

Stanley Vass 2306978 - ACDA4004

by Stanley Vass

Submission date: 12-Jan-2025 10:28PM (UTC+0000)

Submission ID: 248145997

File name: 51629_St Stanley_Vass_St Stanley_Vass_2306978_-_ACDA4004_621295_1029572231.docx (653.41K)

Word count: 4119

Character count: 25595

Micros & Peripherals Report

ACDA4004

Stanley Vass – 2306978

Contents

Section 1 – Circuit Explanation	3
Section 1.1 – Component Function Explanations	3
X1.....	3
C1, C2	3
R4, R3, R2.....	4
R1	4
R5.....	4
Q1.....	4
LS1	4
RN1	4
DISP1	5
U1.....	5
Section 2 – Counting Program Algorithm	5
Interpretation	5
Mapping	5
Output	6
Final Algorithm.....	6
Implementation	6
Section 3 – Produced Program Flowcharts	8
Task 1	8
Task 2.....	9
Task 3.....	10
Task 4.....	11
Task 5.....	12
Task 6.....	13
Bibliography	14
Appendix: Assembly Language Listings	15
Task 1	15
Task 2.....	18
Task 3.....	20
Task 4.....	22
Task 5.....	25
Task 6.....	28

Section 1 – Circuit Explanation

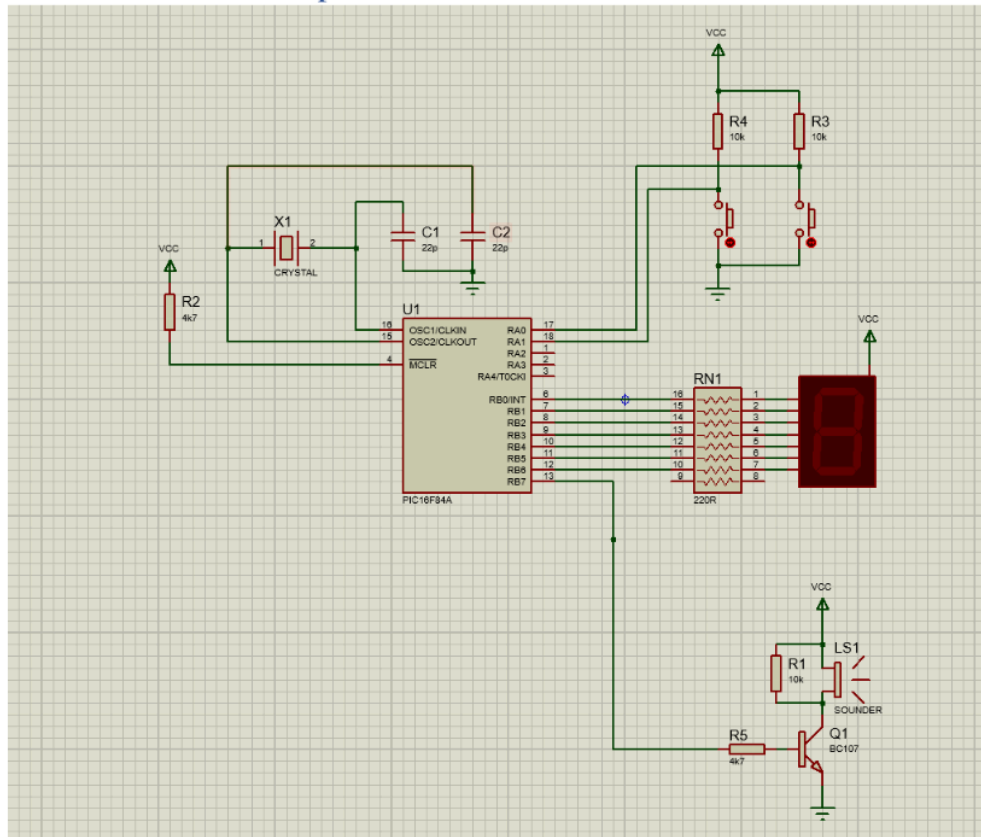


Figure 1

The circuit depicted Figure 1 is a circuit which interfaces a PIC16F84A with a 7-segment display two buttons and a buzzer, as well as providing some supporting circuitry.

Section 1.1 – Component Function Explanations

X1

The X1 component is a crystal oscillator, used to generate the clock signal for the main microcontroller.

It does this through the piezo electric effect, whereby electrical current is converted into mechanical energy, producing vibrations. These vibrations are then converted back into current, creating an oscillating current.

These vibrations can then be tuned during the manufacture of the crystal to achieve the desired frequency output.

C1, C2

C1 and C2 are capacitors which are in place to support the operation of X1 in generating the main system clock.

R4, R3, R2

R4, R3, and R2 are pullup resistors, they function to assert the undriven state of the logic lines they are connected to. As without these resistors, when undriven these lines will drift to an effectively random state, potentially causing phantom inputs.

R1

This resistor is in place to allow the piezo element to settle back to its equilibrium position.

This is needed because the piezoelectric effect works both in forward and reverse, converting electricity to mechanical strain and vice versa. [1]

This means that when Q1 stops conduction the piezo element LS1, will be trying to convert its built-up mechanical strain back into electric charge, if this charge has nowhere to go the crystal will remain deformed, and when Q1 begins conducting again, it will not move and no sound will be produced. Therefore R1 provides somewhere for the accumulated energy to dissipate and the crystal to relax back to its rest state.

R5

This is a current limiting resistor connected between the base of Q1 and an output pin of U1.

Its purpose is to limit the current flow from the Base pin of Q1 to its Emitter, the reason this is needed is that the Base-Emitter connection is effectively a diode (a PN junction). This means that as the base voltage rises above the emitter voltage current will begin to flow with very little to no resistance. And as the Emitter is tied to ground its voltage will always be zero and thus any positive voltage on the base is effectively shorted to ground.

The effects of omitting this resistor could be any number of the following:

1. Q1 fails prematurely, due to the excessive heating generated by the large current flowing across its base and emitter.
2. U1 attempts to source far more current than its output is capable of delivering and the output or entire chip fails permanently.
3. The power supply could not be capable of delivering the required current (possibly multiple amps) and could begin browning out, causing any software running on the chip to malfunction.

Q1

The transistor Q1 is used as a current amplifier, this is because the PIC microcontroller (U1) cannot drive enough current directly out of its GPIO pins to cause the sounder to sound. Therefore, Q1 is used to solve this problem by effectively amplifying the amount of current at the PIC's disposal.

It accomplishes this by acting as a current controlled gate on the sink side of the sounder (LS1), so when the PIC is outputting current into the base of Q1 through R5, Q1 will permit a larger amount of current to flow from its collector to its emitter. This provides a path for current to flow out of, and therefore, through the sounder causing it to produce sound.

LS1

This is a piezo-electric sounder, it uses the piezo-electric effect to turn electrical energy into acoustic energy.

RN1

This is a series of individual resistors which are simply contained within the same physical package for convenience.

Their purpose is to limit the current flow through the Light Emitting Diodes (LEDs) which make up the seven-segment display, this is because the “resistance” of a diode in forward conduction is typically low enough that on their own they would pass more than their maximum current rating, which would lead to the prompt, and potentially violent, destruction of the device.

DISP1

This is the display output of the circuit; its function is to display information to the user.

U1

U1 is a PIC 16F84A a member in a popular line of microcontrollers manufactured by Microchip.

Its purpose in this circuit is to implement all of the logic functions of the circuit, to translate the inputs of the buttons into outputs on the display and sounds from the buzzer.

Section 2 – Counting Program Algorithm

A subroutine for displaying the graphical digits 0 -> F would require the following logical elements:

Interpretation

The most logical method of interpreting inputs to a subroutine of the above described nature would be to interpret the binary representations of the digit values (0 – 15) as an instruction to display the appropriate digit (0 – F). This is an immediately intuitive way of doing this that should be readily understood by any even remotely experienced programmer. It also makes the display calculated numeric values extremely easy.

The alternative method would be to directly write out the required data to make the display show the desired character.

However, this approach comes with some significant downsides relative to the method described above:

1. It makes mistakes by the programmer much easier to spot and fix i.e. if intending to display the digit “d”, the programmer accidentally types 12 rather than 13 this would be immediately obvious. Whereas accidentally typing “00110111” rather than the correct “00110001” is a much harder mistake to notice at a glance.
2. It requires the programmer to have a lookup table next to them, for referencing the correct values for each character they want to display.
3. It also precludes the possibility of looping logic in a lot of case for example a counter would have to be implemented as a massive series of conditional statements.
4. Converting non pre-calculated numeric values into display output is also impossible without effectively implementing some form of binary representation to display value conversion.

Mapping

The most important and fundamental element of a subroutine translating from a data input to an equivalent data output.

mapping 0 -> ‘0’ and 1 -> ‘1’ etc.

A naïve solution to this problem could be manually typing out every single possible case using a large series of comparisons and branching operations.

While such a solution would produce a valid output, it does however carry with it a set of disadvantages:

1. Lack of expandability – If the requirements of the subroutine were to any point change. For instance, it might be required to make a base-64 equivalent function, this would create an

extreme amount of additional work for the programmer, having to manually type out an output case for all 64 possible valid inputs.

2. Processor time inefficiency – A general rule of thumb when programming is that branches are expensive operations which consume significant processor time due to the required comparison operations. For example, if the subroutine was structured to perform each comparison in order of smallest to largest, then the processor would be required to compare the input to every number smaller than 15 before finding the correct value, thus using a significant amount of processor time.

However, if the mapping were to be instead implemented using a Look Up Table (LUT), these disadvantages can be eliminated:

1. Expandability – Unlike the aforementioned branching solution a LUT is readily expandable. For instance, as with the prior example the requirements have changed after the fact and the subroutine requires altering to interpret base 64 integers. To implement this in a LUT based subroutine all that is required to do is add 48 more values onto the end of the LUT, rather than adding 48 more values, 48 more comparisons and 48 more branches.
2. Processor time efficiency – The processor time required to use a LUT is extremely little, as in an efficient system all that would be required to fetch the correct value would be to add an offset to a memory pointer and load the value from memory pointed to by the new pointer. As opposed to performing a comparison (potentially many times), moving the program counter and then fetching from memory.

Output

A subroutine such as described above has only two main ways of outputting the data, either by returning it back up the call stack or by directly outputting it to the display.

In this instance it makes far more sense to directly output it to the display, rather than pulling the value from memory, writing to some other memory, moving the Program Counter, pulling it back from memory and then outputting it to the display.

It would make far more sense to simply pull the value from memory and then output it to the display.

It could potentially make sense to push the value back up the call stack, in an instance where the calling code may want to modify the result before outputting to the display, however in this case, that is unlikely.

Final Algorithm

In conclusion the algorithm selected is one that maps integer values (0 – 15) to their respective hexadecimal graphic symbol ('0' – 'F').

It will do this by using the numeric input values as offsets for a LUT returning a precomputed value, which corresponds to the correct symbol on the display.

Then finally, it will output the retrieved value directly to the display before returning.

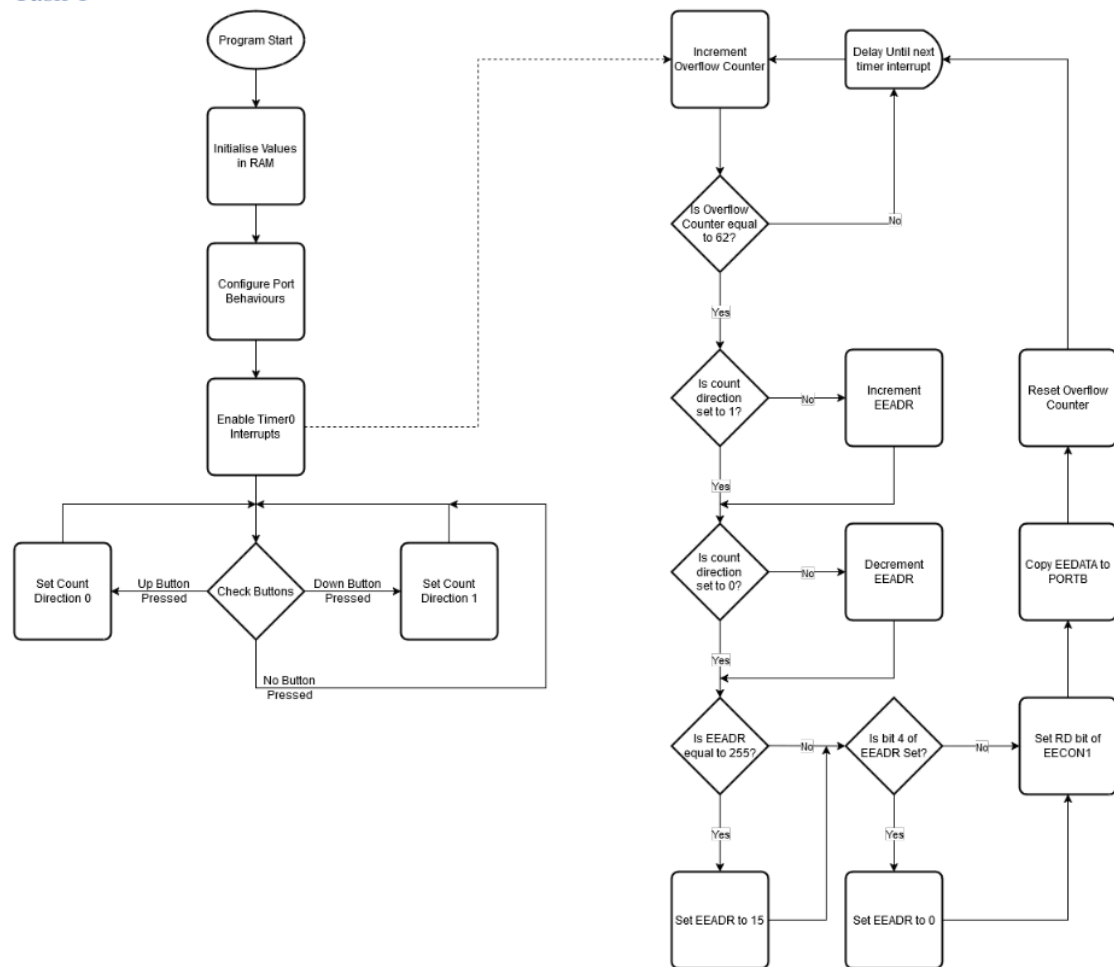
Implementation

When implementing this algorithm on a PIC16F84A I elected to write a table of precalculated display values into the EEPROM of the processor in the order 0->15 and starting at EEPROM address 0.

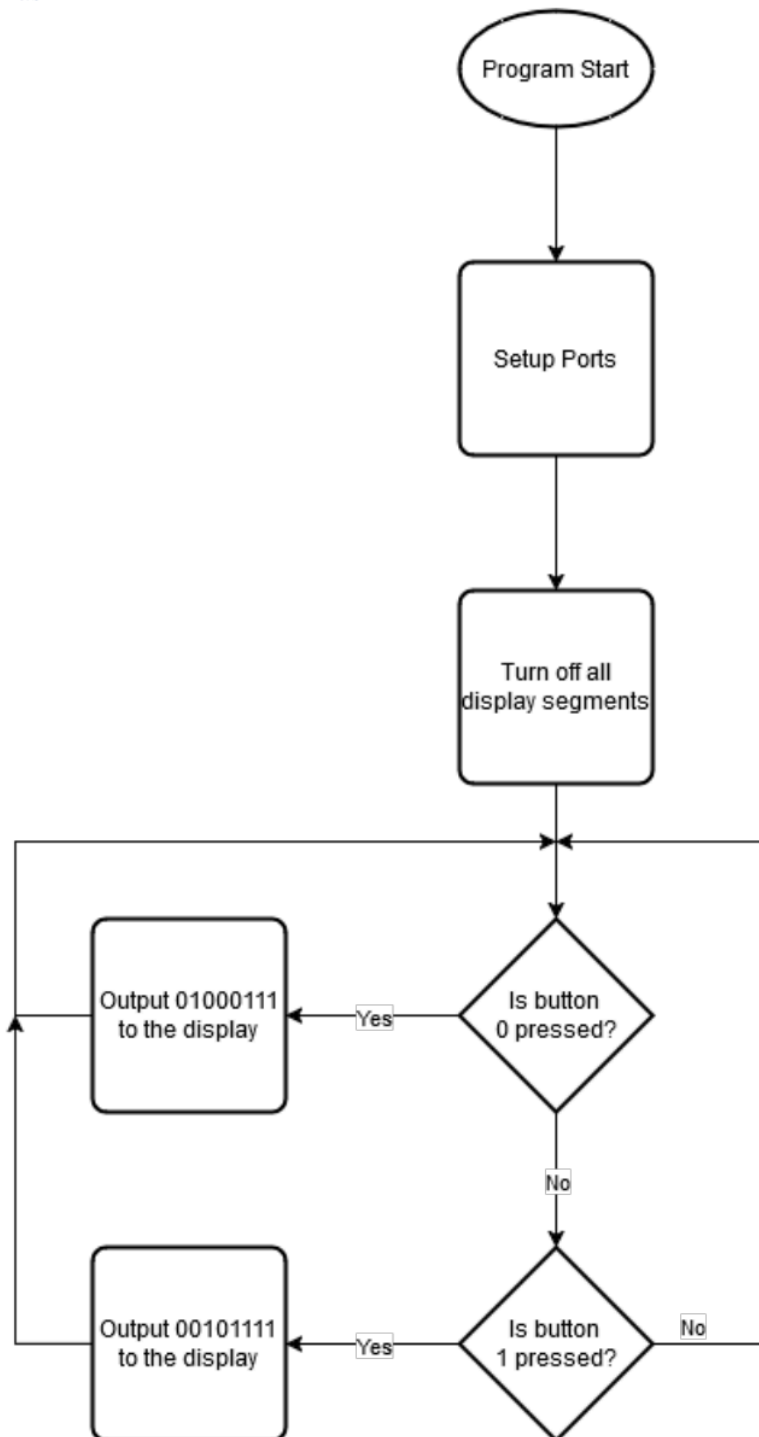
Then when a digit is to be displayed the value of the digit can be loaded into the EEADR register and the RD bit in the EECON1 register, instructing the processor to read the EEPROM.

This then results in the display data required to display the initial digit being present in the EEDATA register, which is then read into the W register, then output to the seven-segment display. An example of the implementation is present in the assembly listing appendix – Task 1.

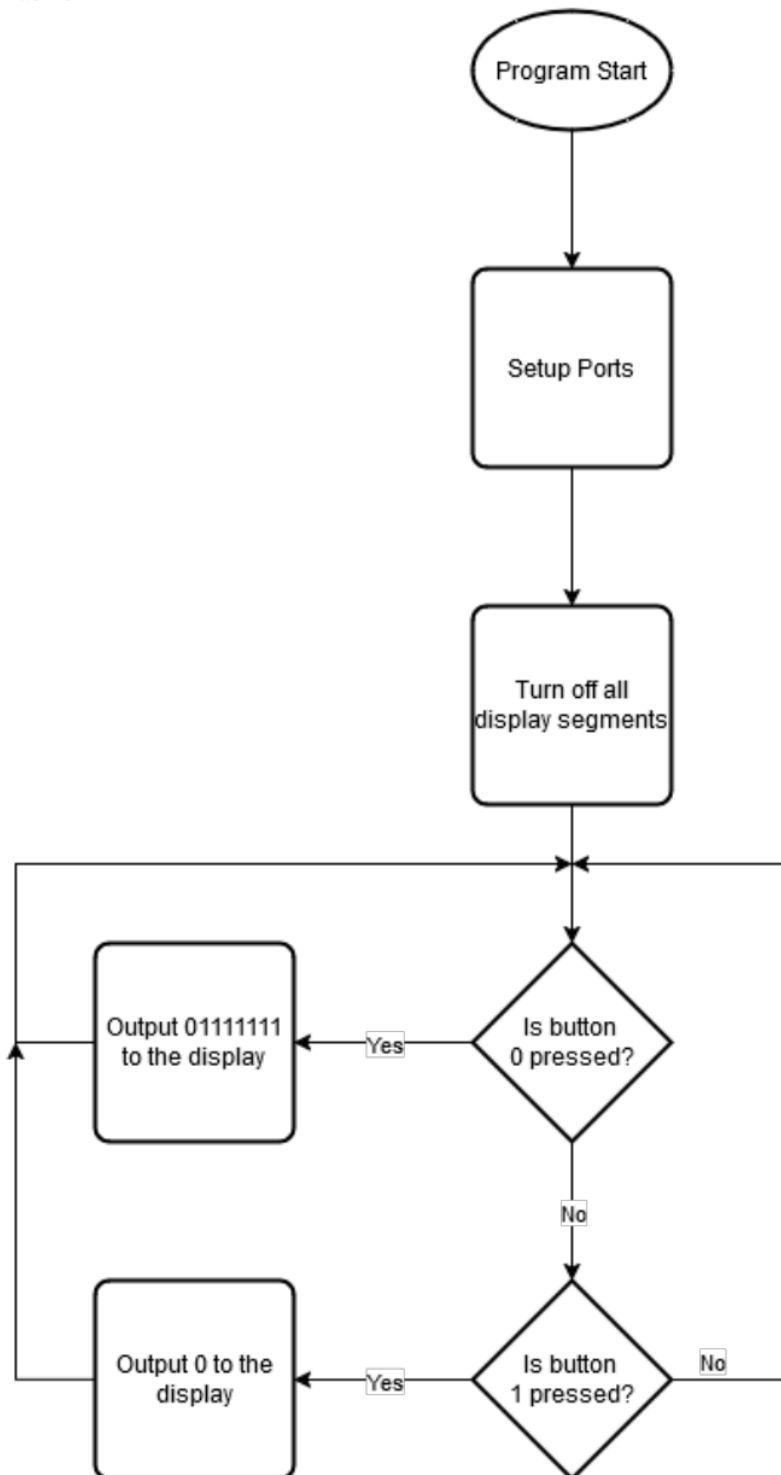
Task 1



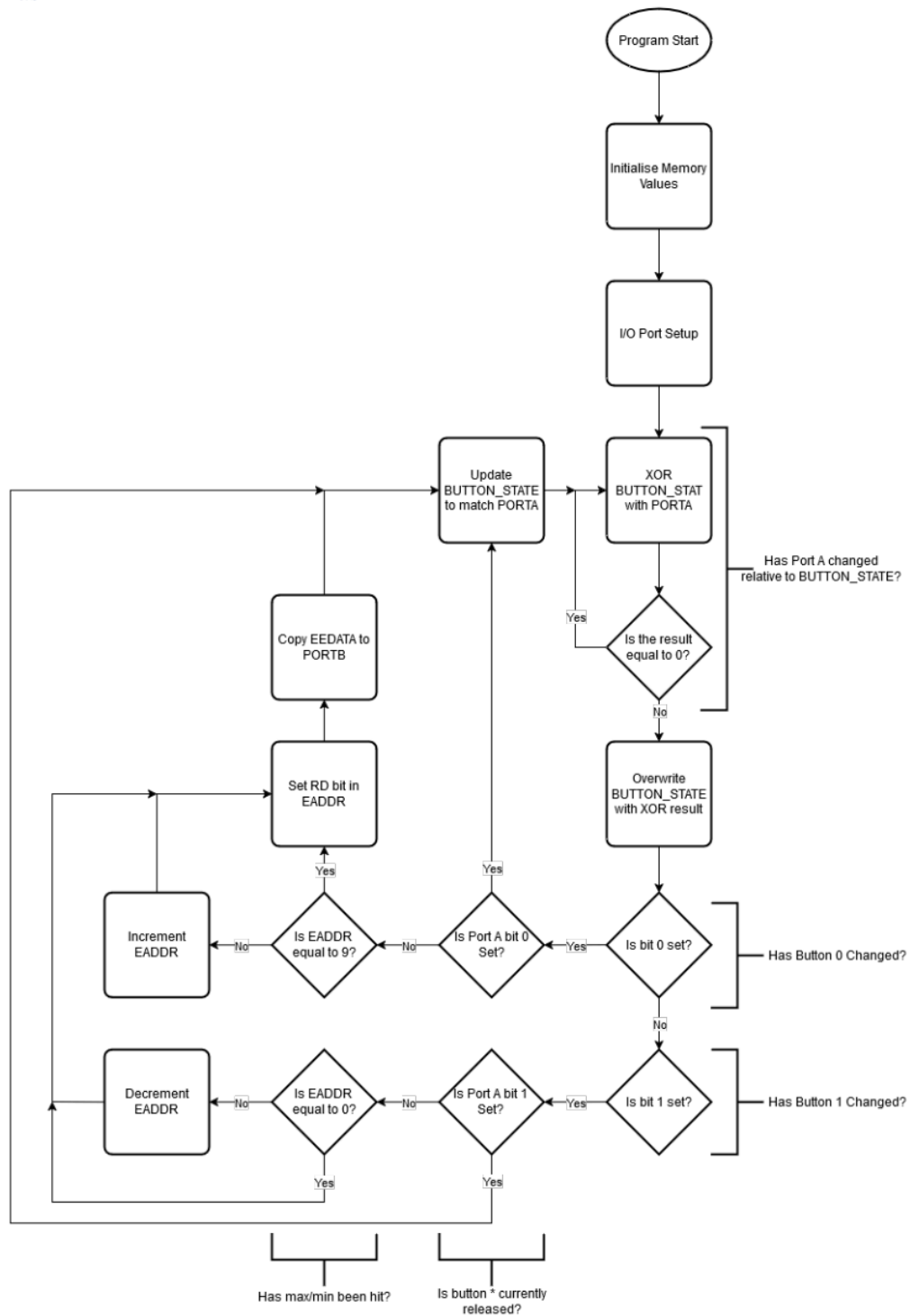
Task 2



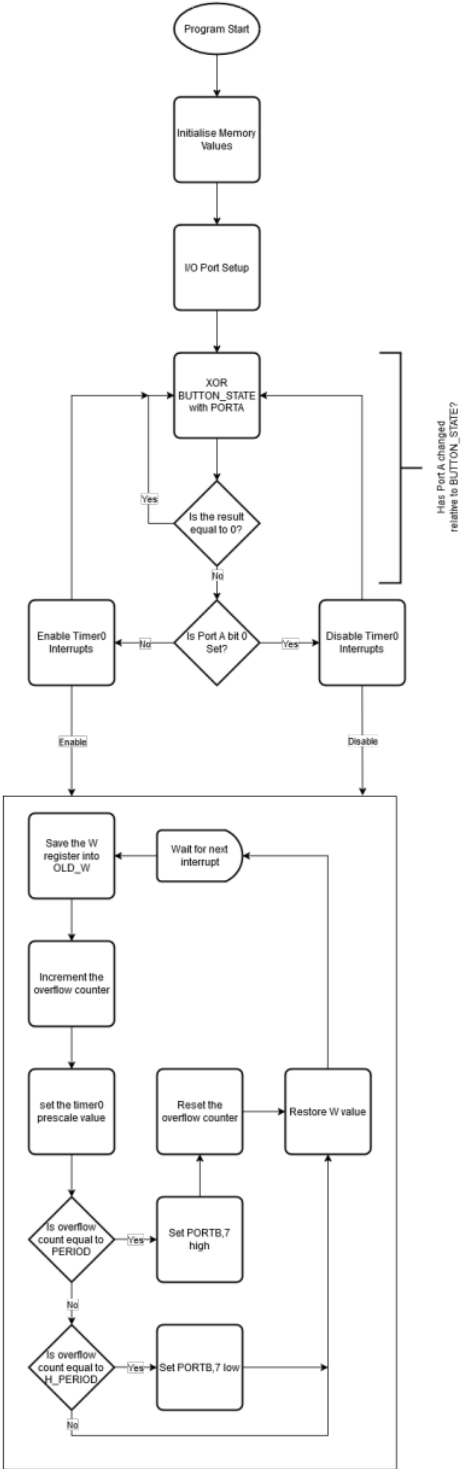
Task 3



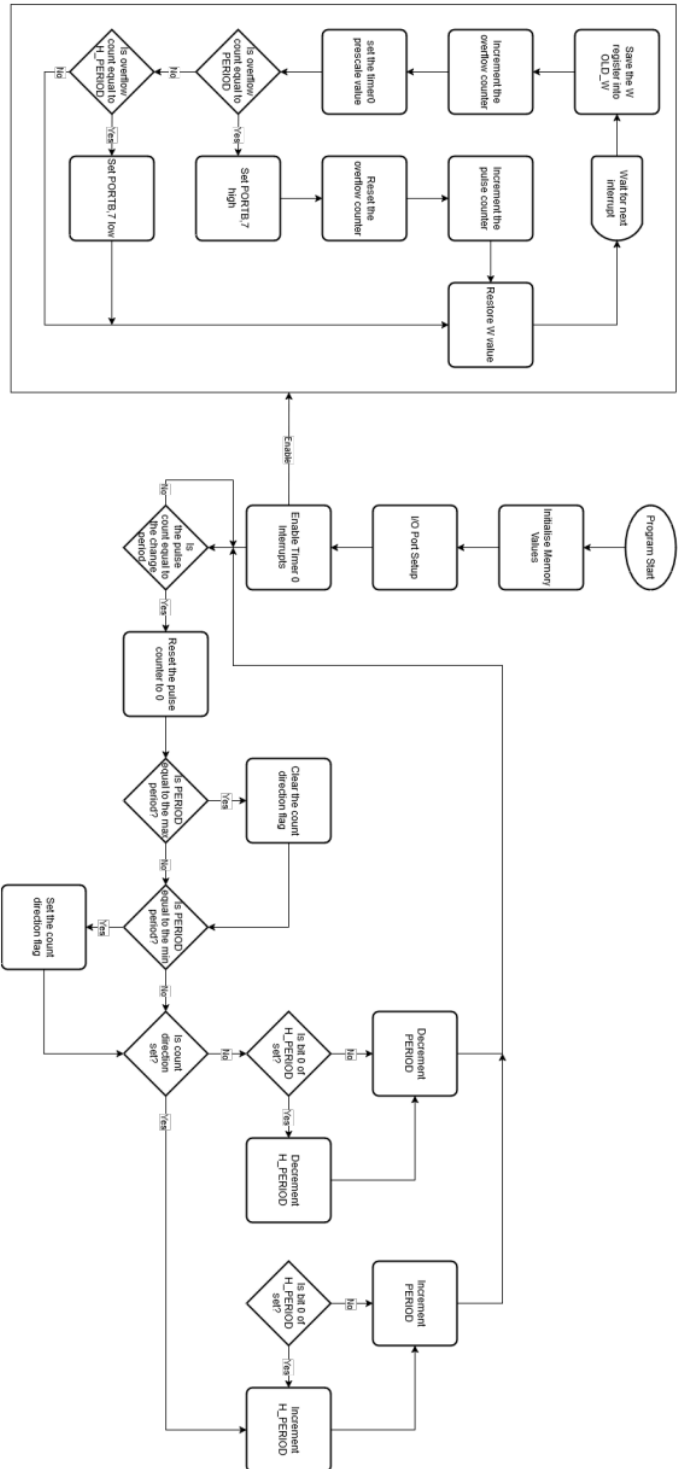
Task 4



Task 5



Task 6



Bibliography

- [1] C. Jacques and C. Pierre, "The Electrical Expansion of Quartz," *Ultrasound*, vol. 17, no. 4, pp. 196-203, 2009(1889).

Appendix: Assembly Language Listings

Task 1

```
;=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include p16f84a.inc           ; Include register definition file

;=====
;===== Data Setup =====
;=====
org      0x2100                ;Set code location to EEPROM
de 0x40, 0x4f, 0x24, 0x30      ;write the lookup table into EEPROM
de 0x19, 0x12, 0x02, 0x78
de 0x00, 0x10, 0x08, 0x03
de 0x27, 0x21, 0x06, 0x0e

OFLOW_COUNT equ 0x20          ;memory location that store the number of
timer0 wraps (allows longer timing intervals)
PRG_STAT     equ 0x21          ;memory location that stores various bitflags
about current status
_F_CNT_DIR   equ 0            ;bit offset flag that stores the current counting
direction

;=====
;===== Code Positioning in ROM =====
;=====

org 0x00
    goto Start
org 0x04
    goto ISR

Start
;=====
;===== uC Setup / Configuration =====
;=====

;Setting up start values in memory
    CLRF OFLOW_COUNT          ;start the overflow counter at 0
    CLRF EEADR                 ;start EEPROM address counter at 0
    MOVLW 0x00
    MOVWF PRG_STAT             ;set count direction down

;Setup Ports
    BSF STATUS,RP0            ;Select Bank 1
    CLRF TRISB                 ;clear TRISB register (all output)
    MOVLW 0x0f
    MOVWF TRISA                ;Set all TRISA (all input)
    BCF STATUS,RP0            ;select bank 0 again

;Setup Timer Interrupt
```

```

        BSF    STATUS,RP0
        BCF    OPTION_REG,PSA        ;clear the prescaler bit in the option
register (set to timer0 rather than WDT)
        MOVLW  b'101'
        XORWF  OPTION_REG,F          ;sets bottom three bits of OPTION_REG to
010 (1:8)

        BCF    STATUS,RP0

        CLRF   TMR0                  ;set timer0 to 0x00
        BSF    STATUS,RP0            ;set bank 1
        BCF    OPTION_REG,TOCS       ;set timer0 to internal clock
        BCF    STATUS,RP1            ;set bank back to 0

        MOVLW  b'10100000'           ;load W with value for INTCON that enables
timer interrupt and global interrupts
        MOVWF  INTCON

;=====
;===== Main Loop =====
;=====

Loop

        BTFSS  PORTA,0
        BCF    PRG_STAT,_F_CNT_DIR

        BTFSS  PORTA,1
        BSF    PRG_STAT,_F_CNT_DIR

        goto   Loop

;=====
;===== Interrupt Handler =====
;=====

ISR
        BCF    INTCON,T0IF           ;clear the timer interrupt flag
        ; Increment a counter which counts time elapsed and compare to some
timing events
        INCF   OFLOW_COUNT,F

        MOVLW  62                    ;load W with the count threshold (62 -> 200.7mS
timer)
        SUBWF  OFLOW_COUNT,W          ;will set zero flag if count == count
threshold
        BTFSC  STATUS,Z              ;if zero flag clear then skip goto increment
        GOTO   increment_lut

        RETFIE

increment_lut
        BTFSS  PRG_STAT,_F_CNT_DIR ;if count direction == 1 skip
        INCF   EEADR,F

        BTFSC  PRG_STAT,_F_CNT_DIR ;if count direction == 0 skip
        DECF   EEADR,F

        MOVLW  0xff                  ;load W 255

```

```

        SUBWF EEADR,W          ;if EEADR == 255 Z flag will be set
        BTFSC STATUS,Z        ;skip if not zero
        GOTO wrap_negative
        GOTO no_negative_wrap

wrap_negative
        MOVLW 0x0f
        MOVWF EEADR

no_negative_wrap
        BTFSC EEADR,4          ;checks if the EEADR has overrun the LUT in
EEPROM
        CLRF EEADR            ;if EEADR has overrun the LUT wrap it back to 0

        BSF STATUS,RP0        ;Select Bank 1
        BSF EECON1,RD          ;Set read Bit in EECON1
        BCF STATUS,RP0
        MOVF EEDATA,0          ;Load EEPROM data to W

        MOVWF PORTB           ; Set Port B to value fetched from LUT

        CLRF OFLOW_COUNT

        RETFIE

;=====
        END

```

Task 2

```
;=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include p16f84a.inc           ; Include register definition fil

;=====
;===== Data Setup =====
;=====

org 0x00
    GOTO Start

Start

    ;Setup Ports
    BSF    STATUS,RP0          ;Select Bank 1
    CLRF   TRISB                ;clear TRISB register (all output)
    MOVLW  0x0f
    MOVWF  TRISA                ;Set all TRISA (all input)
    BCF    STATUS,RP0          ;select bank 0 again

    MOVLW  0x7f                 ;turn all segments off initially
    MOVWF  PORTB

Loop

    BTFSS  PORTA,0              ;if porta 0 is low goto display L
    GOTO   display_l            ;I am using GOTO here rather than CALL to
                                ;encode priority and prevent flickering when both buttons are depressed
                                ;This works because at the end of display_r and
                                ;display_l they goto Loop , so display_l will prevent display_r from running

    BTFSS  PORTA,1              ;if porta 1 is low goto display r
    GOTO   display_r

    GOTO   Loop                 ;loops forever

display_r
    MOVLW  b'00101111'          ;display L
    MOVWF  PORTB

    GOTO   Loop

display_l
    MOVLW  b'01000111'          ; display r
    MOVWF  PORTB

    GOTO   Loop
```

END

Task 3

```
;=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include p16f84a.inc           ; Include register definition fil

;=====
;===== Data Setup =====
;=====

org 0x00
    GOTO Start

;Identical to Task 1, I am using the inbuilt IO latches to do the latching

Start

    ;Setup Ports
    BSF    STATUS,RP0          ;Select Bank 1
    CLRF   TRISB                ;clear TRISB register (all output)
    MOVLW  0x0f
    MOVWF  TRISA                ;Set all TRISA (all input)
    BCF    STATUS,RP0          ;select bank 0 again

    MOVLW  0x7f                ;turn all segments off initially
    MOVWF  PORTB

Loop

    BTFSS  PORTA,0              ;if porta 0 is low goto display L
    GOTO   display_on           ;I am using GOTO here rather than CALL to
                                ;encode priority and prevent flickering when both buttons are depressed
                                ;This works because at the end of display_r and
                                ;display_l they goto Loop , so display_l will prevent display_r from running

    BTFSS  PORTA,1              ;if porta 1 is low goto display r
    GOTO   display_off

    GOTO   Loop                 ;loops forever

display_off
    MOVLW  b'01111111'         ;display off
    MOVWF  PORTB

    GOTO   Loop

display_on
    MOVLW  0x00                ; display on
    MOVWF  PORTB
```

GOTO Loop

END

Task 4

```
;=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include p16f84a.inc                ; Include register definition fil

;=====
;===== Data Setup =====
;=====
org    0x2100                      ;Set code location to EEPROM
de 0x40, 0x4f, 0x24, 0x30          ;write the lookup table into EEPROM
de 0x19, 0x12, 0x02, 0x78
de 0x00, 0x10, 0x08, 0x03
de 0x27, 0x21, 0x06, 0x0e

BUTTON_STAT equ 0x20              ;pointer to value to store the previous state
of the buttons

;=====
; CODE SEGMENT
;=====
org 0x00
    goto Start

Start
;=====
;===== uC Setup / Configuration =====
;=====

;Setting up start values in memory
    MOVLW 0x12                    ;value to display 5
    MOVWF PORTB                  ;start the 7seg on the value 5
    MOVLW 0x05
    MOVWF EEADR

;Setup Ports
    BSF STATUS,RP0               ;Select Bank 1
    CLRF TRISB                   ;clear TRISB register (all output)
    MOVLW 0x0f
    MOVWF TRISA                  ;Set all TRISA (all input)
    BCF STATUS,RP0               ;select bank 0 again

;=====
;===== Main Loop =====
;=====

Loop

    MOVF BUTTON_STAT,W
    XORWF PORTA,W
```

```

        BTFSS STATUS,Z          ;if the zero flag is set than the button states
have not changed
        CALL  increment_lut

        goto  Loop

;=====
;===== Incrementing Function =====
;=====

increment_lut
        MOVWF BUTTON_STAT      ;move the result of PORTA xor BUTTON_STAT into
ram to be used in bit testing

        BTFSC BUTTON_STAT,0    ;if button 0 has changed
        GOTO  increment

        BTFSC BUTTON_STAT,1    ;if button 1 has changed
        GOTO  decrement

        RETURN

decrement
        BTFSC PORTA,1          ;if button 0 not pressed return (button
released)
        GOTO  exit_increment_lut

        MOVF  EEADR,F
        BTFSS STATUS,Z        ;if EEADR is zero skip decrement
        DECF  EEADR,F

        GOTO  update_and_exit

increment
        BTFSC PORTA,0          ;if button 1 not pressed return (button
released)
        GOTO  exit_increment_lut

        MOVLW 0x09
        SUBWF EEADR,W
        BTFSS STATUS,Z        ;if EEADR is 9 skip incrementing
        INCF  EEADR,F

        GOTO  update_and_exit

update_and_exit

        BSF   STATUS,RP0       ;Select Bank 1
        BSF   EECON1,RD        ;Set read Bit in EECON1
        BCF   STATUS,RP0
        MOVF  EEDATA,0         ;Load EEPROM data to W

        MOVWF PORTB           ; Set Port B to value fetched from LUT

exit_increment_lut

        MOVF  PORTA,W
        MOVWF BUTTON_STAT

```

```
      RETURN  
;-----  
      END
```

Task 5

```
;=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include p16f84a.inc           ; Include register definition fil

;=====
;===== Data Setup =====
;=====

; Pointers
OFLOW_COUNT equ 0x20          ;memory location that store the number of
timer0 wraps (allows longer timing intervals)
PRG_STAT     equ 0x21          ;memory location that stores various bitflags
about current status
TEMP_F       equ 0x22          ;temporary data byte register
OLD_W        equ 0x23          ;location for ISR to store W register for
restoration
PERIOD       equ 0x30          ;total PWM period
H_PERIOD     equ 0x31          ;PWM time high

; Literal Value Mnemonics
_BUTTON_STATE equ 0           ;bit offset flag that determines whether the
button update has been dealt with
T0_PERIOD     equ 0xff         ;timer 0 starting value (higher value makes
overflows occur faster)

;=====
; CODE SEGMENT
;=====
org 0x00
    goto Start
org 0x04
    goto ISR

Start
;=====
;===== uC Setup / Configuration =====
;=====

;Setting up start values in memory
    CLRF OFLOW_COUNT          ;start the overflow counter at 0
    MOVLW 0x01
    MOVWF PRG_STAT            ;set count direction down

    MOVLW 0x40
    MOVWF PERIOD              ;setting total period start value

    MOVLW 0x20
    MOVWF H_PERIOD            ;set High Period (H_PERIOD) starting value
```

```

    MOVWF H_PERIOD

;Setup Ports
    BSF    STATUS,RP0      ;Select Bank 1
    CLRF   TRISB           ;clear TRISB register (all output)
    MOVLW  0x0f
    MOVWF  TRISA           ;Set all TRISA (all input)
    BCF    STATUS,RP0      ;select bank 0 again

;Setup Timer Interrupt
;timer prescaler is not being set to get 1:1 timing for more
precision
    BCF    STATUS,RP0

    MOVLW  T0_PERIOD
    MOVWF  TMR0
    BSF    STATUS,RP0      ;set bank 1
    BCF    OPTION_REG,T0CS ;set timer0 to internal clock
    BCF    OPTION_REG,PSA   ;remove the timer prescaler from timer 0
    BCF    STATUS,RP0      ;set bank back to 0
    BSF    INTCON,7        ;enable global interrupts

;=====
;===== Main Loop =====
;=====

Loop

    MOVF   PRG_STAT,W      ;load the program status in the working register
    XORWF  PORTA,w         ;xor port a with the prg_status register and store
in the working register
    BTFSS  STATUS,Z        ;if PRG_STAT bit 0 is different from PORTA bit 0
then this bit will be high, so call process_button
    CALL   process_button
    NOP
    GOTO   Loop

;=====
;===== Interrupt Handler =====
;=====

ISR
    BCF    INTCON,T0IF     ;clear the timer interrupt flag
    MOVWF  OLD_W
    ; Increment a counter which counts time elapsed and compare to some
timing events
    INCF   OFLOW_COUNT,F

    MOVLW  T0_PERIOD
    MOVWF  TMR0

    MOVF   PERIOD,W        ;total period
    SUBWF  OFLOW_COUNT,W   ;will set zero flag if count == count
threshold
    BTFSC  STATUS,Z        ;if zero flag clear then skip goto increment
    GOTO   start_period

    MOVF   H_PERIOD,W      ;move high period into W
    SUBWF  OFLOW_COUNT,W
    BTFSC  STATUS,Z

```

```

        GOTO  end_period

        MOVF  OLD_W,W
        RETFIE

end_period

        BCF   PORTB,7           ;set pin low
        MOVF  OLD_W,W
        RETFIE

start_period

        BSF   PORTB,7           ;set pin high
        CLRF  OFLOW_COUNT       ;reset timer
        MOVF  OLD_W,W
        RETFIE

;=====
;===== Button Processing =====
;=====

process_button

        BTFSC PORTA,0           ;if port a 0 is high (button not pressed) then
disable_pwm
        GOTO  disable_pwm

        enable_pwm              ;otherwise enable PWN
        BSF   INTCON,5
        BCF   PRG_STAT, _BUTTON_STATE ;update button state to reflect the
actual button state
        RETURN

        disable_pwm
        BCF   INTCON,5
        BSF   PRG_STAT, _BUTTON_STATE ;update button state to be accurate
        RETURN

;=====
END

```

Task 6

```
;=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include p16f84a.inc                ; Include register definition fil

;=====
;===== Data Setup =====
;=====

; Pointers
OFLOW_COUNT equ 0x20                ;memory location that store the number of
timer0 wraps (allows longer timing intervals)
PRG_STAT     equ 0x21                ;memory location that stores various bitflags
about current status
TEMP_F       equ 0x22                ;temporary data byte register
OLD_W        equ 0x23                ;location for ISR to store W register for
restoration
PERIOD       equ 0x30                ;total PWM period
H_PERIOD     equ 0x31                ;PWM time high
PULSE_COUNT  equ 0x32

; Literal Value Mnemonics
_BUTTON_STATE equ 0                ;bit offset flag that determines whether the
button update has been dealt with
_COUNT_DIR    equ 1                ;bit offset determines increment or decrement pwm
frequency
TO_PERIOD     equ 0xff              ;timer 0 starting value (higher value makes
overflows occur faster)
CHANGE_PERIOD equ 0x04              ;the number of pulses between incrementing
/ decrementing the pulse frequency
MAX_PERIOD    equ 0x40              ;the highest period it will go to.
MIN_PERIOD    equ 0x02

;=====
; CODE SEGMENT
;=====

org 0x00
    goto Start
org 0x04
    goto ISR

Start
;=====
;===== uC Setup / Configuration =====
;=====

;Setting up start values in memory
    CLRF OFLOW_COUNT                ;start the overflow counter at 0
```

```

    MOVLW 0x01
    MOVWF PRG_STAT      ;set count direction down

    MOVLW 0x40
    MOVWF PERIOD        ;setting total period start value

    MOVLW 0x20          ;set High Period starting value
    MOVWF H_PERIOD

;Setup Ports
BSF STATUS,RP0        ;Select Bank 1
CLRF TRISB            ;clear TRISB register (all output)
MOVLW 0x0f
MOVWF TRISA           ;Set all TRISA (all input)
BCF STATUS,RP0        ;select bank 0 again

;Setup Timer Interrupt
;timer prescaler is not being set to get 1:1 timing for more
precision
BCF STATUS,RP0

    MOVLW T0_PERIOD
    MOVWF TMR0
    BSF STATUS,RP0      ;set bank 1
    BCF OPTION_REG,T0CS ;set timer0 to internal clock
    BCF OPTION_REG,PSA  ;remove the timer prescaler from timer 0
    BCF STATUS,RP0      ;set bank back to 0
    BSF INTCON,7        ;enable global interrupts
    BSF INTCON,5        ;enable timer 0 interrupts

;=====
;===== Main Loop =====
;=====

Loop

    ;I know that changing the period based on the number of pulses rather
    than on the time elapsed, will cause it to accelerate and decelerate.
    ;However I wanted to know what that sounds like, pretty cool as it
    turns out.
    MOVLW CHANGE_PERIOD
    SUBWF PULSE_COUNT,W
    BTFSC STATUS,Z      ;if pulse counts is equal to CHANGE_PERIOD then
change the pwm frequency
    GOTO change_params

    GOTO Loop

change_params
    CLRF PULSE_COUNT    ;reset pulse counts

    MOVLW MAX_PERIOD
    SUBWF PERIOD,W
    BTFSC STATUS,Z      ;if max_period - period is 0 change the count
direction
    BCF PRG_STAT,_COUNT_DIR

    MOVLW MIN_PERIOD
    SUBWF PERIOD,W
    BTFSC STATUS,Z
    BSF PRG_STAT,_COUNT_DIR

```

```

        BTFSC PRG_STAT,_COUNT_DIR ;if count dir flag is one increment
otherwise decrement
        GOTO  increment

        GOTO  decrement           ;this and the label are not needed, however it
is more readable this way

decrement
        BTFSC PERIOD,0           ;if PERIOD is odd decrement H_PERIOD (keeps it
at 50% duty cycle)
        DECF  H_PERIOD,F

        DECF  PERIOD,F
        GOTO  Loop

increment
        BTFSC PERIOD,0
        INCF  H_PERIOD,F

        INCF  PERIOD,F
        GOTO  Loop

;=====
;===== Interrupt Handler =====
;=====

ISR
        BCF   INTCON,T0IF       ;clear the timer interrupt flag
        MOVWF OLD_W
        ; Increment a counter which counts time elapsed and compare to some
timing events
        INCF  OFLOW_COUNT,F
        MOVLW TO_PERIOD
        MOVWF TMR0

        MOVF  PERIOD,W           ;total period
        SUBWF OFLOW_COUNT,W      ;will set zero flag if count == count
threshold
        BTFSC STATUS,Z          ;if zero flag clear then skip goto increment
        GOTO  start_period

        MOVF  H_PERIOD,W        ;move high period into W
        SUBWF OFLOW_COUNT,W
        BTFSC STATUS,Z
        GOTO  end_period

        MOVF  OLD_W,W
        RETFIE

end_period

        BCF   PORTB,7           ;set pin low
        MOVF  OLD_W,W
        RETFIE

start_period

        BSF   PORTB,7           ;set pin high
        CLRF  OFLOW_COUNT       ;reset timer

```

```
INCF PULSE_COUNT    ;increment the number of pulses output
MOVF OLD_W,W
RETFIE
```

```
;=====
END
```

FINAL GRADE

GENERAL COMMENTS

56/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

GRADING FORM: GJ_2019

STANLEY VASS

282

ENGAGEMENT WITH LITERATURE



Insufficient EWL - needs FAR more reference sources - should ideally cover all areas (components) discussed.

10

KNOWLEDGE AND UNDERSTANDING

-  The K&U demonstrated is good, but the mark is limited by the lack of reference to researched sources.

70

COGNITIVE AND INTELLECTUAL

-  Some very innovative software solutions. Also some bizarre decisions made, as task 1 for example did not ask for any buttons to be scanned, and a solution to the problem here could have been far more elegantly implemented, despite how 'clever' the final solution is. A difficult section to mark!

75

PRACTICAL APPLICATION

-  Circuit ok. Software tasks done. Explanation needed though for areas such as task6. No LED bit pattern calculations shown (minor - but it WAS asked for in the brief!)

65

TRANSFERABLE SKILLS

-  Text reads well, but more effort could have gone into the formatting to give a more polished and professional feel.

62