**Austin Sypolt**

**ECE 362**

**Post-Lab #7**

**Introduction:**

The purpose of this lab is to learn to use the Pulse Width Modulation (PWM) to control various elements on our board, as well as learning how to implement and debug interrupts in our programs.

**Lab 7.1:**

Objective/Purpose:

The purpose of this section was to use the PWM to control the speed of the DC motor on our 68HC12 boards.

Explanation:

The code sets all the necessary constants and variables initially. Bit 3 is set as an output so that when switch 4 (bit 3) is off no operation occurs, but when it’s on the motor can be spun using our keypad to set the period of our motor. The period of each keypad number should decrease by 4ms for each increasing number on the keypad, up to 60ms.

Results:

When the keypad buttons were pressed the motor spun appropriately, however for 1 the motor stalled as opposed to spinning properly. 0 stops the motor, and F runs the motor at the shortest period.

Code:

INCLUDE 'derivative.inc'

XDEF Entry, \_Startup, main

XREF \_\_SEG\_END\_SSTACK

constants: SECTION

portt equ $240 ;set location of port t

porttddr equ $242 ;set location of port t ddr

sequence dc.b $70, $B0, $D0, $E0 ;set the sequence

portu equ $268 ;set port u

ddru equ $26a ;set ddr of port u

psru equ $26d ;set polarity selection register of port u

pderu equ $26c ;set pull device enable register of port u

padval dc.b $0, $1, $2, $3, $4, $5, $6, $7, $8, $9, $a, $b, $C, $D, $E, $F

;set sequence of padvals

ports equ $248 ;set port s

portsd equ $24a ;set ddr of ports

keypadval dc.b $eb, $77, $7b, $7d, $b7, $bb, $bd, $d7, $db, $dd, $e7, $ed, $7e, $be, $de, $ee ;set the keypad sequence

variables: SECTION

storage1 ds.b 1 ;set a variable for keypad storage

ton ds.b 1 ;set ton variable

toff ds.b 1 ;set toff variable

tonmem ds.b 1 ;set ton memory variable

toffmem ds.b 1 ;set ton memory variable

MyCode: SECTION

main:

\_Startup:

Entry:

movb #%00001000, porttddr ;set port 3 of porttddr as output

lds #\_\_SEG\_END\_SSTACK ;set the stack

movb #$F0, ddru ;set ddr

movb #$F0, psru ;set pins 0-3 as output

movb #$0F, pderu ;activate pins 0-3

movb #$FF, portsd ;set ports as an output

movb #0, ton

movb #15, toff

reset: ldx #0 ;load 0 to x

ldaa portt ;load the value of port t to accumulator a

cmpa #0 ;compare a with 0 to see if bit not set

beq reset ;if not set, keep checking

mrloop: cpx #5 ;see if sequence is complete

beq yes ;if complete, restart

bne cont ;if incomplete, continue

cont: ldaa sequence,x ;load sequence value to a

inx ;increase x

staa portu ;store the sequence value to u

ldy #1000 ;load the value to delay by

Jsr delay ;jump to delay

ldaa portu ;load value of port u to a

staa storage1 ;save the value in storage1

anda #%00001111 ;mask the upper nibble

cmpa #$F ;compare the lower nibble to see if no button pressed

beq mrloop ;if button not pressed, run through next row

bne mas ;if button pressed, check which button

mas: JSR Keypad ;run the keypad check

bra compton ;jump to compton

yes: ldaa tonmem ;load the last value of ton

staa ton ;store it to ton

ldaa toffmem ;load the last value of toff

staa toff ;store to toff

compton:

JSR PWM ;jump to the pwm subroutine

bra reset ;restart the entire thing

delay:

rep: dey ;decrement y

bne rep ;if y isnt 0 repeat

RTS ;return to main section

Keypad: ldy #0 ;load 0 to y

ldaa storage1 ;return storage value to a

ohdaddy: cpy #$10 ;compare y to see how far through sequence

beq donzo ;if value is not in sequence, jump to end

bne yus ;if sequence not complete, continue

yus: cmpa keypadval,y

;compare the value in a with the possible

;keypad values

beq storeitplz ;if equal, store it

bne increaseit ;if not equal, increase y

increaseit: iny ;increase y

bra ohdaddy ;run it again

storeitplz: ldaa padval,y ;put new value into lower nibble

staa ton ;store to ton value

staa tonmem ;store the value of ton to memory

ldab ton ;load the on value to b

ldaa #15 ;load 15 to a

sba ;subtract b from a then store in a

staa toff ;store the value of toff

staa toffmem ;store to toff to memory

donzo: RTS ;return to main code

PWM:

here: ldaa ton ;load the t on value

cmpa #0 ;see if it equals 0

beq off ;if it does, check t off

bset portt, #$8 ;if its not 0, set bit 3 of port t

ldy #4000 ;load delay value

jsr delay ;jump to delay

deca ;decrement a

staa ton ;store to t on

bra here ;branch to here

off: ldaa toff ;load the value of t off to a

cmpa #0 ;compare a to 0 to see if its done

beq finale ;if its 0, jump to finale

bclr portt, #$8 ;clear bit 3 of port t

ldy #4000 ;load the delay value to y

jsr delay ;jump to delay

deca ;decrement a

staa toff ;store a into t off

bra off ;branch to off

finale: RTS ;jump back to main program

**Lab 7.2:**

Objective/Purpose:

The purpose of this code was to take our code from 7.1 but implement the use of real time interrupts as opposed to a manually coded 4ms delay for each incrementing keypad number.

Explanation:

This code works similar to 7.1 but instead of using our 4ms delay every time upon storing our values we enable the use of real time interrupts and delay the program appropriately from there.

Results:

The program worked as expected with no visual difference between 7.1 and 7.2 in operation, however the delay was implemented in two different methods between the two sections.

Code:

INCLUDE 'derivative.inc'

XDEF Entry, \_Startup, main, RTI\_ISR

XREF \_\_SEG\_END\_SSTACK ; symbol defined by the linker for the end of the stack

constants: SECTION

portt equ $240 ;set location of port t

porttddr equ $242 ;set location of port t ddr

sequence dc.b $70, $B0, $D0, $E0 ;set the sequence

portu equ $268 ;set port u

ddru equ $26a ;set ddr of port u

psru equ $26d ;set polarity selection register of port u

pderu equ $26c ;set pull device enable register of port u

padval dc.b $0, $1, $2, $3, $4, $5, $6, $7, $8, $9, $a, $b, $C, $D, $E, $F

;set sequence of padvals

ports equ $248 ;set port s

portsd equ $24a ;set ddr of ports

keypadval dc.b $eb, $77, $7b, $7d, $b7, $bb, $bd, $d7, $db, $dd, $e7, $ed, $7e, $be, $de, $ee ;set the keypad sequence

variables: SECTION

storage1 ds.b 1 ;set a variable for keypad storage

ton ds.b 1 ;set ton variable

toff ds.b 1 ;set tof variable

count ds.b 1 ;set the count variable

lastpress ds.b 1 ;set the lastpress variable

MyCode: SECTION

main:

\_Startup:

Entry:

movb #%00001000, porttddr ;set port 3 of porttddr as output

lds #\_\_SEG\_END\_SSTACK ;set the stack

movb #$F0, ddru ;set ddr

movb #$F0, psru ;set pins 0-3 as output

movb #$0F, pderu ;activate pins 0-3

movb #$FF, portsd ;set ports as an output

movb #0, ton ;set initial ton value

movb #15, toff ;set initial toff value

movb #0, count ;set count to 0

movb #$60, RTICTL ;set interruptt frequency

movb #%10000000, CRGFLG

movb #%10000000, CRGINT ;enable RTI

CLI ;clear the interrupt flag

reset: ldx #0 ;load 0 to x

mrloop: cpx #5 ;see if sequence is complete

beq yes ;if complete, restart

bne cont ;if incomplete, continue

cont: ldaa sequence,x ;load sequence value to a

inx ;increase x

staa portu ;store the sequence value to u

Jsr delay ;jump to delay for debounce

ldaa portu ;load value of port u to a

staa storage1 ;save the value in storage1

anda #%00001111 ;mask the upper nibble

cmpa #$F ;compare the lower nibble to see if no button pressed

beq mrloop ;if button not pressed, run through next row

bne mas ;if button pressed, check which button

mas: JSR Keypad ;run the keypad check

yes: bra reset ;jump to compton

delay: ldy #1000 ;set delay value

rep: dey ;decrement y

bne rep ;if y isnt 0 repeat

RTS ;return to main section

Keypad: ldy #0 ;load 0 to y

ldaa storage1 ;return storage value to a

ohdaddy: cpy #$10 ;compare y to see how far through sequence

beq donzo ;if value is not in sequence, jump to end

bne yus ;if sequence not complete, continue

yus: cmpa keypadval,y ;compare the value in a with the possible keypad values

beq storeitplz ;if equal, store it

bne increaseit ;if not equal, increase y

increaseit: iny ;increase y

bra ohdaddy ;run it again

storeitplz: ldaa padval,y ;put new value into lower nibble

staa lastpress ;store the value in a to lastpress

donzo: RTS ;return to main code

RTI\_ISR:

wherewithal: ldab lastpress ;load lastpress to b

stab ton ;store to ton value

ldaa count ;load the count value to a

inca ;increase a

staa count ;store a to count

cba ;compare a with b

ble shreddit1 ;branch if less than or equal to

bra off ;otherwise branch to off

shreddit1: bset portt, #$8 ;set bit 3 of port t

bra enderino ;branch to enderino

off: cmpa #$F ;compare a with F

ble shreddit2

;if less than or equal to f, branch to ;shreddit2

bra pretenderino ;otherwise branch to pretenderino

shreddit2: bclr portt, #$8 ;clear bit 3 of port t

bra enderino ;branch to enderino

pretenderino:

movb #0, count ;reset count to 0

bra wherewithal ;branch to wherewithal

enderino: bset CRGFLG, #$80 ;set the RTI flag

RTI ;return from interrupt

**Lab 7.3:**

Objective/Purpose:

The purpose of this lab is to make an operational wall clock.

Explanation:

The code of this lab was supposed to output the BCD value of an incrementing counter that increments every 1 second (1000 ms). The value should increase from 0-59 and then reset back to 0 upon surpassing 59.

Results:

Unfortunately, we were unable to complete this section of the lab on time, but our general implementation is given below. Using given code from John Lee we were able to

Code:

INCLUDE 'derivative.inc'

XDEF second

XREF \_\_SEG\_END\_SSTACK ; symbol defined by the linker for the end of the stack

movw #0,count

WALL\_CLOCK: inc count

ldx count

cpx #1000

bne exitISR

ldx #0

stx count

inc second

ldaa second

cmpa #60

bne exitISR

ldaa #0

staa second

exitISR: bset CRGFLG,#$80

RTI

INCLUDE 'derivative.inc'

XDEF second

XREF \_\_SEG\_END\_SSTACK ; symbol defined by the linker for the end of the stack

variable: SECTION

count: ds.w 1

second ds.b 1

constants: SECTION

ports equ $248

portsddr equ $24a

rtie equ $80

MyCode: SECTION

LDS #\_\_SEG\_END\_SSTACK ; initialize the stack pointer

movb #$ff,portsddr

movb #$40,RTICTL

;movb #$80,RTIENA

clr second

CLI ; enable interrupts

display: ldd second

ldx #$A

idiv

exg x,a

staa ports

bra display

**Conclusion:**

In conclusion we learned how to properly use our Pulse Width Modulation (PWM) as well as implement RTI’s into our programs to properly program and debug solutions. Furthermore, there was an inability to finish lab section 7.3 prior to the due date of this lab. This was caused by taking multiple labs to finish up previous labs as well as struggling with the first 2 sections of this lab as well. Overall our code is on the way to development however it will need to be debugged maliciously. The other 2 sections presented were executed correctly and the valuable knowledge from them has been retained.