

Using and programming the 7-segment
Display Prototype Circuit.

Micros Peripherals and Interfaces

ACDA4004

BEng Embedded Electronic Systems

Zakir Basit

Contents

Section 1.....	3
Overview of Circuit.....	3
Microcontroller	3
Crystal Oscillator Circuit	4
Buttons	4
Resistors R1-R7.....	4
7-Segment Display	5
Section 2.....	5
Section 3.....	7
Task 1.....	7
Task 2.....	9
Task 3.....	11
Task 4.....	14
Task 5.....	16
References.....	18

Section 1

Overview of Circuit

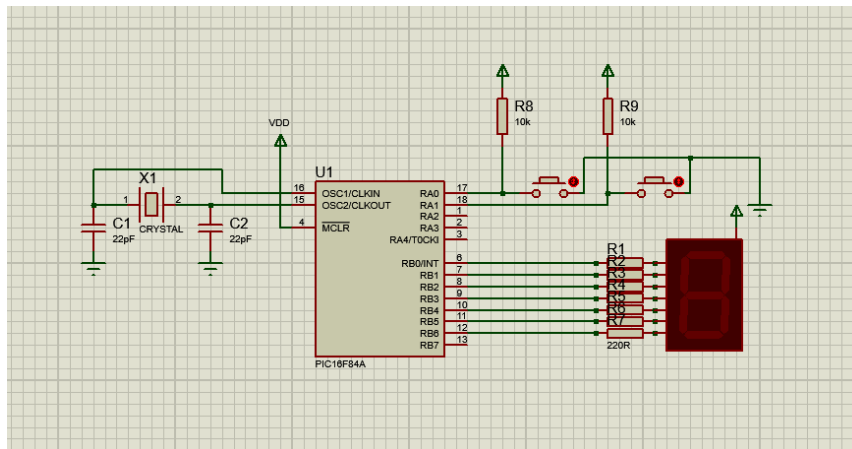


Figure 1: Screengrab of circuit.

Figure 1 consists of these key components:

- PIC16F84A Microcontroller (U1)
- Crystal Oscillator Circuit (X1, C1, C2)
- Buttons which are connected to RA0 and RA1 pins
- Resistors (R1-R7, R8 and R9)
- 7-Segment display
- Power supply

The circuit works by generating a clock signal by the crystal oscillator for the microcontroller. The microcontroller then monitors the buttons connected to RA0 and RA1 pins. The button connected to RA0 decrements the displayed value, the other button will increment. When the buttons are pressed, the microcontroller sends appropriate signals to the 7-segment display via the RB pins.

Microcontroller

Labelled as U1 in figure 1, the PIC16F84A is an 8-bit microcontroller from Microchip Technology. It features 18 pins, 13 of which are input (RA0-RA4) and output (RB0-RB7) pins that can be programmed as such. It also includes an internal EEPROM for non-volatile data storage, a 4MHz instruction cycle frequency, and supports various external oscillators [1].

In figure 1, the microcontroller serves as the central processing unit of the circuit, executing program commands and controlling the connected I/O devices. Pins RA0 and RA1 are linked to the push buttons as digital inputs to allow for user interaction. Through the RB0–RB7 pins,

which are set up as outputs, the microcontroller modifies the output shown on the 7-segment display in response to these inputs.

Additionally, the microcontroller connects to an external oscillator circuit via the OSC1/CLKIN and OSC2/CLKOUT pins, which provide the clock signal required for functioning.

Microcontrollers such as the PIC16F84A require a reliable clock source to execute instructions with exact timing. This microcontroller requires an external oscillator (such as a crystal) to generate a clock signal for internal functioning. The clock frequency directly impacts the speed of execution. For example, a 4 MHz crystal allows the PIC16F84A to execute one instruction every 1 microsecond.

Crystal Oscillator Circuit

A crystal oscillator circuit including a 4 MHz crystal (X1) and two 22 pF capacitors (C1 and C2) provides the clock signal for the PIC16F84A. While the capacitors stabilise the oscillations and filter out noise that can interfere with the clock signals, the crystal generates a dependable oscillating signal at a preset frequency that the microcontroller uses to coordinate its operations.

Buttons

The circuit includes two push buttons connected to the RA0 and RA1 input pins of the microcontroller. Pressing a button causes the corresponding pin to drop, telling the microcontroller to do something like increment or reset the screen.

Pull-up resistors (R8 and R9, each 10k Ω) are attached to the input pins to guarantee steady functioning. When the buttons are not pressed, these resistors maintain the pins at a high voltage level (logic HIGH), which stops the pins from floating. Because floating pins may take up stray signals, they can cause the microcontroller to behave in an unanticipated way. Digital circuits commonly use pull-up resistors, whose value (usually 10k Ω) guarantees a balance between signal integrity and power consumption [2].

Resistors R1-R7

To prevent the 7-segment display from frying, there are the 7 resistors (R1-R7) each connected in series and each holding a value of 220 Ω . LEDs have a forward voltage drop (typically 2-3V) and must be driven with a limited current to avoid overheating [2]. With a 5V supply, the LED is supplied with around 15mA of current, which is perfectly safe.

7-Segment Display

The seven LEDs that make up the seven-segment display are placed in the letters A through G to produce a number. The microcontroller can display the numbers 0–9 by selectively turning these segments on and off. A single display segment is controlled by each of the RB0–RB7 output pins on the microcontroller.

Because of their cost, convenience of use, and compatibility with microcontrollers, seven-segment displays are widely employed for numerical output in digital systems. The display in this design has a common cathode configuration, where all cathodes are grounded and the anodes are connected to the microcontroller via current-limiting resistors.

Section 2

To show the hexadecimal characters 0 through F on a 7-segment display, the circuit utilises a counter to run over the hexadecimal values and a lookup table to translate each value into the correct 7-segment binary pattern. The 7-segment display is then driven by the PIC16F84A microcontroller, which receives this binary pattern on its PORTB pins.

The pseudocode below outlines the steps required to display the hexadecimal values 0 through to F:

Initialize PORTB as output

Set count = 0

// Main loop

while true:

 binary_pattern = [count] // Retrieve binary pattern for current counter value

 output PORTB = binary_pattern // Send pattern to 7-segment display

 delay() // Wait before updating the display

 count = count + 1 // Increment the counter

 if count == 16: // Reset counter after reaching hexadecimal 16 (0x10)

 count = 0

end while

// Function: Binary pattern lookup table

function lookupTable(value):

 patterns = [0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
 0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E]

 return patterns[value]

The 7-segment display employs 8 bits to determine which segments light up. Each bit in the binary pattern represents a distinct segment:

- Bit 6: Segment G
- Bit 5: Segment F
- Bit 4: Segment E
- Bit 3: Segment D
- Bit 2: Segment C
- Bit 1: Segment B
- Bit 0: Segment A

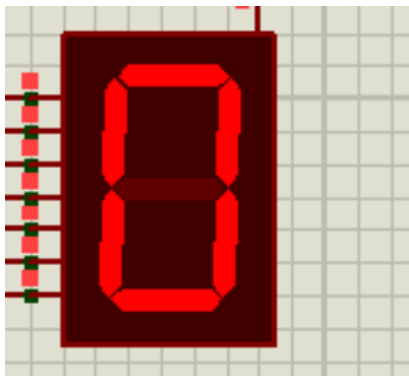


Figure 2: The binary pattern 11000000 lights up segments A, B, C, D, E, F and leaves G off. This results in the LED displaying '0'.

Hex	Binary Pattern	Segments Lit
0	11000000	ABCDEF
1	11111001	BC
2	10100100	ABCDEG
3	10110000	ABCDG
4	10011001	BCFG
5	10010010	ACDFG
6	10000010	ACDEFG
7	11111000	ABC
8	10000000	ABCDEFG
9	10010000	ABCDFG
A	10001000	ABCEFG
b	10000011	CDEFG
C	11000110	ADEF
d	10100001	BCDEG
E	10000110	ADEFG
F	10001110	AEFG

Table 1: The binary patterns and their corresponding segments for hexadecimal values 0 to F.

Section 3

Task 1

```
=====
; Main.asm file for counter (0-F), incrementing in 0.2 seconds
; Processor: PIC16F84A
=====
#include p16f84a.inc

; Define registers
portB equ 0x06
trisb equ 0x06
status equ 0x03
RP0 equ 5

; Variables
count equ 0x0c          ; Counter value
nCount equ 0x0d
mCount equ 0x0e

; Reset vector
RST code 0x0
    goto Start

; Main program
PGM code
Start
    ; Setup ports
    bsf status,RP0      ; Bank 1
    movlw 0x00
    movwf trisb         ; PORTB = outputs
    bcf status,RP0      ; Bank 0

    ; Initialize
    clrf count          ; Start counter at 0

main_loop
    movf count,W         ; Get current count
    call table           ; Convert to display pattern
    movwf portB          ; Output to display
```

call pause

```
incf count,F      ; Increment counter
movlw 0x10        ; Load 16 (0x10)
subwf count,W     ; Check if count = 16
btfsc STATUS,Z   ; Skip if not 16
clrf count        ; Reset to 0 if reached 16
```

goto main_loop ; Repeat

pause

```
movlw 0xc8        ; Load outer loop count (200 decimal)
movwf mCount      ; Initialize outer loop counter
```

loadN

```
movlw 0xc8        ; Load inner loop count (200 decimal)
movwf nCount      ; Initialize inner loop counter
```

decN

```
decfsz nCount,f   ; Decrement inner counter, skip if zero
goto decN         ; Repeat inner loop if not zero
```

```
decfsz mCount,f   ; Decrement outer counter, skip if zero
goto loadN        ; Repeat outer loop if not zero
```

return

; Display patterns for 7-segment (0-F)

table

```
addwf PCL
retlw 0xC0        ; 0
retlw 0xF9        ; 1
retlw 0xA4        ; 2
retlw 0xB0        ; 3
retlw 0x99        ; 4
retlw 0x92        ; 5
retlw 0x82        ; 6
retlw 0xF8        ; 7
retlw 0x80        ; 8
retlw 0x90        ; 9
retlw 0x88        ; A
retlw 0x83        ; b
```

```

    retlw 0xC6        ; C
    retlw 0xA1        ; d
    retlw 0x86        ; E
    retlw 0x8E        ; F

```

```

END

```

Task 2

```

;=====
; Main.asm file which when you press one of the two buttons, it displays "L" on the 7
; segment LED display, then when you press the other one, it displays "r".
; Processor: PIC16F84A
;=====

#include p16f84a.inc

; Define registers
portB equ 0x06
trisb equ 0x06
portA equ 0x05
trisa equ 0x05
status equ 0x03
RP0 equ 5

; Variables
count equ 0x0c        ; Counter value
lastState equ 0x0d     ; Previous button state
dCount equ 0x0e        ; For delay

; Reset vector
RST    code 0x0
    goto Start

; Main program
PGM    code
Start
    ; Setup ports
    bsf status,RP0      ; Bank 1
    movlw 0x00
    movwf trisb          ; PORTB = outputs

```

```
movlw 0x0f
movwf trisa      ; PORTA lower = inputs
bcf status,RPO   ; Bank 0
```

```
; Initialize
clrf count       ; Start counter at 0
movlw 0xFF
movwf lastState  ; Initialize button state
```

main_loop

```
movf portA,W      ; Get current button state
xorwf lastState,W  ; Compare with last state
andlw 0x03        ; Only care about first two buttons
btfsc STATUS,Z    ; Skip if buttons changed
goto display      ; No change, just update display
```

```
movf portA,W      ; Get button state again
movwf lastState   ; Save for next time
```

```
btfss portA,0     ; Skip if left button pressed
call left_button
```

```
btfss portA,1     ; Skip if right button pressed
call right_button
```

display

```
movf count,W      ; Get current count
call table        ; Convert to display pattern
movwf portB       ; Output to display
call delay        ; Small delay
goto main_loop    ; Repeat
```

; Increment counter (max 1 or r)

right_button

```
subwf count,W     ; Check if count = 9
btfsc STATUS,Z
return            ; At max, don't increment
incf count,F      ; Increment
return
```

; Decrement counter (min 0 or L)

```

left_button
    movf count,W          ; Check if count = 0
    btfsc STATUS,Z
    return                ; At min, don't decrement
    decf count,F          ; Decrement
    return

; Delay for debounce
delay
    movlw 0xff
    movwf dCount
delay_loop
    decfsz dCount,F
    goto delay_loop
    return

; Display patterns for 7-segment (L and r)
table
    addwf PCL
    retlw 0xC7            ; L
    retlw 0xAF            ; r

END

```

Task 3

```

;=====
; Main.asm file for on/off
; Processor: PIC16F84A
;=====

#include p16f84a.inc

; Define registers
portB equ 0x06
trisb equ 0x06
portA equ 0x05
trisa equ 0x05
status equ 0x03
RP0 equ 5

```

```

; Variables
count equ 0x0c          ; Counter value
lastState equ 0x0d       ; Previous button state
dCount equ 0x0e          ; For delay

; Reset vector
RST code 0x0
    goto Start

; Main program
PGM code
Start
    ; Setup ports
    bsf status,RP0        ; Bank 1
    movlw 0x00
    movwf trisb            ; PORTB = outputs
    movlw 0x0f
    movwf trisa            ; PORTA lower = inputs
    bcf status,RP0        ; Bank 0

    ; Initialize
    clrf count             ; Start counter at 0
    movlw 0xFF
    movwf lastState        ; Initialize button state

main_loop
    movf portA,W           ; Get current button state
    xorwf lastState,W       ; Compare with last state
    andlw 0x03             ; Only care about first two buttons
    btfsc STATUS,Z         ; Skip if buttons changed
    goto display           ; No change, just update display

    movf portA,W           ; Get button state again
    movwf lastState        ; Save for next time

    btfss portA,0          ; Skip if left button pressed
    call left_button

    btfss portA,1          ; Skip if right button pressed
    call right_button

```

```

display
    movf count,W          ; Get current count
    call table            ; Convert to display pattern
    movwf portB           ; Output to display
    call delay            ; Small delay
    goto main_loop        ; Repeat

```

; Increment counter (max 1)

```

right_button
    subwf count,W          ; Check if count = 1
    btfsc STATUS,Z
    return                ; At max, don't increment
    incf count,F           ; Increment
    return

```

; Decrement counter (min 0)

```

left_button
    movf count,W          ; Check if count = 0
    btfsc STATUS,Z
    return                ; At min, don't decrement
    decf count,F           ; Decrement
    return

```

; Delay for debounce

```

delay
    movlw 0xff
    movwf dCount
delay_loop
    decfsz dCount,F
    goto delay_loop
    return

```

; Display patterns for 7-segment (on and off)

```

table
    addwf PCL
    retlw 0x00            ; lights on
    retlw 0xFF           ; lights off

```

END

Task 4

```
=====
; Main.asm file for up/down counter (0-9) starting at 5
; Processor: PIC16F84A
=====

#include p16f84a.inc

; Define registers
portB equ 0x06
trisb equ 0x06
portA equ 0x05
trisa equ 0x05
status equ 0x03
RP0 equ 5

; Variables
count equ 0x0c      ; Counter value
lastState equ 0x0d   ; Previous button state
dCount equ 0x0e      ; For delay

; Reset vector
RST code 0x0
    goto Start

; Main program
PGM code
Start
    ; Setup ports
    bsf status,RP0    ; Bank 1
    movlw 0x00
    movwf trisb        ; PORTB = outputs
    movlw 0x0f
    movwf trisa        ; PORTA lower = inputs
    bcf status,RP0     ; Bank 0

    ; Initialize
    movlw .5           ; Changed from clrf to movlw 5 - start at 5
    movwf count        ; Initialize counter to 5
    movlw 0xFF
```

```
movwf lastState      ; Initialize button state
```

```
main_loop
```

```
movf portA,W          ; Get current button state
xorwf lastState,W      ; Compare with last state
andlw 0x03             ; Only care about first two buttons
btfsc STATUS,Z         ; Skip if buttons changed
goto display          ; No change, just update display
```

```
movf portA,W          ; Get button state again
movwf lastState        ; Save for next time
```

```
btfss portA,0         ; Skip if left button pressed
call left_button
```

```
btfss portA,1         ; Skip if right button pressed
call right_button
```

```
display
```

```
movf count,W          ; Get current count
call table             ; Convert to display pattern
movwf portB           ; Output to display
call delay             ; Small delay
goto main_loop        ; Repeat
```

```
; Increment counter (max 9)
```

```
right_button
```

```
subwf count,W          ; Check if count = 9
btfsc STATUS,Z
return                 ; At max, don't increment
incf count,F           ; Increment
return
```

```
; Decrement counter (min 0)
```

```
left_button
```

```
movf count,W          ; Check if count = 0
btfsc STATUS,Z
return                 ; At min, don't decrement
decf count,F           ; Decrement
return
```

```

; Delay for debounce
delay
    movlw 0xff
    movwf dCount
delay_loop
    decfsz dCount,F
    goto delay_loop
    return

; Display patterns for 7-segment (0-9)
table
    addwf PCL
    retlw 0xC0        ; 0
    retlw 0xF9        ; 1
    retlw 0xA4        ; 2
    retlw 0xB0        ; 3
    retlw 0x99        ; 4
    retlw 0x92        ; 5
    retlw 0x82        ; 6
    retlw 0xF8        ; 7
    retlw 0x80        ; 8
    retlw 0x90        ; 9

    END

```

Task 5

```

;=====
; Main.asm file for piezo sounder
; Processor: PIC16F84A
;=====
#include p16f84a.inc

; Define registers
portB equ 0x06
trisb equ 0x06
portA equ 0x05
trisa equ 0x05
status equ 0x03
RP0 equ 5

```

```

; Variables
toneCount equ 0x0c ; Counter for tone generation
lastState equ 0x0d ; Previous button state

; Reset vector
RST code 0x0
    goto Start

; Main program
PGM code
Start
    ; Setup ports
    bsf status,RP0 ; Bank 1
    movlw 0x00
    movwf trisb ; PORTB = outputs (for piezo)
    movlw 0x0f
    movwf trisa ; PORTA lower = inputs (for buttons)
    bcf status,RP0 ; Bank 0

    ; Initialize
    movlw 0xFF
    movwf lastState ; Initialize button state
    clrf portB ; Clear output

main_loop
    movf portA,W ; Get current button state
    andlw 0x03 ; Mask for first two buttons
    xorlw 0x03 ; XOR with 0x03 (buttons are active low)
    btfsc STATUS,Z ; Skip if any button pressed
    clrf portB ; Turn off piezo

    ; Generate tone
    comf portB,F ; Toggle piezo output
    call tone_delay ; Delay for frequency
    goto main_loop

; Delay for tone generation
tone_delay
    movlw 0x50 ; tone frequency

```

```
    movwf toneCount
delay_loop
    decfsz toneCount,F
    goto delay_loop
return

END
```

References

- [1] Microchip Technology, "PIC16F84A | Microchip Technology," 2013. [Online]. Available: <https://www.microchip.com/en-us/product/PIC16F84A>.
- [2] M. Parks, "Don't Burn Out! Calculating LED Current Limiting Resistor," Mouse Electronics, 8 April 2014. [Online]. Available: <https://www.mouser.com/blog/dont-burn-out-calculating-led-current-limiting-resistor>.
- [3] Sparkfun, "Pull-up Resistors," Sparkfun, [Online]. Available: <https://learn.sparkfun.com/tutorials/pull-up-resistors/all>.