

James Gavin
2311749
ACDA4004

January 14, 2025

Contents

1	Introduction	6
2	Circuit	6
2.1	Voltage Regulator	6
2.2	Clock	9
2.3	Buttons	9
2.4	Buzzer	9
2.5	7-Segment Display	9
3	Displaying Hexadecimal Digits	11
4	Tasks	14
4.1	Configure PORTA and PORTB	14
4.2	Configure Timer and Interrupts	14
4.3	Handle the Timer interrupt	15
4.4	Configure the EEPROM	16
4.5	Reading from the EEPROM	16
4.6	Task 1	17
4.7	Task 2	18
4.8	Task 3	19
4.9	Task 4	20
4.10	Task 5	21
4.11	Task 6	22
5	Alternative Task Solutions	23
5.1	Task 4	23
5.2	Hardware Pulse-Width Modulation	25
5.2.1	Configuration	25
5.2.2	Enable & Disable	26
5.3	Task 5	27
5.4	Task 6	28
6	References	29
A	Assembly Code	30
A.1	Task 1	30
A.2	Task 2	33
A.3	Task 3	34
A.4	Task 4	35
A.5	Task 5	37
A.6	Task 6	39
A.7	Task 4 Version 2	41
A.8	Task 5 Version 2	44
A.9	Task 6 Version 2	47

List of Figures

1	Schematic of the Circuit	6
2	Circuit diagram of the voltage regulator	6
3	Circuit diagram of the voltage regulator connected to PIC16F84A with optional load	7
4	Simulating the voltage regulator in ngspice with and without C_o	8
5	Circuit diagram of the crystal oscillator	9
6	Circuit diagram of the buttons	9
7	Circuit diagram of the buzzer	10
8	Circuit diagram of the 7-segment display	10
9	Inputs and corresponding LED for a 7-segment display	11
10	0 on the 7-segment display	11
11	PORTB connected to the 7-segment display	11
12	0 on the 7-segment display via 0b0100000 on PORTB	12
13	Table of hexadecimal digits and their 7-segment display representation	13
14	Table of characters and their 7-segment display representation for Task 2	18
15	Table of characters and their 7-segment display representation for Task 3	19
16	Truth table for edge detection of a button press	20
17	Schematic of the Modified Circuit	25

List of Algorithms

1	Pseudo-code configuring PORTA and PORTB	14
2	Pseudo-code configuring the Timer and Interrupts	14
3	Pseudo-code configuring the EEPROM	16
4	Pseudo-code reading from the EEPROM to PORTB	16
5	Program 1 Reset Program	17
6	Program 1 Timer Interrupt	17
7	Program 2 Reset Program	18
8	Program 3 Reset Program	19
9	Program 4 Reset Program	20
10	Program 5 Reset Program	21
11	Program 5 Timer Interrupt	21
12	Program 6 Reset Program	22
13	Program 6 Timer Interrupt	22
14	Program 4 Version 2 Reset Program	23
15	Program 4 Version 2 Timer Interrupt	24
16	Pseudo-code configuring the hardware PWM	25
17	Pseudo-code enabling the hardware PWM	26
18	Program 5 Version 2 Reset Program	27
19	Program 6 Version 2 Reset Program	28
20	Program 6 Version 2 PWM Interrupt	28

Listings

1	Assembly code configuring PORTA and PORTB	14
2	Assembly code configuring the Timer and Interrupts	14
3	Assembly code handling the Timer interrupt	15
4	Assembly code configuring the EEPROM	16
5	Assembly code reading from the EEPROM to PORTB	16
6	Assembly code configuring the hardware PWM	25
7	Assembly code enabling the hardware PWM	26
8	Assembly code for Task 1	30
9	Assembly code for Task 2	33
10	Assembly code for Task 3	34
11	Assembly code for Task 4	35
12	Assembly code for Task 5	37
13	Assembly code for Task 6	39
14	Assembly code for Task 4 Version 2	41
15	Assembly code for Task 5 Version 2	44
16	Assembly code for Task 6 Version 2	47

over 6 in away [2, p. 14].

The second capacitor, C_o , is a bypass capacitor intended to reduce transient response. This means that when the micro-controller has a large increase in current requirement, for example all PORTB pins high, the output voltage of the voltage regulator drops rapidly, before shortly recovering. The capacitor dampens this voltage drop [3, p. 617].

Using ratings from the data-sheet [1, pp. 49, 51], Figure 3 approximates the largest possible current draw change for the PIC16F84A.

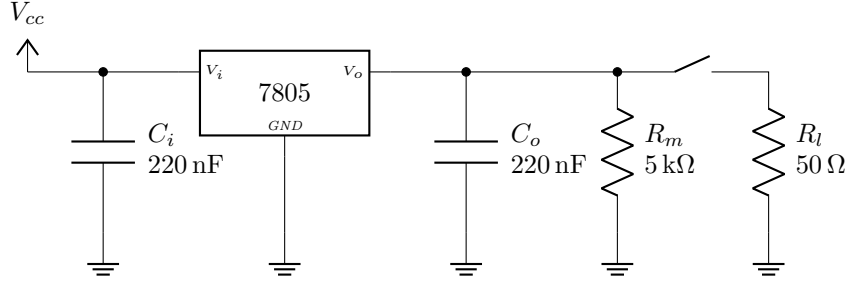


Figure 3: Circuit diagram of the voltage regulator connected to PIC16F84A with optional load

R_m represents the typical 1 mA the PIC16F84A draws during standard operation, and R_l represents a 99 mA load that will be sourced from PORTB. Figure 4 shows the output of simulating this circuit in **ngspice** [4]. The switch, representing 99 mA of collective current draw through PORTB's pins, is closed at 10 μ s.

With no capacitor, the voltage drops to 4.3 V. This is below the minimum rating of the supply voltage, when the micro-controller is configured to work with a high speed oscillator, of 4.5 V [1, p. 51]. This would most likely cause the micro-controller to reset, creating an infinite loop.

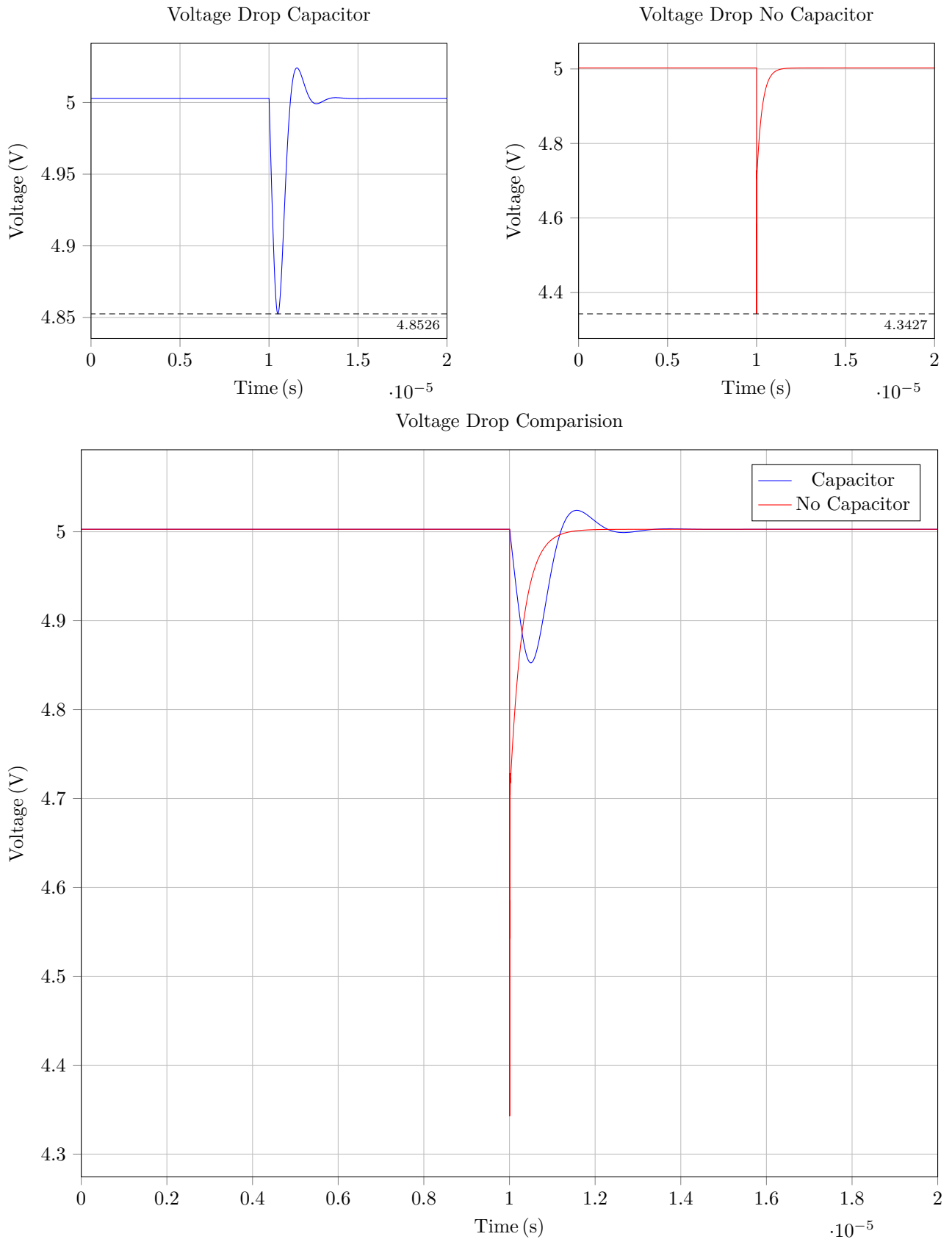


Figure 4: Simulating the voltage regulator in **ngspice** with and without C_o

2.2 Clock

The circuit uses a 4 MHz external quartz crystal oscillator to provide a clock for the micro-controller. Figure 5 shows the implementation of the crystal oscillator.

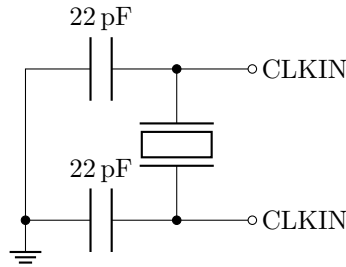


Figure 5: Circuit diagram of the crystal oscillator

The piezo-electric effect causes the crystal to generate positive voltage by applying an electric field across it [3, p. 443]. Internally, in the PIC16F84A [1, p. 22], a NOT gate inverts this output voltage, cutting the electric field and generating negative voltage. This is inverted by the NOT gate to apply an electric field, and so on. The crystal stabilises the frequency of this oscillation to that of its resonant frequency, in this case 4 MHz. The external capacitors lower the resonant frequency.

2.3 Buttons

There are two inputs the user can interface with, both of them buttons. Figure 6 shows the implementation of the buttons.

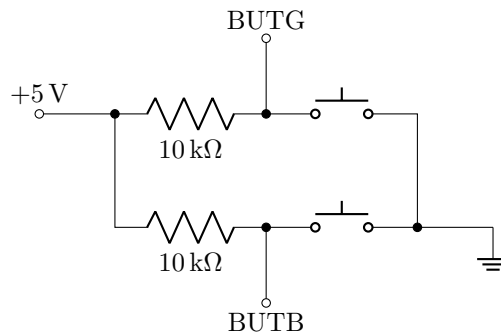


Figure 6: Circuit diagram of the buttons

BUTB and BUTG are connected to pins RA0 and RA1 respectively. Transistor-Transistor Logic (TTL) buffers are used for these pins [1, p. 4], resulting in 0 V to 0.8 V as logical low and 2 V to 5 V as logical high.

In this configuration, the resistor pulls the signal high when the button is not pressed, and shorts to ground when pressed. Without this setup, the inputs to RA0 and RA1 would be floating at an unknown voltage when the button is not pressed. The 10 kΩ resistor limits the current shorted to ground to 0.5 mA.

2.4 Buzzer

The first of two outputs is the buzzer. Figure 7 shows the implementation of the buzzer.

The buzzer has piezo-electric properties. When the input signal is high, the base of the BC107 has an input current of 0.9 mA. This gives a collector current of approximately 88 mA [5, p. 3].

Turning the input signal on and off applies an ac signal across the piezo buzzer, with the straining this causes generating sound.

2.5 7-Segment Display

The other output in this circuit is a 7-segment display. Figure 8 shows the implementation of the 7-segment display.

This 7-segment display implementation shares a common anode for powering the Light Emitting Diodes (LED). This results in logical high on the cathodes (a to g) turning the LED's off and logical low turning the LED's on.

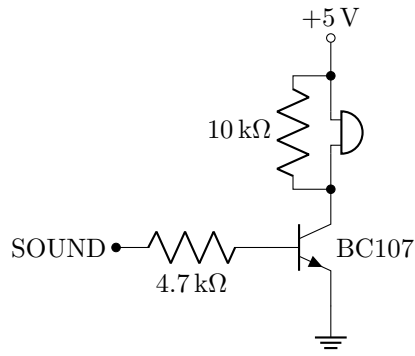


Figure 7: Circuit diagram of the buzzer

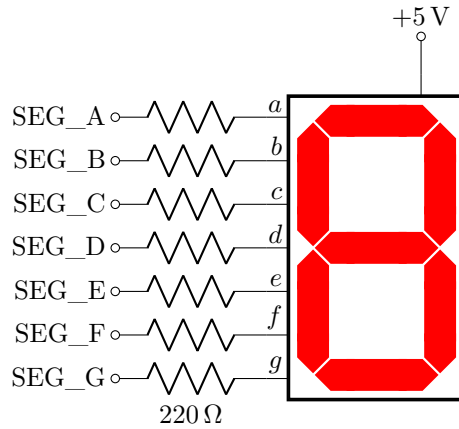


Figure 8: Circuit diagram of the 7-segment display

Using the HDSP-550G as an example of a typical red anode 7-segment display [6, p. 4], the 2.05 V forward voltage for the LED's results in 13.64 mA of current.

3 Displaying Hexadecimal Digits

The hexadecimal digits comprise of 0–9 and A–F. It is possible to show them all on a 7-segment display, by manipulating the LED's. Figure 9 shows the inputs and the corresponding LED they control.

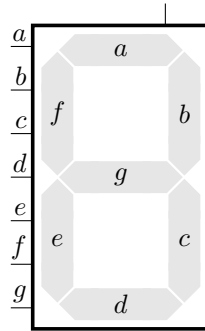


Figure 9: Inputs and corresponding LED for a 7-segment display

Since this circuit uses a 7-segment display with a common anode, logical low on an input pin turns the LED on and logical high turns the LED off. To show 0, *a*, *b*, *c*, *d*, *e*, and *f* are pulled low, and *g* is pulled high, as shown in Figure 10.

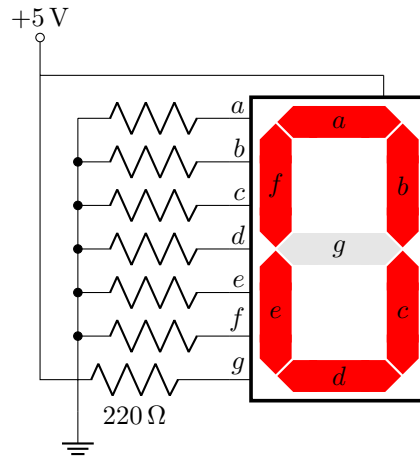


Figure 10: 0 on the 7-segment display

In the circuit, the 7-segment display is connect to PORTB pins 0–6 as shown in Figure 11.

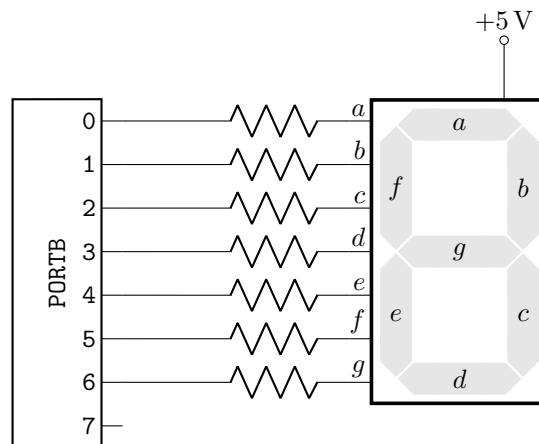


Figure 11: PORTB connected to the 7-segment display

To show 0, PORTB pins 0–5 are pulled low, and pin 6 is pulled high. PORTB needs to be manipulated to output these values. If this setup is rewritten as an 8-bit binary number, where pin 0 is the least significant bit (LSB)

and pin 7 is the most significant bit (MSB), the output becomes 01000000. The result of outputting 01000000 is shown in Figure 12.

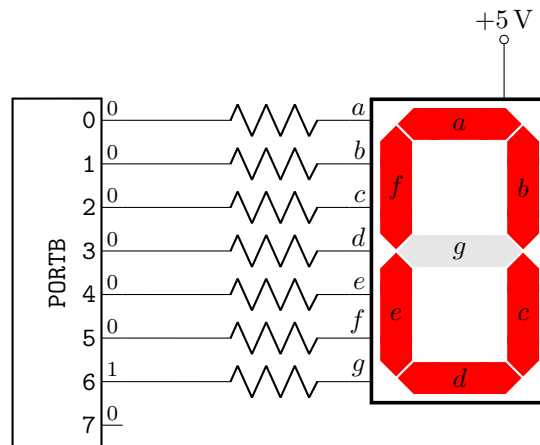


Figure 12: 0 on the 7-segment display via 0b0100000 on PORTB

Working with binary numbers when writing programs can make the code verbose and easier to introduce mistakes. Instead, hexadecimal is more common. The table of hexadecimal digits and their corresponding PORTB output value is shown in Figure 13.

Hex Digit	Binary Output	Hex Output	Display
0	01000000	0x40	
1	01111001	0x79	
2	00100100	0x24	
3	00110000	0x30	
4	00011001	0x19	
5	00010010	0x12	
6	00000010	0x02	
7	01111000	0x78	
8	00000000	0x00	
9	00010000	0x10	
A	00001000	0x08	
B	00000011	0x03	
C	01000110	0x46	
D	00100001	0x21	
E	00000110	0x06	
F	00001110	0x0E	

Figure 13: Table of hexadecimal digits and their 7-segment display representation

4 Tasks

4.1 Configure PORTA and PORTB

All the tasks, though not necessarily using them, can have the same configurations for PORTA and PORTB: all PORTA pins inputs and all PORTB pins outputs.

The manual states that upon any reset, PORTA and PORTB are configured as inputs [1, pp. 15–18]. PORTA also gets all its pins cleared to 0. This means that no additional configuration is required for PORTA, but PORTB must be configured as as output and have all its pins set low.

In the pseudo-code detailing each task, Algorithm 1 is used to represent the corresponding assembly instructions in Listing 1.

Algorithm 1 Pseudo-code configuring PORTA and PORTB

configure $PORTB_{0:7}$ $\leftarrow$ as outputs
configure $PORTA_{0:1}$ $\leftarrow$ as inputs

```
1      ; PORTA is input by default
2      ; setup PORTB
3      banksel PORTB
4      clrf     PORTB      ; clear PORTB pins
5      banksel TRISB
6      clrf     PORTB      ; set PORTB as outputs
```

Listing 1: Assembly code configuring PORTA and PORTB

4.2 Configure Timer and Interrupts

Instead of delaying by wasting time executing lots of instructions, typically loops, a timer is used instead. Not only are they more precise, but they allow, when required, the program to perform other actions in the meantime.

The timer is a counter that increments via either the instruction clock or an external clock (RA4). A prescaler can require multiple clock pulses per counter increment, allowing the timer to have a longer period. When the counter overflows (increments from 255 to 0), the TOIF flag of INT_CON (bit 2) is set high. This can be used, which is done in the tasks, to trigger an interrupt to handle this overflow.

To enable timer mode (increment with the instruction clock), the TOCS flag of OPTION_REG (bit 5) must be cleared. The prescaler allows the timer clock to be incremented at a binary multiple slower than the instruction clock. To enable the prescaler for the timer, the PSA flag of OPTION_REG (bit 3) must be cleared. The prescaler can be selected by modifying the PS2:PS0 bits of OPTION_REG (bits 2–0). To enable the timer overflow interrupt, the TOIE and GIE flags of INT_CON (bits 5 and 7 respectively) need to be set.

In the circuit, the oscillator runs at 4 MHz, resulting in an instruction clock of 1 MHz. The most useful prescaler here is 1:4, which gives an interrupt frequency of 980 Hz, or about once a 1 ms.

In the pseudo-code tasks using the timer, Algorithm 2 is used to represent the corresponding assembly instructions in Listing 2.

Algorithm 2 Pseudo-code configuring the Timer and Interrupts

Enable 1 ms Timer Interrupt

```
1      ; set TMRO clock to internal, and set prescaler to 1:4
2      banksel OPTION_REG
3      movf     OPTION_REG, w
4      andlw    0xc0      ; mask non-TMRO settings
5      iorlw    0x01      ; set TMRO settings
6      movlw    OPTION_REG
7
8      ; setup interrupts
9      bcf      INTCON, RBIF
10     bsf      INTCON, TOIE
11     bsf      INTCON, GIE
```

Listing 2: Assembly code configuring the Timer and Interrupts

4.3 Handle the Timer interrupt

When the timer counter overflows the Interrupt Service Routine (ISR) will be called.

First the interrupt needs to save the working (W) and STATUS registers. This is because the ISR uses the same registers as the normal program, and leaving these registers in a bad state upon leaving the ISR will have unintended, potentially disastrous, side effects on the normal program.

Next, the reason for the ISR being called needs to be established. All interrupts call the same ISR, and which one caused the call can be determined by checking the corresponding flag in INT_CON. A jump can then be made to dedicated handlers for each type of interrupt.

The first thing the handler should do is clear its corresponding flag. If this isn't done, then no more interrupts can be caused by that flag in the future.

After the handler has finished, the W and STATUS registers need to be restored to the values saved earlier. Finally, the instruction `retfie` ends the ISR and the normal program continues where it was paused.

Currently, the interrupt frequency is 980 Hz. If the counter could overflow after 250 increments, instead of 255, then the interrupt frequency would become exactly 1 kHz.

It is possible to change the value in the timer's counter, by modifying TMRO. By setting TMRO to 6, the initial increments will be skipped, giving a 1 kHz interrupt frequency. However, when debugging the program, the time between interrupts is larger than 1 ms.

This is because, the counter never stopped when the ISR was called, so when it's modified to contain 6, it already contained 3. Instead, at a precise instruction dependent on the prescaler (in this case 1:4), 10 (hex 0x0a) should be written.

There is no pseudo-code equivalent for Listing 3, but all the tasks that use the timer interrupt require this assembly code.

```
1 handler:
2     ; store Working and STATUS register
3     movwf    w_temp
4     swapf    STATUS, w
5 ; tmr0 is 2
6     movwf    s_temp
7
8     ; check if TMRO flag high
9     btfss    INTCON, TOIF
10    goto     handler_end
11
12 tmr0_handler:
13     ; clear the TMRO flag
14     bcf      INTCON, TOIF
15
16 ; tmr0 is 3
17     ; TMRO interrupt caused by overflow
18     ; 4 * 250 = 1000 (1ms)
19     ; so offset TMRO by 6
20     ; however TMRO has been running in background
21     ; so put 10 into TMRO at the correct instruction
22     movlw    0x0a
23     banksel  TMRO
24     nop
25     nop
26 ; tmr0 is 4
27     movwf    TMRO
28
29
30     ; do task specific stuff here
31
32
33 handler_end:
34     ; restore Working and STATUS register
35     swapf    s_temp, w
36     movwf    STATUS
37     swapf    w_temp, f
38     swapf    w_temp, w
39
40     retfie
```

4.4 Configure the EEPROM

The EEPROM is used to store the hexadecimal digits that are displayed on the 7-segment display. This is due to the fact that data cannot be stored in the flash program memory, as there are no instructions that can read it. The hexadecimal digits are stored in an array with a label marking the beginning and the end.

Unlike other data, the directive `de` is used specifically for EEPROM data. Additionally, the address `0x2100` is the location of the first byte on the EEPROM [7, p. 96].

In the pseudo-code tasks using hexadecimal digits for the 7-segment display, Algorithm 3 is used to represent the corresponding assembly instructions in Listing 4.

Algorithm 3 Pseudo-code configuring the EEPROM

$hex_digits \leftarrow ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F']$

```

1  eeprom          code    0x2100
2  hex_7seg:      de 0x40
3                de 0x79
4                de 0x24
5                de 0x30
6                de 0x19
7                de 0x12
8                de 0x02
9                de 0x78
10               de 0x00
11               de 0x10
12               de 0x08
13               de 0x03
14               de 0x46
15               de 0x21
16               de 0x06
17               de 0x0e
18  hex_7seg_end:

```

Listing 4: Assembly code configuring the EEPROM

4.5 Reading from the EEPROM

Reading data from the EEPROM requires indirect-addressing, making it ideal to store the hexadecimal digits array. The address to read on the EEPROM is stored in `EEADR`. A read is initiated by setting the `RD` flag of `EECON1` (bit 0). The data from the EEPROM at `EEADR` is now stored in `EEDATA`, where it can be retrieved.

In the pseudo-code tasks using hexadecimal digits for the 7-segment display, Algorithm 4 is used to represent the corresponding assembly instructions in Listing 5 (hex_cntr is `EEADR`).

Algorithm 4 Pseudo-code reading from the EEPROM to PORTB

$PORTB_{0:7} \leftarrow hex_digits[hex_cntr]$

```

1  ; EEADR is modified elsewhere
2
3      ; get the next hex digit
4      banksel EECON1
5      bsf     EECON1, RD    ; read from the EEPROM into EEDATA
6      banksel EEDATA
7      movf    EEDATA, w
8
9      ; display the next hex digit
10     banksel PORTB
11     movwf    PORTB

```

Listing 5: Assembly code reading from the EEPROM to PORTB

4.6 Task 1

Task 1 uses the 7-segment display to count from 0 to F with a 200 ms delay in between.

To achieve this, the timer interrupt is configured to fire every 1 ms. A counter can then be incremented until it reaches 200, at which point 200 ms has passed. Then a flag is set and the counter reset to 0. The main loop waits for this flag, after which the next hex digit is shown on the 7-segment display. If the end of the hex digits is reached, reset back to the start.

The reset program configures `PORTB`, enables the 1 ms timer interrupt, and sets the initial 7-segment display digit to 5 (via `EEADR`). A loop is entered, waiting for 200 ms to pass (`delay_flag` set).

When `delay_flag` is set by the timer interrupt, it gets cleared here. Then, `hex_cntr` is incremented. If `hex_cntr` is 16, then it is set to 0. Finally, the corresponding `hex_digits` value for `hex_cntr` is outputted on `PORTB` for the 7-segment display. The pseudo-code for the reset program is in Algorithm 5.

Algorithm 5 Program 1 Reset Program

```
hex_digits ← ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F']
configure PORTB0:7 ← as outputs
tmr_cnt ← 0
EEADR ← 0
Enable 1 ms Timer Interrupt
loop
    hex_cntr ← 0
    while hex_cntr ≠ 16 do
        while delay_flag = 0 do
            end while
        delay_flag ← 0
        PORTB0:7 ← hex_digits[hex_cntr]
        hex_cntr ← counter + 1
    end while
end loop
```

Every 1 ms, the timer interrupt is fired. `tmr_cnt` is incremented. If `tmr_cnt` is 200 ms, then `delay_flag` is set, and `tmr_cnt` reset to 0 ms. The pseudo-code for the timer interrupt is in Algorithm 6.

Algorithm 6 Program 1 Timer Interrupt

```
tmr_cnt ← tmr_cnt + 1
if tmr_cnt = 200 then
    tmr_cnt ← 0
    delay_flag ← 1
end if
```

Listing 8 shows the assembly code implementing the pseudo-code in Algorithm 5 and Algorithm 6.

4.7 Task 2

Task 2 uses the 7-segment display to show 'L' if BUTB is pressed, 'r' if BUTG is pressed, or ' ' if neither are pressed. The pseudo-code for the reset program is in Algorithm 7.

Algorithm 7 Program 2 Reset Program

```

configure  $PORTB_{0:7}$   $\leftarrow$  as outputs
configure  $PORTA_{0:1}$   $\leftarrow$  as inputs
loop
     $output \leftarrow ' '$ 
    if  $PORTA_0 = 0$  then
         $output \leftarrow 'L'$ 
    end if
    if  $PORTA_1 = 0$  then
         $output \leftarrow 'r'$ 
    end if
     $PORTB_{0:7} \leftarrow output$ 
end loop

```

Figure 14 shows ' ', 'L', and 'r' converted for output on PORTB.

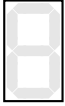
Character	Binary Output	Hex Output	Display
			
L	01111111	0x7F	
r	01000111	0x47	

Figure 14: Table of characters and their 7-segment display representation for Task 2

Listing 9 shows the assembly code implementing the pseudo-code in Algorithm 7.

4.8 Task 3

Task 3 uses the 7-segment display to show '8' if BUTB is pressed, ' ' if BUTG is pressed, or continue showing the previous value if neither are pressed. The pseudo-code for the reset program is in Algorithm 8.

Algorithm 8 Program 3 Reset Program

```

configure  $PORTB_{0:7} \leftarrow$  as outputs
configure  $PORTA_{0:1} \leftarrow$  as inputs
 $output \leftarrow$  ' '
loop
  if  $PORTA_0 = 0$  then
     $output \leftarrow$  '8'
  end if
  if  $PORTA_1 = 0$  then
     $output \leftarrow$  ' '
  end if
   $PORTB_{0:7} \leftarrow output$ 
end loop

```

Figure 15 shows ' ', '8' converted for output on PORTB.

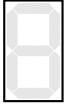
Character	Binary Output	Hex Output	Display
	01111111	0x7F	
8	00000000	0x00	

Figure 15: Table of characters and their 7-segment display representation for Task 3

Listing 10 shows the assembly code implementing the pseudo-code in Algorithm 8.

4.9 Task 4

Task 4 uses the 7-segment display to show a counter. This counter can be incremented by pressing BUTB or decremented by pressing BUTG.

The reset program configures PORTA and PORTB and sets the initial 7-segment display digit to 5 (via `hex_index`). A loop is entered, waiting for BUTB or BUTG to be pressed.

If BUTB is pressed, then `hex_index` is incremented (if possible). If BUTG is pressed, then `hex_index` is decremented (if possible). Finally, the corresponding `hex_digits` value for `hex_index` is outputted on PORTB for the 7-segment display. The pseudo-code for the reset program is in Algorithm 9.

Algorithm 9 Program 4 Reset Program

```

hex_digits ← ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
configure PORTB0:7 ← as outputs
configure PORTA0:1 ← as inputs
hex_index ← 5
loop
  if PORTA0 goes low then
    if hex_index ≠ 9 then
      hex_index ← hex_index + 1
    end if
  end if
  if PORTA1 goes low then
    if hex_index ≠ 0 then
      hex_index ← hex_index - 1
    end if
  end if
  PORTB0:7 ← hex_digits[hex_index]
end loop

```

So that only one increment or decrement is performed when a button is pressed, edge detection is required. To detect when a button is pressed, the input will go from logical high to logical low. Figure 16 shows the truth table for the logic that detects if a button has just been pressed. B and B^{-1} are the current and previous state of the button respectively. By using this method of edge detection, both buttons can be checked in parallel.

B^{-1}	B	$(B \vee B^{-1}) \wedge B^{-1}$
0	0	0
1	0	1
0	1	0
1	1	0

Figure 16: Truth table for edge detection of a button press

Listing 11 shows the assembly code implementing the pseudo-code in Algorithm 9.

4.10 Task 5

Task 5 outputs a tone on the buzzer when BUTB is pressed.

The reset program configures PORTB. Once the timer is enabled, the signal is handled by the timer interrupt, so an empty loop is entered. The pseudo-code for the reset program is in Algorithm 10.

Algorithm 10 Program 5 Reset Program

```
configure  $PORTB_7 \leftarrow$  as output  
Enable 0.5 ms Timer Interrupt  
loop  
end loop
```

When the timer interrupt is fired, the output sent to the buzzer is toggled if BUTB is pressed. The pseudo-code for the reset program is in Algorithm 11.

Algorithm 11 Program 5 Timer Interrupt

```
if  $PORTA_0 = 0$  then  
    toggle  $PORTB_7$   
end if
```

Listing 12 shows the assembly code implementing the pseudo-code in Algorithm 10 and Algorithm 11.

4.11 Task 6

Task 6 outputs a fixed frequency pulse train on the buzzer. The duty cycle of the signal loops from 0 % to 50 %, and then back to 0 %.

The reset program configures `PORTB`, clears `duty_cnt`, and enables the timer and timer interrupt. Once the timer is enabled, the pulse train is handled by the timer interrupt, so an empty loop is entered. The pseudo-code for the reset program is in Algorithm 12.

Algorithm 12 Program 6 Reset Program

```
duty_cnt ← 0
duty_add ← 1
configure PORTB7 ← as output
Enable Timer Interrupt
loop
end loop
```

`duty_cnt` is used to track the current duty cycle. The timer delays for $255 - \text{duty_cnt}$ with the output low, then for `duty_cnt` with the output high. `duty_cnt` can then be incremented/decrement between 0 and 128 as required to generate the pulse train. The pseudo-code for the timer interrupt is in Algorithm 13.

Algorithm 13 Program 6 Timer Interrupt

```
toggle PORTB7
if PORTB7 = 1 then
    TMR0 ← 255 − duty_cnt
else
    duty_cnt ← duty_cnt + duty_add
    if duty_cnt = 0 then
        duty_add ← 1
    end if
    if duty_cnt = 128 then
        duty_add ← −1
    end if
    TMR0 ← duty_cnt
end if
```

Listing 13 shows the assembly code implementing the pseudo-code in Algorithm 12 and Algorithm 13.

5 Alternative Task Solutions

5.1 Task 4

This version of Task 4 allows the user to hold the button down to perform multiple increments and decrements with one button press. When the button is first pressed, the counter is incremented/decremented, and a 400 ms delay starts. At the end of the delay, if the button is still pressed down the counter increments/decrements at 100 ms intervals until the button is released.

The reset program configures PORTA and PORTB, clears `but1_cnt` and `but2_cnt`, enables the 2 ms timer interrupt, and sets the initial 7-segment display digit to 5 (via `digits_index`). A loop is entered, waiting for BUTB or BUTG to be pressed.

If BUTB is pressed, then `digits_index` is incremented (if possible), and `but1_cnt` is set to 200. Otherwise, `but1_cnt` is set to 0. If BUTG is pressed, then `digits_index` is decremented (if possible), and `but2_cnt` is set to 200. Otherwise, `but2_cnt` is set to 0. Finally, the corresponding `digits` value for `digits_index` is outputted on PORTB for the 7-segment display. The pseudo-code for the reset program is in Algorithm 14.

Algorithm 14 Program 4 Version 2 Reset Program

```
digits  $\leftarrow$  ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
configure PORTB0:7  $\leftarrow$  as outputs
configure PORTA0:1  $\leftarrow$  as inputs
digits_index  $\leftarrow$  5
but1_cnt  $\leftarrow$  0
but2_cnt  $\leftarrow$  0
Enable 2 ms Timer Interrupt
loop
  if PORTA0 goes low then
    if digits_index  $\neq$  9 then
      digits_index  $\leftarrow$  digits_index + 1
    end if
    but1_cnt  $\leftarrow$  200
  else
    but1_cnt  $\leftarrow$  0
  end if
  if PORTA1 goes low then
    if digits_index  $\neq$  0 then
      digits_index  $\leftarrow$  digits_index - 1
    end if
    but2_cnt  $\leftarrow$  200
  else
    but2_cnt  $\leftarrow$  0
  end if
  PORTB0:7  $\leftarrow$  digits[digits_index]
end loop
```

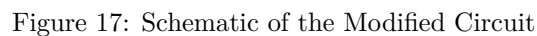
Every 2 ms, the timer interrupt is fired. `but1_cnt` is decremented. If `but1_cnt` is 0, then `digits_index` is incremented (if possible), and `but1_cnt` is set to 50. `but2_cnt` is decremented. If `but2_cnt` is 0, then `digits_index` is decremented (if possible), and `but2_cnt` is set to 50. The pseudo-code for the timer interrupt is in Algorithm 15.

Listing 14 shows the assembly code implementing the pseudo-code in Algorithm 14 and Algorithm 15.

Algorithm 15 Program 4 Version 2 Timer Interrupt

```
but1_cnt ← but1_cnt − 1
if but1_cnt = 0 then
  if digits_index ≠ 9 then
    digits_index ← digits_index + 1
  end if
  but1_cnt ← 50
end if
but2_cnt ← but2_cnt − 1
if but2_cnt = 0 then
  if digits_index ≠ 0 then
    digits_index ← digits_index − 1
  end if
  but2_cnt ← 50
end if
```

Tasks 4 and 5 are similar to their originals, but they use hardware pulse-width modulation (PWM). The PIC16F84A does not have hardware PWM, so the PIC16882 is used instead [8]. Figure 17 shows the modified circuit used for Tasks 5 and 6.



The **Capture/Control/PWM** module (**CCP1**) is used for the PWM. **TIMER2** controls the frequency of the PWM, and the duty cycle is controlled by **CCPR1L** (0 is 0 %, 255 is 100 %). An interrupt gets called each time the PWM pulse is finished transmitting. The complete setup for PWM and **TIMER2** can be found in the manuals [8, p. 129].

Algorithm 16 Pseudo-code configuring the hardware PWM

```

1      ; setup varying pwm
2      movlw    0x1
3      movwf    pwm_add
4      clrf     pwm_cnt
5
6      ; set TMR2 period to 255
7      banksel  PR2
8      movlw    0xff
9      movwf    PR2
10
11     ; set PWM mode for CCP1

```

```

12      banksel CCP1CON
13      movlw 0x0c
14      movwf CCP1CON
15
16      ; set the duty cycle to 0%
17      banksel CCPR1L
18      clrf CCPR1L
19
20      ; clear the TMR2 flag
21      banksel PIR1
22      bcf PIR1, TMR2IF
23
24      ; setup interrupts
25      banksel PIE1
26      bsf PIE1, TMR2IE
27
28      ; setup interrupts
29      bcf INTCON, RBIF
30      bsf INTCON, PEIE
31      bsf INTCON, GIE
32
33      ; set the TMR2 presaler to 1:16
34      banksel T2CON
35      bsf T2CON, 0
36      bsf T2CON, 1
37
38      ; force TMR2 counter to immediately overflow
39      movlw 0xff
40      banksel TMR2
41      movwf TMR2

```

Listing 6: Assembly code configuring the hardware PWM

5.2.2 Enable & Disable

To enable PWM, the timer needs to be turned on. This is done by setting the TMR2ON flag in T2CON (bit 2). Algorithm 17 is used to represent the corresponding assembly instructions in Listing 7.

Algorithm 17 Pseudo-code enabling the hardware PWM

Enable hardware PWM

```

1      ; enable TMR2
2      bsf T2CON, TMR2ON

```

Listing 7: Assembly code enabling the hardware PWM

5.3 Task 5

This version of Task 5 uses hardware PWM instead of the timer to generate the buzzer signal. Additionally, instead of outputting a tone on the buzzer when BUTB is pressed, BUTB toggles the tone on and off.

The reset program configures hardware PWM. A loop is entered waiting for BUTB to be pressed. When it is, the hardware PWM is toggled on or off. The pseudo-code for the reset program is in Algorithm 18.

Algorithm 18 Program 5 Version 2 Reset Program

Configure hardware PWM

loop

if $PORTA_0$ goes low **then**

 Toggle Pulse Width Modulation

end if

end loop

After transmitting a pulse, the PWM interrupt is fired. The interrupt does nothing, but is required for hardware PWM to work. Listing 15 shows the assembly code implementing the pseudo-code in Algorithm 18.

5.4 Task 6

This version of Task 6 uses hardware PWM instead of the timer to generate the buzzer signal.

The reset program clears `pwm_cnt`, and configures and enables hardware PWM. Once PWM is enabled, the pulse train is handled by the PWM interrupt, so an empty loop is entered. The pseudo-code for the reset program is in Algorithm 19.

Algorithm 19 Program 6 Version 2 Reset Program

```
pwm_cnt ← 0
pwm_add ← 1
Configure hardware PWM
Enable hardware PWM
loop
end loop
```

After transmitting a pulse, the PWM interrupt is fired. To create the pulse train, `pwm_cnt` gets incremented/decremented between 0 and 128, and then written to `CCPR1L` to become the duty cycle for the next pulse. The pseudo-code for the PWM interrupt is in Algorithm 20.

Algorithm 20 Program 6 Version 2 PWM Interrupt

```
pwm_cnt ← pwm_cnt + pwm_add
if pwm_cnt = 0 then
    pwm_add ← 1
end if
if pwm_cnt = 128 then
    pwm_add ← -1
end if
CCPR1L ← pwm_cnt
```

Listing 16 shows the assembly code implementing the pseudo-code in Algorithm 19 and Algorithm 20.

6 References

- [1] *PIC16F84A Data Sheet*, Microchip, 2001. [Online]. Available: <https://ww1.microchip.com/downloads/en/devicedoc/35007b.pdf>, (accessed: 12.12.2024).
- [2] *LM340, LM340A and LM7805 Family Wide V_{IN} 1.5-A Fixed Voltage Regulators*, Texas Instruments, 2000. [Online]. Available: <https://www.ti.com/lit/ds/symlink/lm7800.pdf>, (accessed: 28.12.2024).
- [3] P. Horowitz and W. Hill, *The Art of Electronics*, 3rd ed., 2015.
- [4] “Ngspice, the open source Spice circuit simulator,” ngspice. (2024), [Online]. Available: <https://ngspice.sourceforge.io/>. (accessed: 26.12.2024).
- [5] *LOW NOISE GENERAL PURPOSE AUDIO AMPLIFIERS*, STMicroelectronics, 2006. [Online]. Available: <https://www.st.com/resource/en/datasheet/cd00003228.pdf>, (accessed: 05.01.2025).
- [6] *HDSP-550x, HDSP-552x, HDSP-560x, HDSP-562x, HDSP-570x, HDSP-572x, HDSP-H15x Series*, Broadcom, 2024. [Online]. Available: <https://docs.broadcom.com/doc/AV02-1107EN>, (accessed: 08.01.2025).
- [7] *MPASMTM Assembler, MPLINKTM Object Linker, MPLIBTM Object Librarian User’s Guide*, Microchip, 2013. [Online]. Available: <https://ww1.microchip.com/downloads/en/devicedoc/33014L.pdf>, (accessed: 05.12.2024).
- [8] *PIC16F882/883/884/886/887*, Microchip, 2015. [Online]. Available: <https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/40001291H.pdf>, (accessed: 11.12.2024).

A Assembly Code

A.1 Task 1

```
1 ;
2 ; name:      task1
3 ; objective:
4 ;   Display the hex digits (0-f) sequentially on a 7 segment LCD display.
5 ;   Wait 200ms between changing.
6 ;
7 ; design:
8 ;   Store the hex digit combinations in the PIC's EEPROM.
9 ;   Increment through them all, waiting 200ms inbetween using the timer interrupt
10 ;   When at the end, reset back to the beginning
11 ;
12
13 #include p16f84a.inc
14
15 eeprom          code      0x2100
16 hex_7seg:       de 0x40
17                 de 0x79
18                 de 0x24
19                 de 0x30
20                 de 0x19
21                 de 0x12
22                 de 0x02
23                 de 0x78
24                 de 0x00
25                 de 0x10
26                 de 0x08
27                 de 0x03
28                 de 0x46
29                 de 0x21
30                 de 0x06
31                 de 0x0e
32 hex_7seg_end:
33
34                 udata_shr
35 w_temp:         res 1
36 s_temp:         res 1
37
38 tmr0_cnt:       res 1
39 tmr0_end:       res 1
40
41 ; reset and interrupt vectors
42
43 rst             code      0x00
44     goto        start
45 int             code      0x04
46     goto        handler
47
48 ; code
49
50 prog            code
51 start:
52     ; setup PORTB
53     banksel PORTB
54     clrf        PORTB    ; clear PORTB pins
55     banksel TRISB
56     clrf        TRISB    ; set PORTB as outputs
57
58     ; the timer interrupts every 1ms
59     ; use a counter (0xc8 is 200)
60     movlw       0xc8
61     movwf       tmr0_cnt
62
```

```

63      ; set TMRO clock to internal, and set prescaler to 1:4
64      banksel OPTION_REG
65      movf    OPTION_REG, w
66      andlw   0xc0      ; mask non-TMRO settings
67      iorlw   0x01      ; set TMRO settings
68      movwf   OPTION_REG
69
70      ; setup interrupts
71      bcf     INTCON, RBIF
72      bsf     INTCON, TOIE
73      bsf     INTCON, GIE
74
75      ; force TMRO counter to immediately overflow
76      movlw   0xff
77      banksel TMRO
78      movwf   TMRO
79
80 reset:
81      ; reset EEADR to start of hex digits
82      movlw   hex_7seg
83      banksel EEADR
84      movwf   EEADR
85
86 loop:
87      ; wait for 200ms flag to go high
88      btfss   tmr0_end, 1
89      goto    loop
90      ; clear 200ms flag
91      bcf     tmr0_end, 1
92
93      ; get the next hex digit
94      banksel EECON1
95      bsf     EECON1, RD      ; read from the EEPROM into EEDATA
96      banksel EEDATA
97      movf    EEDATA, w
98
99      ; display the next hex digit
100     banksel PORTB
101     movwf    PORTB
102
103     ; increment to next hex digit
104     banksel EEADR
105     incf     EEADR, f
106
107     ; reset if at end
108     movf     EEADR, w
109     sublw    hex_7seg_end
110     btfsc    STATUS, Z
111     goto     reset
112
113     goto     loop
114
115
116 handler:
117     ; store Working and STATUS register
118     movwf    w_temp
119     swapf    STATUS, w
120 ; tmr0 is 2
121     movwf    s_temp
122
123     ; check if TMRO flag high
124     btfss    INTCON, TOIF
125     goto     handler_end
126
127 tmr0_handler:
128     ; clear the TMRO flag

```

```

129         bcf      INTCON, TOIF
130
131 ; tmr0 is 3
132 ; TMRO interrupt caused by overflow
133 ; 4 * 250 = 1000 (1ms)
134 ; so offset TMRO by 6
135 ; however TMRO has been running in background
136 ; so put 10 into TMRO at the correct instruction
137 movlw    0x0a
138 banksel  TMRO
139 nop
140 nop
141 ; tmr0 is 4
142 movwf    TMRO
143
144 ; decrment counter and reset if 0
145 decf     tmr0_cnt, f
146 btfss    STATUS, Z
147         goto     handler_end
148
149 ; reset counter
150 movlw    0xc8
151 movwf    tmr0_cnt
152
153 ; set flag at 200ms mark
154 bsf      tmr0_end, 1
155
156 handler_end:
157 ; restore Working and STATUS register
158 swapf    s_temp, w
159 movwf    STATUS
160 swapf    w_temp, f
161 swapf    w_temp, w
162
163         retfie
164
165         end

```

Listing 8: Assembly code for Task 1

A.2 Task 2

```
1 ;
2 ; name:      task2
3 ; objective:
4 ;   Display 'L' or 'r' on a 7 segment LCD display when
5 ;   either button1 or button2 are pressed down
6 ;   respectively
7 ;
8
9 #include p16f84a.inc
10
11 ; reset and interrupt vectors
12
13 rst          code      0x00
14             goto      start
15 int          code      0x04
16             retfie
17
18 ; code
19
20 prog         code
21 start:
22             ; PORTA is input by default
23             ; setup PORTB
24             banksel PORTB
25             clrf      PORTB      ; clear PORTB pins
26             banksel TRISB
27             clrf      TRISB      ; set PORTB as outputs
28
29 loop:
30             banksel PORTA
31             ; ' ' by default
32             movlw      0x7f
33             ; if button1 (pin A0) pressed, 'L'
34             btfss      PORTA, RA0
35             movlw      0x47
36             ; if button2 (pin A1) pressed, 'r'
37             btfss      PORTA, RA1
38             movlw      0x2f
39
40             ; display ' ', 'L', or 'r' on the LCD
41             banksel PORTB
42             movwf      PORTB
43
44             goto      loop
45
46             end
```

Listing 9: Assembly code for Task 2

A.3 Task 3

```
1 ;
2 ; name:      task3
3 ; objective:
4 ;   Display '8' or ' ' on a 7 segment LCD display when
5 ;   either button1 or button2 are pressed down
6 ;   respectively and keep showing the value after
7 ;   button release
8 ;
9
10 #include p16f84a.inc
11
12 ; reset and interrupt vectors
13
14 rst          code      0x00
15             goto      start
16 int          code      0x04
17             retfie
18
19 ; code
20
21 prog         code
22 start:
23             ; PORTA is input by default
24             ; setup PORTB
25             banksel PORTB
26             clrf      PORTB    ; clear PORTB pins
27             banksel TRISB
28             clrf      TRISB    ; set PORTB as outputs
29
30             ; ' ' by default
31             movlw     0x7f
32
33 loop:
34             banksel PORTA
35             ; if button1 (pin A0) pressed, '8'
36             btfss     PORTA, RA0
37             movlw     0x00
38             ; if button2 (pin A1) pressed, ' '
39             btfss     PORTA, RA1
40             movlw     0x7f
41
42             ; display ' ' or '8' on the LCD
43             banksel PORTB
44             movwf     PORTB
45
46             goto      loop
47
48             end
```

Listing 10: Assembly code for Task 3

A.4 Task 4

```
1
2 #include p16f84a.inc
3
4 eeprom          code 0x2100
5 hex_7seg:       de 0x40
6                 de 0x79
7                 de 0x24
8                 de 0x30
9                 de 0x19
10                de 0x12
11                de 0x02
12                de 0x78
13                de 0x00
14                de 0x10
15 hex_7seg_end:
16
17                udata_shr
18 press:         res 1
19 prev_press:    res 1
20
21 ; reset and interrupt vectors
22
23 rst            code    0x00
24                goto    start
25 int            code    0x04
26                retfie
27
28 ; code
29
30 prog          code
31 start:
32                ; PORTA is input by default
33                ; setup PORTB
34                banksel PORTB
35                clrf    PORTB ; Clear PORTB pins
36                banksel TRISB
37                clrf    TRISB ; Set PORTB as outputs
38
39                ; clear press and prev_press
40                clrf    press
41                clrf    prev_press
42
43                ; set EEADR to digit 5
44                movlw   hex_7seg+5
45                banksel EEADR
46                movwf   EEADR
47
48 loop:
49                ; read buttons from PORTA
50                banksel PORTA
51                movf    PORTA, w
52
53                ; mask the button pins (0,1)
54                andlw   0x03
55                movwf   press ; store new button state
56
57                ; W = (press ^ prev_press) & prev_press
58                ; corresponding bit high if pin changed from high to low
59                xorwf   prev_press, w
60                andwf   prev_press, f ; store high to low state
61
62                ; if button1 (pin A0) high to low
63                btfsc   prev_press, 0
64                call    up_pressed
```

```

65         ; if button2 (pin A1) high to low
66         btfsc    prev_press, 1
67         call     down_pressed
68
69         ; store new button state in previous
70         movf     press, w
71         movwf    prev_press
72
73         ; get the hex digit
74         banksel  EECON1
75         bsf      EECON1, RD    ; read from the EEPROM into EEDATA
76         banksel  EEDATA
77         movf     EEDATA, w
78
79         ; display the next hex digit
80         banksel  PORTB
81         movwf    PORTB
82
83         goto     loop
84
85 up_pressed:
86         ; stop incrementing if at highest digit
87         movlw    hex_7seg_end-1
88         banksel  EEADR
89         subwf    EEADR, w
90         btfss    STATUS, Z
91         incf     EEADR, f      ; safe to increment
92         return
93
94 down_pressed:
95         ; stop decrementing if at highest digit
96         movlw    hex_7seg
97         banksel  EEADR
98         subwf    EEADR, w
99         btfss    STATUS, Z
100         decf     EEADR, f      ; safe to decrement
101         return
102
103         end

```

Listing 11: Assembly code for Task 4

A.5 Task 5

```
1 ;
2 ; name:      task5
3 ; objective:
4 ;   Buzz the piezo via PWM with a 50% duty cycle
5 ;
6 ; design:
7 ;   Use TMRO to control the piezo.
8 ;
9
10 #include p16f84a.inc
11
12 ; reset and interrupt vectors
13
14 rst          code      0x0
15             goto      start
16 int          code      0x04
17             goto      handler
18
19 ; code
20
21 prog         code
22 start:
23             ; setup PORTB
24             banksel PORTB
25             clrf      PORTB      ; clear PORTB pins
26             banksel TRISB
27             clrf      TRISB      ; set PORTB as outputs
28
29             ; set TMRO clock to internal, and set prescaler to 1:4
30             banksel OPTION_REG
31             movf      OPTION_REG, w
32             andlw     0xc0      ; mask non-TMRO settings
33             iorlw     0x01      ; set TMRO settings
34             movwf     OPTION_REG
35
36             ; setup interrupts
37             bcf       INTCON, RBIF
38             bsf       INTCON, TOIE
39             bsf       INTCON, GIE
40
41             ; force TMRO counter to immediately overflow
42             movlw     0xff
43             banksel TMRO
44             movwf     TMRO
45
46 loop:
47             ; do nothing
48             goto      loop
49
50
51 handler:
52             ; check if TMRO flag high
53             btfss     INTCON, TOIF
54             goto      handler_end
55 ; tmr0 is 2
56
57 tmr0_hndlr:
58             ; clear the TMRO flag
59             bcf       INTCON, TOIF
60
61             ; TMRO interrupt caused by overflow
62             ; 4 * 125 = 500 (0.5ms)
63             ; so offset TMRO by 131
64             ; however TMRO has been running in background
```

```

65      ; so put 134 into TMRO at the correct instruction
66      movlw    0x86
67      banksel  TMRO
68      nop
69      ; tmr0 is 3
70      movwf    TMRO
71
72      ; skip toggle if button not pressed
73      banksel  PORTA
74      btfsc    PORTA, 0
75      goto     handler_end
76
77      ; toggle the piezo (pin B7) via xor
78      banksel  PORTB
79      movf     PORTB, w
80      xorlw    0x80
81      movwf    PORTB
82      ; fallthrough
83
84 handler_end:
85      retfie
86
87      end

```

Listing 12: Assembly code for Task 5

A.6 Task 6

```
1 ;
2 ; name:      task6
3 ; objective:
4 ;   Buzz the piezo via PWM with a varying duty cycle
5 ;
6 ; design:
7 ;   Use TMRO to control the piezo.
8 ;
9
10 #include p16f84a.inc
11
12             udata_shr
13 tmr0_cnt:   res 1
14 tmr0_add:   res 1
15
16 ; reset and interrupt vectors
17
18 rst         code    0x0
19             goto    start
20 int         code    0x04
21             goto    handler
22
23 ; code
24
25 prog        code
26 start:
27             ; setup PORTB
28             banksel PORTB
29             clrf    PORTB    ; clear PORTB pins
30             banksel TRISB
31             clrf    TRISB    ; set PORTB as outputs
32
33             ; setup varying pwm
34             clrf    tmr0_cnt
35             movlw   0x1
36             movwf   tmr0_add
37
38             ; set TMRO clock to internal, and set prescaler to 1:32
39             banksel OPTION_REG
40             movf    OPTION_REG, w
41             andlw   0xc0    ; mask non-TMRO settings
42             iorlw   0x04    ; set TMRO settings
43             movwf   OPTION_REG
44
45             ; setup interrupts
46             bcf     INTCON, RBIF
47             bsf     INTCON, TOIE
48             bsf     INTCON, GIE
49
50             ; force TMRO counter to immediately overflow
51             movlw   0xff
52             banksel TMRO
53             movwf   TMRO
54
55 loop:
56             ; do nothing
57             goto    loop
58
59
60 handler:
61             ; check if TMRO flag high
62             btfss   INTCON, TOIF
63             goto    handler_end
64
```

```

65 tmr0_hndlr:
66     ; clear the TMRO flag
67     bcf      INTCON, TOIF
68
69     ; toggle the piezo (pin B7) via xor
70     banksel  PORTB
71     movf     PORTB, w
72     xorlw    0x80
73     movwf    PORTB
74
75     ; handle high or low phase
76     btfss    PORTB, 7
77     goto     tmr0_hndlr_main
78
79     ; 255 - tmr0_cnt (for high time)
80     movf     tmr0_cnt, w
81     sublw    0xff
82     ;sublw    0x00
83
84     goto     tmr0_hndlr_set
85
86 tmr0_hndlr_main:
87     ; tmr0_cnt += tmr0_add
88     ; tmr0_add is either 0x01 (+1) or 0xff (-1)
89     movf     tmr0_add, w
90     addwf    tmr0_cnt, f
91
92     ; if tmr0_cnt is 0, start incrementing
93     btfsc    STATUS, Z
94     movlw    0x01
95     ; if tmr0_cnt is 128, start decrementing
96     btfsc    tmr0_cnt, 7
97     movlw    0xff
98     movwf    tmr0_add
99
100    ; load tmr0_cnt (for low time)
101    movf     tmr0_cnt, w
102    ; fallthrough
103
104 tmr0_hndlr_set:
105     ; set the TMRO counter
106     banksel  TMRO
107     movwf    TMRO
108
109 handler_end:
110     retfie
111
112     end

```

Listing 13: Assembly code for Task 6

A.7 Task 4 Version 2

```
1
2 #include p16f84a.inc
3
4 eeprom          code 0x2100
5 hex_7seg:       de 0x40
6                 de 0x79
7                 de 0x24
8                 de 0x30
9                 de 0x19
10                de 0x12
11                de 0x02
12                de 0x78
13                de 0x00
14                de 0x10
15 hex_7seg_end:
16
17                udata_shr
18 w_temp:        res 1
19 s_temp:        res 1
20
21 press:         res 1
22 prev_press:    res 1
23 but1_cnt:      res 1
24 but2_cnt:      res 1
25
26 ; reset and interrupt vectors
27
28 rst            code    0x00
29                goto    start
30 int            code    0x04
31                goto    handler
32
33 ; code
34
35 prog           code
36 start:
37     ; PORTA is input by default
38     ; setup PORTB
39     banksel PORTB
40     clrf PORTB ; Clear PORTB pins
41     banksel TRISB
42     clrf TRISB ; Set PORTB as outputs
43
44     ; clear prev_press
45     clrf prev_press
46
47     ; set EEADR to digit 5
48     movlw hex_7seg+5
49     banksel EEADR
50     movwf EEADR
51
52     ; set TMRO clock to internal, and set prescaler to 1:8
53     banksel OPTION_REG
54     movf OPTION_REG, w
55     andlw 0xc0 ; mask non-TMRO settings
56     iorlw 0x02 ; set TMRO settings
57     movwf OPTION_REG
58
59     ; setup interrupts
60     bcf INTCON, RBIF
61     bsf INTCON, TOIE
62     bsf INTCON, GIE
63
64     ; force TMRO counter to immediately overflow
```

```

65      movlw    0xff
66      banksel  TMRO
67      movwf    TMRO
68
69 loop:
70      ; read buttons from PORTA
71      banksel  PORTA
72      movf     PORTA, w
73
74      ; mask the buttons pins (0,1)
75      andlw    0x03
76      movwf    press    ; store new button state
77
78      ; W = (press ^ prev_press) & prev_press
79      ; corresponding bit high if pin changed from high to low
80      xorwf    prev_press, w
81      andwf    prev_press, f    ; store high to low state
82
83      movlw    0xc8
84      ; if button1 (pin A0) high to low
85      btfsc    prev_press, 0
86      call     up_and_reset
87      ; if button2 (pin A1) high to low
88      btfsc    prev_press, 1
89      call     down_and_reset
90
91      ; if button1 (pin A0) low
92      btfsc    press, 0
93      clrf     but1_cnt
94      ; if button2 (pin A1) low
95      btfsc    press, 1
96      clrf     but2_cnt
97
98      ; store new button state in previous
99      movf     press, w
100     movwf    prev_press
101
102     ; get the hex digit
103     banksel  EECON1
104     bsf      EECON1, RD    ; read from the EEPROM into EEDATA
105     banksel  EEDATA
106     movf     EEDATA, w
107
108     ; display the next hex digit
109     banksel  PORTB
110     movwf    PORTB
111
112     goto     loop
113
114 up_and_reset:
115     ; reset counter
116     movwf    but1_cnt
117
118     ; stop incrementing if at highest digit
119     movlw    hex_7seg_end-1
120     banksel  EEADR
121     subwf    EEADR, w
122     btfss    STATUS, Z
123     incf     EEADR, f    ; safe to increment
124     return
125
126 down_and_reset:
127     ; reset counter
128     movwf    but2_cnt
129
130     ; stop decrementing if at highest digit

```

```

131      movlw    hex_7seg
132      banksel EEADR
133      subwf    EEADR, w
134      btfss    STATUS, Z
135      decf     EEADR, f      ; safe to decrement
136      return
137
138 handler:
139      ; store Working and STATUS register
140      movwf    w_temp
141      swapf    STATUS, w
142      ; tmr0 is 1
143      movwf    s_temp
144
145      ; check if TMRO flag high
146      btfss    INTCON, TOIF
147      goto     handler_end
148
149 tmr0_handler:
150      ; clear the TMRO flag
151      bcf       INTCON, TOIF
152
153      ; TMRO interrupt caused by overflow
154      ; 8 * 250 = 1000 (2ms)
155      ; so offset TMRO by 6
156      ; however TMRO has been running in background
157      ; so put 8 into TMRO at the correct instruction
158      movlw    0x08
159      banksel  TMRO
160      nop
161      nop
162      ; tmr0 is 2
163      movwf    TMRO
164
165      movlw    0x32
166      ; decrement button1 counter and reset if 0
167      decf     but1_cnt, f
168      btfsc    STATUS, Z
169      call     up_and_reset
170      ; decrement button2 counter and reset if 0
171      decf     but2_cnt, f
172      btfsc    STATUS, Z
173      call     down_and_reset
174
175 handler_end:
176      ; restore Working and STATUS register
177      swapf    s_temp, w
178      movwf    STATUS
179      swapf    w_temp, f
180      swapf    w_temp, w
181
182      retfie
183
184      end

```

Listing 14: Assembly code for Task 4 Version 2

A.8 Task 5 Version 2

```
1 ;
2 ; name:      task5
3 ; objective:
4 ;   Buzz the piezo via PWM with a 50% duty cycle
5 ;
6 ; design:
7 ;   Use hardware PWM to control the piezo.
8 ;
9
10 #include p16f882.inc
11
12     __config    _CONFIG1, _WDT_OFF
13
14             udata_shr
15 w_temp:     res 1
16 s_temp:     res 1
17
18 press:      res 1
19 prev_press: res 1
20
21 ; reset and interrupt vectors
22
23 rst         code    0x0
24     goto    start
25 int         code    0x04
26     goto    handler
27
28 ; code
29
30 prog        code
31 start:
32     ; disable the watchdog
33     banksel WDTCN
34     bcf     WDTCN, SWDTEN
35
36     ; disable analogue inputs on PORTA
37     banksel ANSEL
38     clrf    ANSEL
39
40     ; set TMR2 period to 255
41     movlw   0xff
42     banksel PR2
43     movwf   PR2
44
45     ; set PWM mode for CCP1
46     movlw   0x0c
47     banksel CCP1CON
48     movwf   CCP1CON
49
50     ; set the duty cycle to 50%
51     movlw   0x80
52     banksel CCPR1L
53     movwf   CCPR1L
54
55     ; clear the TMR2 flag
56     banksel PIR1
57     bcf     PIR1, TMR2IF
58
59     ; setup interrupts
60     banksel PIE1
61     bsf     PIE1, TMR2IE
62
63     ; setup interrupts
64     bcf     INTCON, RBIF
```

```

65      bsf      INTCON, PEIE
66      bsf      INTCON, GIE
67
68      ; set the TMR2 presaler to 1:16
69      movlw    0x03
70      banksel  T2CON
71      iorwf    T2CON, f
72
73      ; force TMR2 counter to immediately overflow
74      movlw    0xff
75      banksel  TMR2
76      movwf    TMR2
77
78 loop:
79      ; read buttons from PORTA
80      banksel  PORTA
81      movf     PORTA, w
82
83      ; mask the button pin 0
84      andlw    0x01
85      movwf    press ; store the new button state
86
87      ; W = (press ^ prev_press) & prev_press
88      ; STATUS zero flag low if if pin changed from high to low
89      xorwf    prev_press, w
90      andwf    prev_press, w
91
92      ; if button1 (pin A0) high to low
93      btfss    STATUS, Z
94      call     toggle_tmr2
95
96      ; store new button state in previous
97      movf     press, w
98      movwf    prev_press
99
100     goto     loop
101
102 toggle_tmr2:
103     ; toggle TMR2
104     movlw    0x04
105     banksel  T2CON
106     xorwf    T2CON, f
107     return
108
109
110 handler:
111     ; store Working and STATUS register
112     movwf    w_temp
113     swapf    STATUS, w
114     movwf    s_temp
115
116     ; check if TMR2 flag high
117     banksel  PIR1
118     btfss    PIR1, TMR2IF
119     goto     handler_end
120
121 tmr2_handler:
122     ; clear the TMR2 flag
123     bcf      PIR1, TMR2IF
124
125     ; make pin C2 an output
126     banksel  TRISC
127     bcf      TRISC, 2
128     ; fallthrough
129
130 handler_end:

```

```
131      ; restore Working and STATUS register
132      swapf    s_temp, w
133      movwf    STATUS
134      swapf    w_temp, f
135      swapf    w_temp, w
136
137      retfie
138
139      end
```

Listing 15: Assembly code for Task 5 Version 2

A.9 Task 6 Version 2

```
1 ;
2 ; name:      task6
3 ; objective:
4 ;   Buzz the piezo via PWM with a varying duty cycle
5 ;
6 ; design:
7 ;   Use hardware PWM to control the piezo.
8 ;
9
10
11 #include p16f882.inc
12
13     __config    _CONFIG1, _WDT_OFF
14
15             udata_shr
16 pwm_add:    res 1
17 pwm_cnt:    res 1
18
19 ; reset and interrupt vectors
20
21 rst                code    0x0
22     goto    start
23 int                code    0x04
24     goto    handler
25
26 ; code
27
28 prog                code
29 start:
30     ; disable the watchdog
31     banksel WDTCN
32     bcf     WDTCN, SWDTEN
33
34     ; setup varying pwm
35     movlw   0x1
36     movwf   pwm_add
37     clrf    pwm_cnt
38
39     ; set TMR2 period to 255
40     banksel PR2
41     movlw   0xff
42     movwf   PR2
43
44     ; set PWM mode for CCP1
45     banksel CCP1CON
46     movlw   0x0c
47     movwf   CCP1CON
48
49     ; set the duty cycle to 0%
50     banksel CCPR1L
51     clrf    CCPR1L
52
53     ; clear the TMR2 flag
54     banksel PIR1
55     bcf     PIR1, TMR2IF
56
57     ; setup interrupts
58     banksel PIE1
59     bsf     PIE1, TMR2IE
60
61     ; setup interrupts
62     bcf     INTCON, RBIF
63     bsf     INTCON, PEIE
64     bsf     INTCON, GIE
```

```

65
66      ; set the TMR2 presaler to 1:16
67      banksel T2CON
68      bsf      T2CON, 0
69      bsf      T2CON, 1
70
71      ; enable TMR2
72      bsf      T2CON, TMR2ON
73
74      ; force TMR2 counter to immediately overflow
75      movlw    0xff
76      banksel  TMR2
77      movwf    TMR2
78
79  loop:
80      ; do nothing
81      goto     loop
82
83
84  handler:
85      ; check if TMR2 flag high
86      banksel  PIR1
87      btfss    PIR1, TMR2IF
88      goto     handler_end
89
90  tmr2_handler:
91      ; clear the TMR2 flag
92      bcf      PIR1, TMR2IF
93
94      ; tmr0_cnt += tmr0_add
95      ; tmr0_add is either 0x01 (+1) or 0xff (-1)
96      movf     pwm_add, w
97      addwf    pwm_cnt, f
98
99      ; if tmr0_cnt is 0, start incrementing
100     btfsc     STATUS, Z
101         movlw  0x1
102     ; if tmr0_cnt is 128, start decrementing
103     btfsc     pwm_cnt, 7
104         movlw  0xff
105     movwf     pwm_add
106
107     ; load tmr0_cnt (for low time)
108     ; set the duty cycle
109     movf     pwm_cnt, w
110     addlw    0x02
111     banksel  CCPR1L
112     movwf    CCPR1L
113
114     ; make pin C2 an output
115     banksel  TRISC
116     bcf      TRISC, 2
117     ; fallthrough
118
119  handler_end:
120     retfie
121
122     end

```

Listing 16: Assembly code for Task 6 Version 2