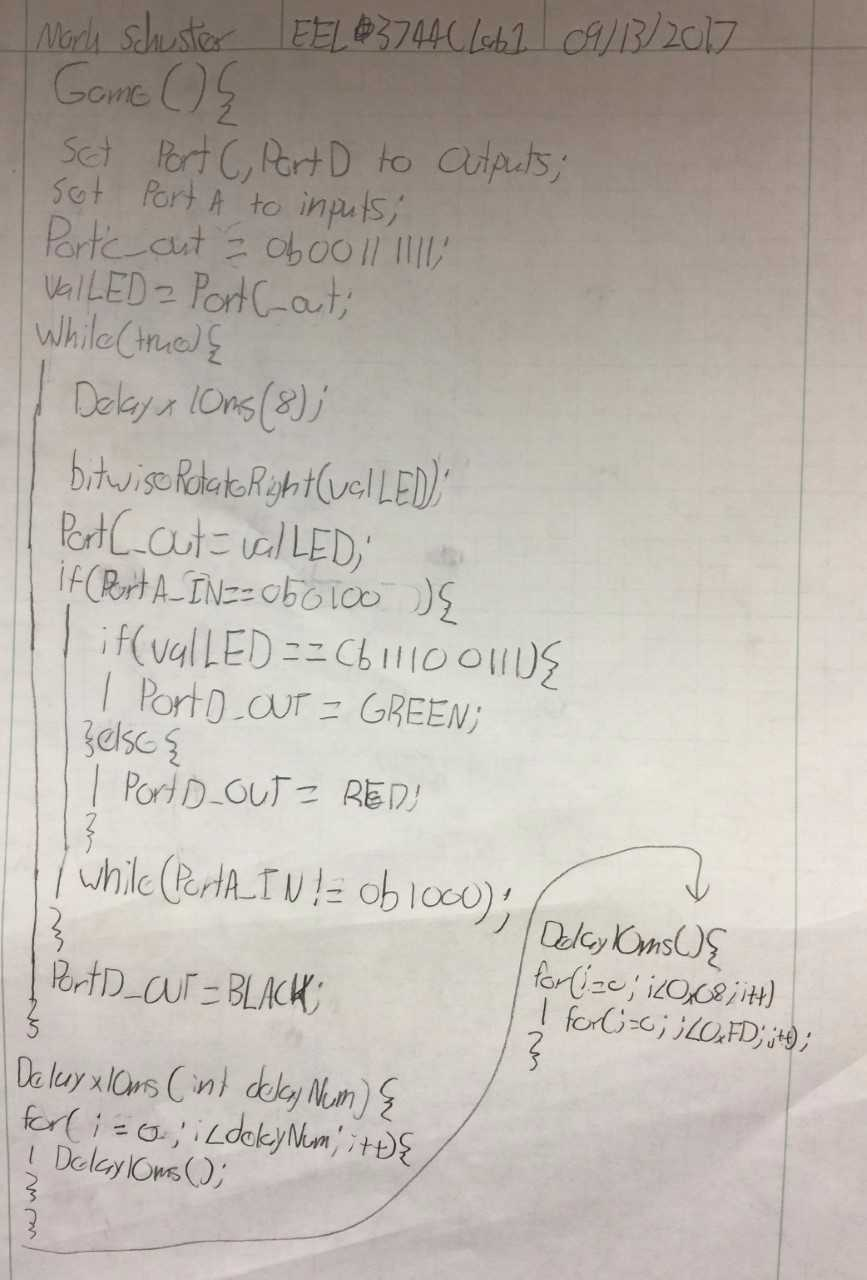
Part B pseudocode:



Part B waveform:



Part C pseudocode:



Program Code:

Part A code:

; Lab 1 part A

; Name: Mark L. Schuster

; Section #: 1540

; TA Name: Christopher Crary

; Description: This assembly application reads the current state of

; the switches on the 'SWITCH & LED BACKPACK' and turns

; on the corresponding LED.

.nolist ; Included for fun.

.include "ATxmega128A1Udef.inc" ;

.list ;

.def initVal = r16 ; Will hold the port DIRSET vlaues.

.def switchValues = r17 ; Will hold the switch's input value.

.equ ALL\_IN = 0x00 ; 8-bit vector that will set an 8-bit GPIO port to input.

.equ ALL\_OUT = 0xFF ; 8-bit vector that will set an 8-bit GPIO port to output.

.org 0x0000

rjmp init ; Start at 0x0000 and jump to program.

.org 0x200 ; Start functionality at 0x200 to avoid precious interupt vectors.

init:

ldi initVal, ALL\_IN ; Set 'initValue' to 8-bit GPIO input vector.

sts PORTA\_DIRSET, initVal ; Set PORTA to be input.

ldi initVal, ALL\_OUT ; Set 'initValue' to 8-bit GPIO output vector.

sts PORTC\_DIRSET, initVal ; Set PORTC to be output.

rjmp main

main:

lds switchValues, PORTA\_IN ; Load the switch's value into the 'switchValues'.

sts PORTC\_OUT, switchValues ; Load 'switchValues' into the output to the LEDs.

rjmp main ; Loop for continued updating.

Part B code:

; Lab 1 part B

; Name: Mark L. Schuster

; Section #: 1540

; TA Name: Christopher Crary

; Description: This assembly application causes a LED to blink

; with variable period using delay subroutines.

.nolist ; Included for fun.

.include "ATxmega128A1Udef.inc" ;

.list ;

.def initVal = r16 ; Will hold the port DIRSET vlaues.

.def ledToggle = r17 ; Will hold the value for the port output toggle.

.def countA = r18 ; Will hold the count for the outer busy loop.

.def countB = r19 ; Will hold the count for the inner busy loop.

.def delayTime = r20 ; Will hold the scaler for the number of times the 10ms delay is to occur.

.def temp = r21 ; Will hold the count for how many 10ms delays have occurred.

.equ ALL\_OUT = 0xFF ; 8-bit vector that will set an 8-bit GPIO port to output.

.equ LED\_OFF = 0xFF ; Used to initialize the LEDs to off.

.equ LED\_TOGGLE = 0x01 ; Used to toggle the right-most LED.

.equ numberOfDelays = 127

.org 0x0000

rjmp init ; Start at 0x0000 and jump to program.

.org 0x200

init:

ldi initVal, ALL\_OUT ; Store the 8-bit output vector in 'initVal'.

sts PORTC\_DIRSET, initVal ; Set portc to an output.

ldi initVal, LED\_OFF ; Store the initial LED state in 'initVal'.

sts PORTC\_OUT, initVal ; Set the LEDs to the initial value.

ldi ledToggle, LED\_TOGGLE ; Set the LED toggle value.

ldi delayTime, numberOfDelays ; Set the arbitrary number of 10ms delays.

rjmp main ;

main:

sts PORTC\_OUTTGL, ledToggle ; Toggle the LED.

rcall DELAYx10ms ; Call DELAYx10ms.

rjmp main ; Repeat.

.org 0x250

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUBROUTINES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine Name: DELAYx10ms

; Causes the device to pause operation for delayTime x 10ms.

; Inputs: delayTime

; Ouputs: None

; Affected: None

DELAYx10ms:

push delayTime ; Store the contents of delayTime.

push temp ; Store the contents of temp.

ldi temp, 0x00 ; Initialize temp to zero.

delayXLoop:

cp temp, delayTime ; Compare current count of delays and the total number of delays.

breq exitDELAYx10ms ; Exit if they are equal.

rcall DELAY\_10ms ; If not, perform a 10ms delay.

inc temp ; Increment the count.

rjmp delayXLoop ; Repeat.

exitDELAYx10ms:

pop temp ; Restore temp.

pop delayTime ; Restore delayTime.

ret ; Return.

.org 0x300

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUBROUTINES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine Name: DELAY\_10ms

; Causes the device to pause operation for 10ms.

; Inputs: None

; Ouputs: None

; Affected: None

DELAY\_10ms:

push countA ; Store the contents of countA.

push countB ; Store the contents of countB.

ldi countA, 0x00 ; Initialize countA to zero.

ldi countB, 0x00 ; Initialize countB to zero.

outerLoop:

cpi countA, 0x08 ; Compare countA with 0x08.

breq exitDELAY\_10ms ; If equal, exit.

innnerLoop:

cpi countB, 0xFD ; If not, compare countB with 0xFD.

breq incCountA ; If equal, exit the inner loop.

inc countB ; Increment countB.

rjmp innnerLoop ; Return to the nested loop.

incCountA:

ldi countB, 0x00 ; Reset countB.

inc countA ; Increment countA.

rjmp outerLoop ; return to the outer loop.

exitDELAY\_10ms:

pop countB ; Restore countB.

pop countA ; Restore countA.

ret ; Return.

Part C code:

; Lab 1 part C

; Name: Mark L. Schuster

; Section #: 1540

; TA Name: Christopher Crary

; Description: This assembly application is a game

; which is played by trying to press S2

; when LEDs 3 and 4 are lit.

.nolist ; Included for fun.

.include "ATxmega128A1Udef.inc" ;

.list ;

.def initVal = r16 ; Will hold the port DIRSET values.

.def valLED = r17 ; Will hold the LED value.

.def countA = r18 ; Will hold the count for the outer busy loop.

.def countB = r19 ; Will hold the count for the inner busy loop.

.def delayTime = r20 ; Will hold the scaler for the number of times the 10ms delay is to occur.

.def temp = r21 ; Will hold the count for how many 10ms delays have occurred.

.equ BLACK = 0xFF ; Denotes the value of black for the RGB LED.

.equ RED = 0xEF ; Denotes the value of red for the RGB LED.

.equ GREEN = 0xDF ; Denotes the value of green for the RGB LED.

.equ ALL\_IN = 0x00 ; 8-bit vector that will set an 8-bit GPIO port to input.

.equ ALL\_OUT = 0xFF ; 8-bit vector that will set an 8-bit GPIO port to output.

.equ LED\_INIT = 0x3F ; Denotes the starting configuration of PortC's LEDs.

.equ numberOfDelays = 8 ; Use to create the 80ms delay specified in the prompt.

.org 0x0000

rjmp init ; Start at 0x0000 and jump to program.

.org 0x200

init:

ldi initVal, ALL\_OUT ; Set 'initValue' to 8-bit GPIO output vector.

sts PORTC\_DIRSET, initVal ; Set PORTC to be output.

sts PORTD\_DIRSET, initVal ; Set PORTD to be output.

ldi initVal, LED\_INIT ; Store the initial LED state in 'initVal'.

sts PORTC\_OUT, initVal ; Set the LEDs to the initial value.

ldi initVal, ALL\_IN ; Set 'initValue' to 8-bit GPIO input vector.

sts PORTF\_DIRSET, initVal ; Set PORTF to be output.

ldi valLED, LED\_INIT ; Load the inital LED configuration into 'valLED'.

ldi delayTime, numberOfDelays ; Set the arbitrary number of 10ms delays.

rjmp main

main:

rcall DELAYx10ms ; Call the 80ms delay.

mov temp, valLED ; Move the LED value into 'temp' to be processed. The program

andi temp, 0x01 ; will check if the least significant bit is set to determine if

cpi temp, 0x00 ; the carry flag should be set or cleared for the impending rotation.

breq noCarry ; If the least significant bit is clear, then the carry flag will be cleared.

sec ; If not, set the carry flag.

rjmp loadLED ; Jump to pulling the LEDs current value.

noCarry:

clc ; Clear the carry flag.

loadLED:

ror valLED ; Perform a bit-wise rotation right on the LEDs' value.

sts PORTC\_OUT, valLED ; Push the rotated value to the output.

checkForS2:

lds temp, PORTF\_IN ; Pull the button values into 'temp'.

andi temp, 0x08 ; And 'temp' and 0x08 to check if S2 is being pressed.

cpi temp, 0x08 ; Compare 'temp' and 0x08.

breq loop ; If they're equal S2 is not being pressed, thus proceed.

checkIfWin:

mov temp, valLED ; Now the program will check if a win has occurred.

andi temp, 0xE7 ; This is done by checking if the LEDs are in the correct position.

cpi temp, 0xE7 ; Compare 'temp' and 0xE7 to check if it is the correct value.

breq win ; If it is, move to 'win'.

loss:

ldi temp, RED ; If not, set the RGB LED to red.

sts PORTD\_OUT, temp ; Push RED to the RGB LED.

rjmp startOver ; Jump to the end of game.

win:

ldi temp, GREEN ; Set the RGB LED to green.

sts PORTD\_OUT, temp ; Push GREEN to the RGB LED.

rjmp startOver ; Jump to the end of game.

loop:

ldi temp, BLACK ; If S2 is not pressed or the program returned from the

sts PORTD\_OUT, temp ; end of game, reset the RGB LED to off.

rjmp main ; Jump to the beginning.

startOver:

lds temp, PORTF\_IN ; Once S2 was pressed, the program waits till S1 is pressed

andi temp, 0x04 ; to reset the game.

cpi temp, 0x04 ; After pulling the button value's, compare them.

brne loop ; If S1 has been pressed, jump back to main loop.

rjmp startOver ; If not, keep checking.

.org 0x300

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUBROUTINES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine Name: DELAYx10ms

; Causes the device to pause operation for delayTime x 10ms.

; Inputs: delayTime

; Ouputs: None

; Affected: temp

DELAYx10ms:

push delayTime ; Store the contents of delayTime.

push temp ; Store the contents of temp.

ldi temp, 0x00 ; Initialize temp to zero.

delayXLoop:

cp temp, delayTime ; Compare current count of delays and the total number of delays.

breq exitDELAYx10ms ; Exit if they are equal.

rcall DELAY\_10ms ; If not, perform a 10ms delay.

inc temp ; Increment the count.

rjmp delayXLoop ; Repeat.

exitDELAYx10ms:

pop temp ; Restore temp.

pop delayTime ; Restore delayTime.

ret ; Return.

.org 0x350

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUBROUTINES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Subroutine Name: DELAY\_10ms

; Causes the device to pause operation for 10ms.

; Inputs: None

; Ouputs: None

; Affected: None

DELAY\_10ms:

push countA ; Store the contents of countA.

push countB ; Store the contents of countB.

ldi countA, 0x00 ; Initialize countA to zero.

ldi countB, 0x00 ; Initialize countB to zero.

outerLoop:

cpi countA, 0x08 ; Compare countA with 0x08.

breq exitDELAY\_10ms ; If equal, exit.

innnerLoop:

cpi countB, 0xFD ; If not, compare countB with 0xFD.

breq incCountA ; If equal, exit the inner loop.

inc countB ; Increment countB.

rjmp innnerLoop ; Return to the nested loop.

incCountA:

ldi countB, 0x00 ; Reset countB.

inc countA ; Increment countA.

rjmp outerLoop ; return to the outer loop.

exitDELAY\_10ms:

pop countB ; Restore countB.

pop countA ; Restore countA.

ret ; Return.

Appendix:

* lab1.pdf
* lab1a.asm
* lab1b.asm
* lab1c.asm