

Programming A Seven Segment Display – Micros Peripherals & Interfaces – ACDA4004 Electrical & Electronic Engineering Degree Apprenticeship

Joshua H – Student Number: 2420839

Contents

Section 1	3
Section 2	4
Section 3	6
Section 3.1	8
Section 3.2	10
Section 3.3	12
Section 3.4	14
Section 3.5	16
References	17

[illegible]

Shown above in Figure 1 is the circuit representation of the PIC16F84 Assignment Board in Proteus 9. In the circuit, the microcontroller is used to drive a common anode seven-segment display, respond to inputs from pushbuttons and generate sound from a piezo-electric sounder. The function of the circuit is software-defined, meaning how the microcontroller drives various parts of the circuit is dependent on which task it is programmed to fulfil.

The circuit includes a common anode seven-segment display connected to PORTB of the microcontroller through current limiting resistors. A seven-segment display consists of seven individual LEDs, arranged to form numbers and a limited set of characters. By selectively driving different combinations of segments, digits from 0 to 9 and certain letters can be displayed. Seven-segment displays are most used for numerical output, such as counters, timers, and measurement readouts[3]. Due to the display being common anode, all LEDs are connected to supply voltage because of this, segments

must be biased off by driving segments' corresponding pins' low. The outputs are on Port B and are set to 01001000 or 0x12 in hexadecimal initially to display the number 5.

A piezo electric sounder is used to generate an audible tone. Through the rapid changes in voltage, the sounder converts electrical signals into mechanical vibrations producing sound. The microcontroller cannot source enough electrical current to drive the sounder however, the final pin on PORTB biases a transistor on and off. This allows the microcontroller to gate the flow of current through the sounder outputting a square wave. The speed at which the transistor is biased on and off determines the pitch of the sound and in the circuit, this is controlled through a series of delays between the pin being toggled.

Section 2

As stated previously, a seven-segment display is made up of seven LEDs controlled through seven data lines. Each data line is controlled by the microcontroller allowing the programmer to select each segment separately. The segments are generally labelled from A-G.

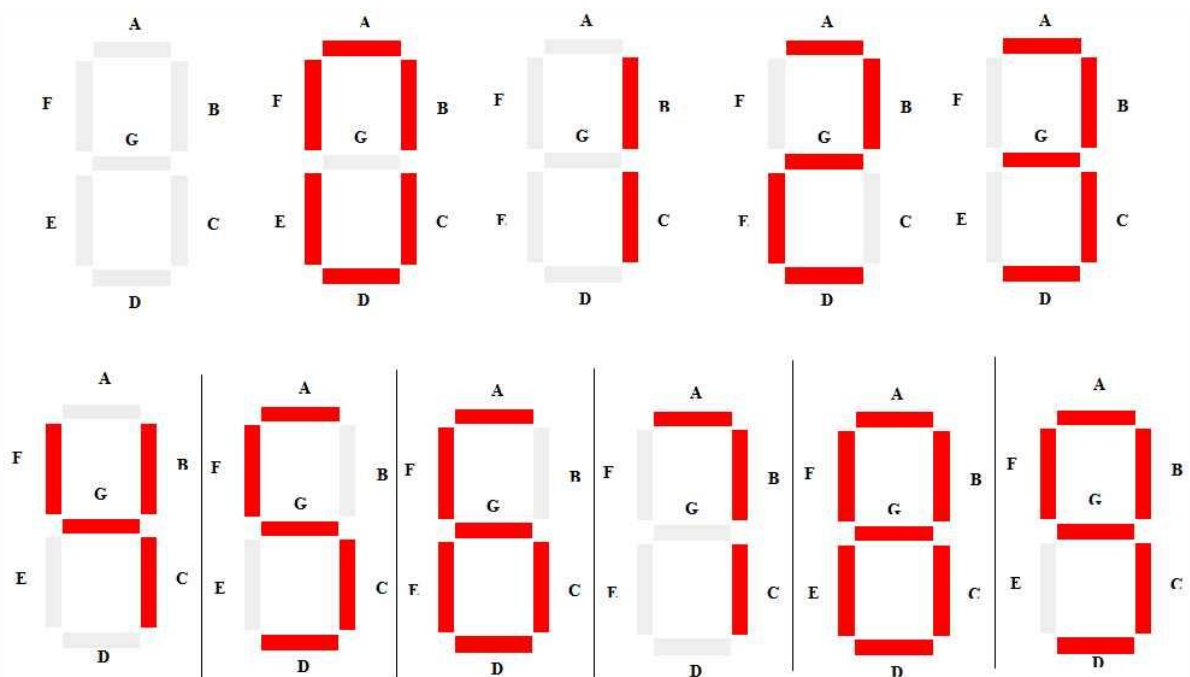


Figure 2: Seven-segment display annotated[4]

As our circuit uses a common anode, if no data pins are high all segments will be lit. By applying a voltage to a specific pin, the voltage across the LED falls enough to bias the LED off. In this circuit all pins are connected to PORTB this allows us to effectively drive the display through outputting a binary number on the port. As each number displayed on seven-segment display is unique and non-continuous the best method to program with this component in mind is a look-up table.

Denary Value	Hex Value	Binary Pin Out and Segment Correspondence							Hexadecimal Pin Out
		A	B	C	D	E	F	G	
0	0	0	0	0	0	0	0	1	0x40
1	1	1	0	0	1	1	1	1	0x79
2	2	0	0	1	0	1	0	0	0x24
3	3	0	0	0	0	1	1	0	0x30
4	4	1	0	0	1	1	0	0	0x19
5	5	0	1	0	0	1	0	0	0x12
6	6	0	1	0	0	0	0	0	0x02
7	7	0	0	0	1	1	1	1	0x78
8	8	0	0	0	0	0	0	0	0x00
9	9	1	1	1	1	0	1	1	0x18
10	A	0	0	0	1	0	0	0	0x08
11	B	1	1	0	0	0	0	0	0x03
12	C	0	1	1	0	0	0	1	0x46
13	D	1	0	0	0	0	1	0	0x21
14	E	0	1	1	0	0	0	0	0x06
15	F	0	1	1	1	0	0	0	0x0E

Table 1

To efficiently convert numerical and character values into segment patterns, the program uses a look-up table. This table is stored in program memory and using the RETLW instruction which allows for a literal value to be returned directly to the working register. The main program then only needs to loop 0-15 which using the Program Counter Low register allows the program to jump to the appropriate table entry. This is an improvement from count.asm which uses MOVLW to move a preset number to the working memory and then MOVWF to move the literal to a file to be read from later. This is inefficient as it requires extra instructions than RETLW making the program execute slower, further MOVWF stores values in file registers which is limited whereas PCL uses flash memory.

Section 3

Shown below is the common code used by all sub-tasks. In places, this code has been modified however this will be mentioned wherever necessary.

```

4; DESCRIPTION: Count to 255 at 1 second intervals and display on a seven
5;
6;           segment display in binary.
7;           The pause subroutine takes apx. 0.2 sec.
8;           Pause is called 5 times
9
10      list    p=16F84          ; select processor type
11      radix   hex              ; define radix as hexadecimal
12      #include "p16f84.inc"
13;-----
14;      cpu equates (memory map)
15portA    equ    0x05
16portB    equ    0x06          ;port B address
17trisA    equ    0x05
18trisb    equ    0x06          ;trisB address in register bank 1
19PCL       equ    0x02
20Status   equ    0x03          ;status register
21RP0      equ    5              ;bit 5 of status register
22RP1      equ    6              ;bit 6 of status register
23
24INDF     equ    0x00          ;indirect file register
25FSR      equ    0x04          ;file select register
26
27count    equ    0x0c          ; counter 0 to 255
28nCount   equ    0x0d          ; pause routine inner loop counter
29mCount   equ    0x0e          ; pause routine outer loop counter
30tablestart equ 0x10           ; constant location of lookup table
31

```

Figure 3

The figure above shows the header code. This allows the assembler to compile the code correctly for the specified microcontroller as well as allowing the programmer to define spaces in RAM for variables or constants to be stored.

```

;-----
      org      0x000

start
      bcf      Status,RP0      ; clear bank selection bits in the status reg
      bcf      Status,RP1

      bsf      Status,RP0      ; select bank 1
      movlw    0x00            ; Load w with 0x00
      movwf    trisB           ; Load it into trisb to make portb all outputs
      bcf      Status,RP0      ; reselect bank 0

      clrf     portB           ; clear all lines low
      clrf     count

      clrf     portA           ; clear all lines low
      clrf     count           ; initialize counter to 0

      call     display
;-----

```

Figure 4

The code above initialises the memory banks, ports and counter and sets them to zero. This is important as registers and memory may be corrupted by default. Previously run code may be left in memory so it is best practice to clear all registers that will be relied upon. Furthermore, this increases the programs predictability as the program starts at a known state which can help when debugging.

```

74
75 SegTable
76     addwfc PCL, f           ; add W to PCL to jump into table
77
78     retlw 0x40              ; 0
79     retlw 0x79              ; 1
80     retlw 0x24              ; 2
81     retlw 0x30              ; 3
82     retlw 0x19              ; 4
83     retlw 0x12              ; 5
84     retlw 0x02              ; 6
85     retlw 0x78              ; 7
86     retlw 0x00              ; 8
87     retlw 0x18              ; 9
88     retlw 0x08              ; A
89     retlw 0x03              ; B
90     retlw 0x46              ; C
91     retlw 0x21              ; D
92     retlw 0x06              ; E
93     retlw 0x0E              ; F
94

```

Figure 5

The code above shows the look-up table used by the code. This code is used by section 3.1 and 3.4. When the function “SegTable” is called the working register will contain a hexadecimal digit between 0 and 15. ADDWFC takes the value in the working register and adds it to the PCL, this effectively jumps to the index point in this table and returns with the value storing it in the working register.

Section 3.1

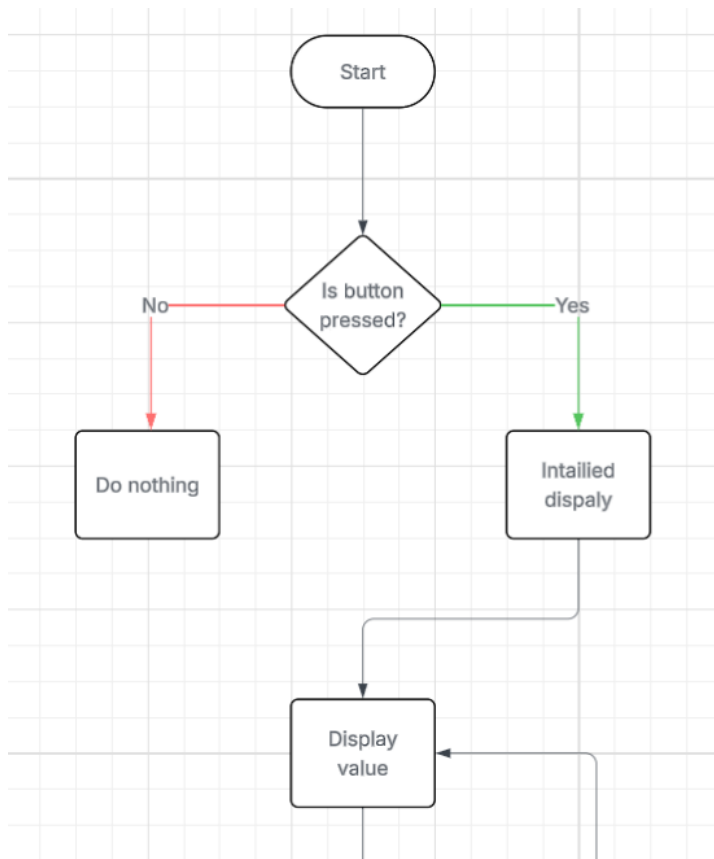


Figure 6

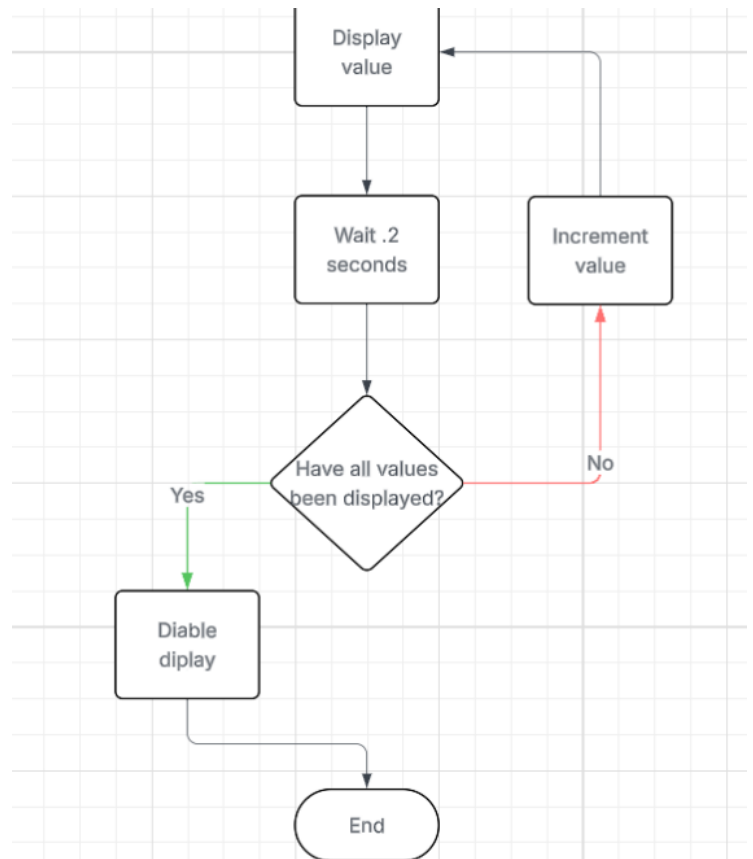


Figure 7

The task was to have the seven-segment display loop through 0-F in 0.2 second intervals. The flowchart above describes the operation. Once the program starts 0 is initially displayed, 0.2 seconds pass and the number displayed is incremented. This repeats until all values are displayed and the program ends.

```

55     call    display
56
57
58 main_loop
59     call pause200           ; delay ~0.2 s per step
60
61     incf count, f           ; increment counter
62     btfsc count, 4          ; if count > 0x0F
63     clrf count              ; wrap back to 0
64
65     call display            ; update display
66     goto main_loop         ; repeat forever
67
68
69 display
70     movf count, w           ; W = counter value 0-15
71     call SegTable           ; get segment pattern in W
72     movwf portB             ; output to 7-segment
73     return
74
75 SegTable
76     addwf PCL, f           ; add W to PCL to jump into table
77
78

```

Figure 8

The figure above shows the main loop in which the program runs. Above this is the header and start functions shown above and below is the look-up table also mentioned above. Once the program starts “0” is displayed then the main loop is entered, at which point the pause function is called. Once the pause function is finished the counter is incremented, this is then checked using the instruction BTFSC in the program this checks the fourth most significant bit is a zero to ensure the count hasn’t exceeded the index length. Once the count has been checked and is less than 16 the “display” function is called, this function moves the current value of count to the working and then passes the value to the segment table. The value from the segment table is returned and outputted on port B. “main_loop” is repeated until the index reaches fifteen and which point the program will loop back to zero.

The pause function seen in figure X below works as each iteration of the inner loop takes a total of 3 cycles to complete. If the skip doesn’t occur, this is repeated 253 times plus another cycle for when it does skip in total resulting in 761 cycles total. Then the outer loop repeats 255 times. With an instruction cycle taking 1 μ s this the total time taken to exit the loop is 194,055 μ s or 0.194 seconds.

```

// -----
98 pause200
99     movlw 0xD0             ; outer loop ~ 208
100    movwf mCount
101
102 outer_loop
103     movlw 0xFE             ; inner loop ~ 254
104     movwf nCount
105
106 inner_loop
107     decfsz nCount, f
108     goto inner_loop
109     decfsz mCount, f
110     goto outer_loop
111     return
112
113     end

```

Figure 9

Section 3.2

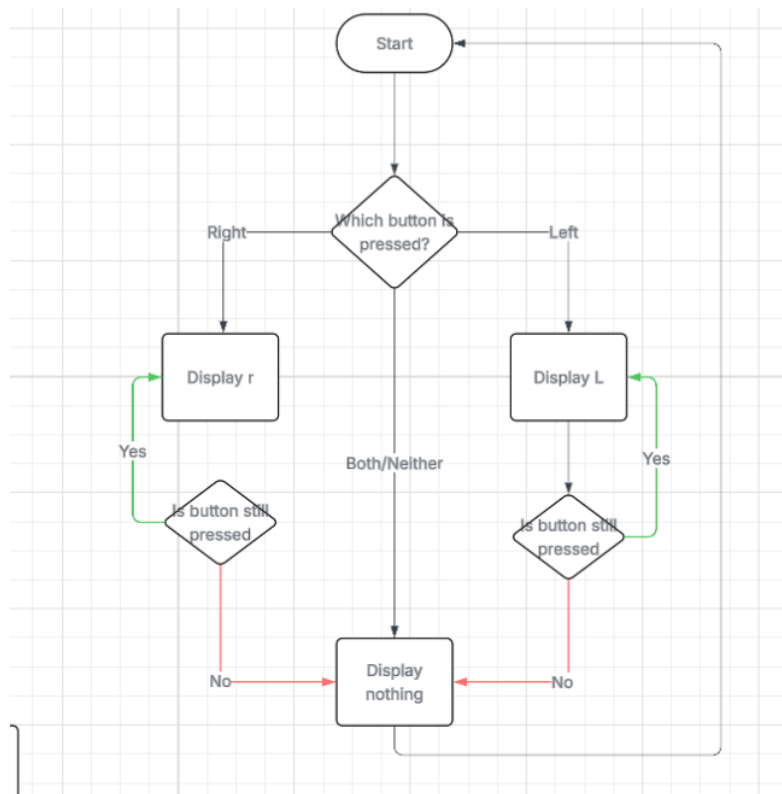


Figure 10

The task was to have the seven-segment display either and “r” or and” L” when the right or left button was pressed, respectively. The flowchart above shows the operation. If the right button is pressed a lowercase “r” will be displayed and if left button is pressed an uppercase “L” will be displayed.

```

9      call    display
0 main_loop
1      btfss   portA,0      ; check if black button (connected to RA0) is pressed
2      call    BLACKBUTTON  ; if pressed, display "r"
3      btfss   portA,1      ; check if green button (connected to RA1) is pressed
4      call    GREENBUTTON  ; if pressed, display "L"
5      goto    main_loop    ; repeat forever
6
7 BLACKBUTTON
8
9      call    pause20      ;debounce
0      btfss   portA,0      ;check if pressed
1      goto    BLACKBUTTON  ; wait for release
2      movlw   0xC7          ; segment pattern for "r"
3      movwf   portB         ; update display
4
5
6      return
7
8 GREENBUTTON
9
0      call    pause20      ;debounce
1      btfss   portA,1      ;check if pressed
2      goto    GREENBUTTON  ; wait for release
3      movlw   0xAF         ; segment pattern for "L"
4      movwf   portB         ; update display
5      return
6
  
```

Figure 11

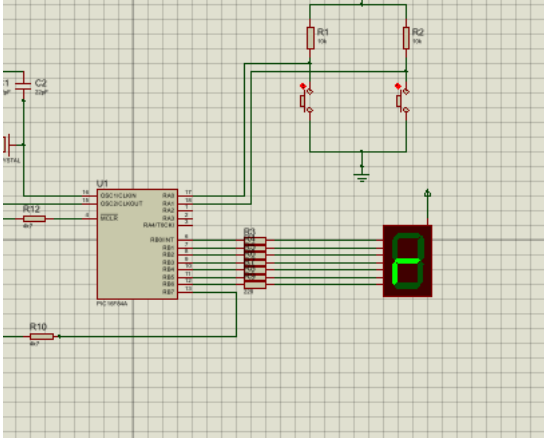


Figure 12

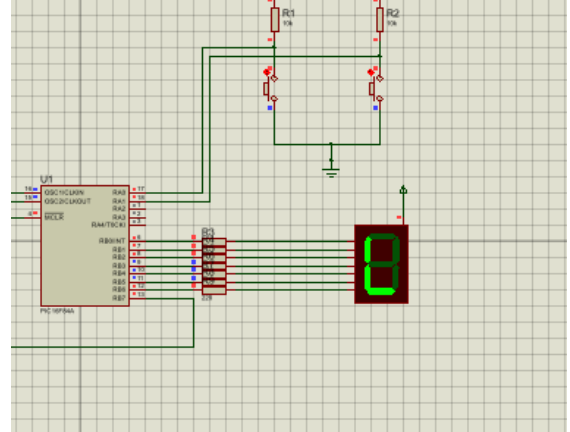


Figure 13

Above are the code and its output in Proteus. Above the code is the header and start code as shown above. Inside the main loop the program polls the two input pins, once a button is pressed the input pins go low causing BLACKBUTTON or GREENBUTTON to be called. The two button functions are almost identical, and both debounce by implementing a pause. This pause is identical to the pause200 function however the outer loop only repeats 80 times. This results in a loop of approximately 0.02 seconds. Once that has completed and the button is definitely pressed, the appropriate segments are biased off. For “L” segments A,B,C and G and for “r” segments A,B,C, D and F.

Section 3.3

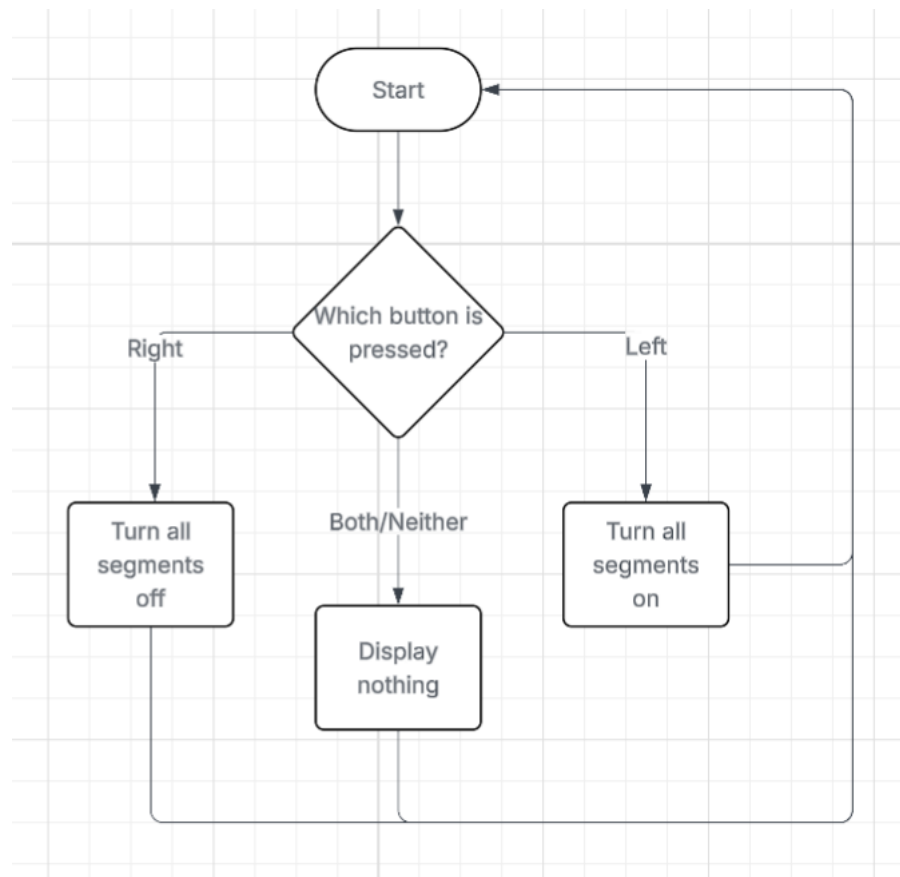


Figure 14

The task was to have the seven-segment switch entirely on and stay on. If the right button is pressed all segments were to be turned off and if left button is pressed all segments should be turned on.

```

*main.asm
59     call    display
60 main_loop
61     btfss   portA,0      ; check if black button (connected to RA0) is pressed
62     call    BLACKBUTTON ; if pressed, display "r"
63     btfss   portA,1      ; check if green button (connected to RA1) is pressed
64     call    GREENBUTTON ; if pressed, display "L"
65     goto    main_loop    ; repeat forever
66
67 BLACKBUTTON
68
69     call    pause20      ;debounce
70     btfss   portA,0      ;check if pressed
71     goto    BLACKBUTTON ; wait for release
72     movlw   0xFF         ; all on
73     movwf   portB        ; update display
74
75
76     return
77
78 GREENBUTTON
79
80     call    pause20      ;debounce
81     btfss   portA,1      ;check if pressed
82     goto    GREENBUTTON ; wait for release
83     movlw   0x00         ; all off
84     movwf   portB        ; update display
85     return
86
  
```

Figure 15

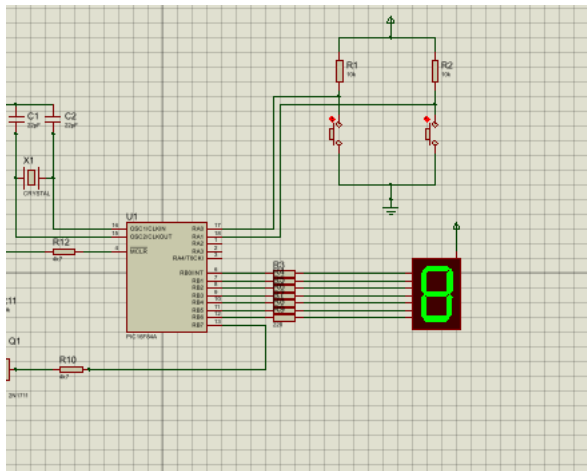


Figure 16

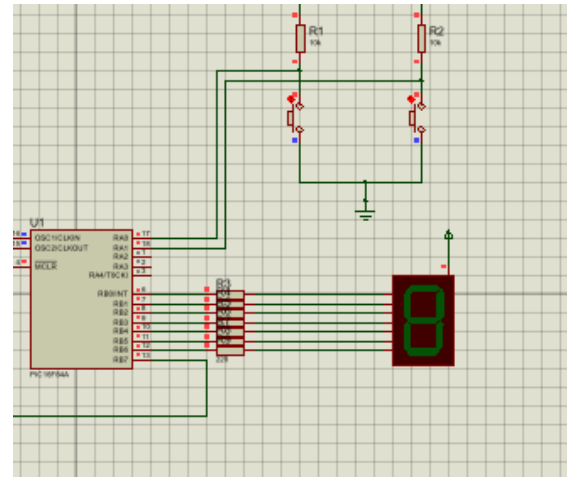


Figure 17

Above are the code and its output in Proteus. Above the code is the header and start code as shown above. Inside the main loop the program polls the two input pins, once a button is pressed the input pins go low causing BLACKBUTTON or GREENBUTTON to be called. If the BLACKBUTTON is called all pins go high resulting in all segments turning off. If the GREENBUTTON is called all pins go low resulting in all segments turning on.

Section 3.4



Figure 18

The task was to have the seven-segment switch display the values 0-9. The user then should be able to loop through the different values using the two buttons. One button should count to 9 and stay at 9 the other should count down to 0 and stay at 0.

```

54
55
56     movlw    0x05           ; display five
57     movwf    count
58
59     call     display
60 main_loop
61     btfss    portA,0        ; check if black button (connected to RA0) is pressed
62     call     BLACKBUTTON    ; if pressed, count up
63     btfss    portA,1        ; check if green button (connected to RA1) is pressed
64     call     GREENBUTTON    ; if pressed, count down
65     goto     main_loop      ; repeat forever
66
67 BLACKBUTTON
68
69     call     pause20         ;debounce
70     btfss    portA,0        ;
71     goto     BLACKBUTTON    ; wait for release
72     movf     count,w        ; move count to working
73     sublw    0x09           ; sub 9 from working
74     btfss    Status, Z      ; if count != 9
75     incf     count,f        ; increment
76     call     display        ; display
77
78     return
79
80
81 GREENBUTTON

```

Figure 19

```

68
69     call    pause20           ;debounce
70     btfss   portA,0           ;
71     goto    BLACKBUTTON      ; wait for release
72     movf    count,w           ; move count to working
73     sublw   0x09              ; sub 9 from working
74     btfss   Status, Z         ; if count != 9
75     incf    count,f           ; increment
76     call    display           ; display
77
78     return
79
80
81 GREENBUTTON
82
83     call    pause20           ;debounce
84     btfss   portA,1           ;
85     goto    GREENBUTTON      ; wait for release
86     movf    count,w           ; move count to workign
87     btfss   Status, Z         ; if count != 0
88     decf    count,f           ; decrement
89     call    display           ; display
90     return
91
92
93

```

Figure 20

Above the code is the header and start code as shown above. Above the main loop the seven-segment display is initialised with the number 5. Inside the main loop the program polls the two input pins, once a button is pressed the input pins go low causing BLACKBUTTON or GREENBUTTON to be called. If the BLACKBUTTON is called the input is debounced, count is placed in working and then count is subtracted from 9 if this output of this operation is not 9 count increments if it is 0 then the increment is skipped and 0 is displayed again. If GREENBUTTON is called the input is debounced, count is placed in working and then checked against the zero flag in the Status register. If count is not zero, then count is decremented. If it is zero, then count is displayed again.

Section 3.5

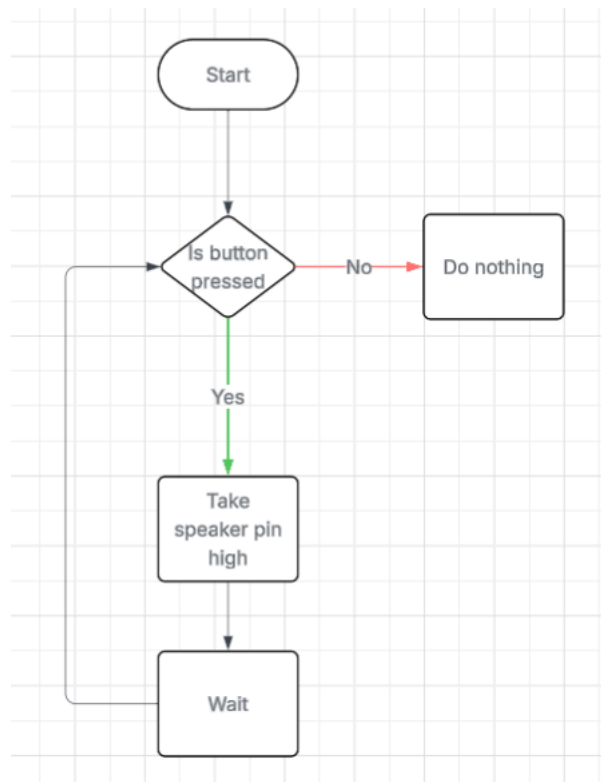


Figure 21

The task was to have the sounder beep on the press of a button. If the button is pressed the sounder should sound

```

main_loop    call    display
main_loop    btfss   portA,0      ; check if black button (connected to RA0) is pressed
main_loop    call    BLACKBUTTON  ; if pressed, beep
main_loop    goto    main_loop    ; repeat forever

BLACKBUTTON
    call    beep                ;beep
    btfss   portA,0            ; If button is released skip next
    goto    BLACKBUTTON
    return

beep
    bsf     portB,7            ; Set RB7 high
    call    short_pause        ; Delay
    bcf     portB,7            ; Set RB7 low
    call    short_pause        ; Delay
    return
  
```

Figure 22

Above the code is the header and start code as shown above. The main loop polls the black button only, once pressed beep is called this biases the transistor on causing the sounder to sound. Then a short pause occurs before the transistor is brought low, this repeats for as long as the button is held.

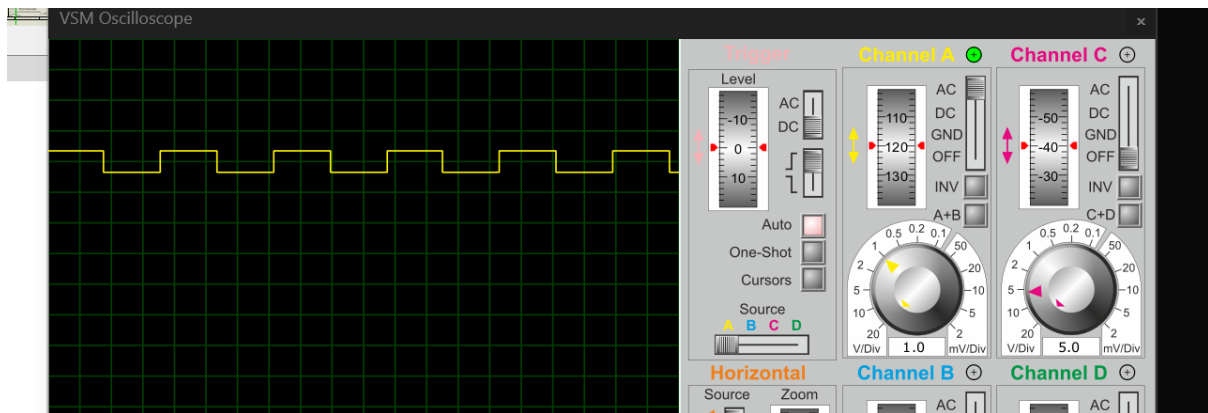


Figure 23

The figure above shows the square wave produced by the code biasing the transistor on and off.

References

- [1]
"PIC16x84," *Wikipedia*, Jun. 25, 2022. Available: <https://en.wikipedia.org/wiki/PIC16x84>
- [2]
"PIC16F84," *Microchip.com*, 2025. Available: <https://www.microchip.com/en-us/product/PIC16F84>
- [3]
Wikipedia Contributors, "Seven-segment display," *Wikipedia*, Feb. 05, 2019. Available: https://en.wikipedia.org/wiki/Seven-segment_display
- [4]
Available: <https://datavisualexpert.com/seven-segment-display-diagram>