Journal of the Nigerian Association of Mathematical Physics
***Volume 33*, (*January, 2016*), pp 229 – 240**

Operating A PIC16F877 Microcontroller-Based Timing System

Auwal Mustapha Imam

Department of Physics with Electronics, Federal University Birnin Kebbi, Nigeria.

Abstract

Electronic clocks have predominantly replaced the mechanical clocks. They are much reliable, accurate, maintenance free and portable. Changes in time keeping technology have influenced the character of scientific observation, aided the development of other machine technologies and brought significant revisions in the way people think about and behave in time. Unlike other clocks with 555 timers or other digital control circuitry, this system uses PIC microcontroller which is a more advanced design, so unique and different from all other designs. The codes are written on MPLAB programming environment and programmed on the microcontroller. The PIC16F877 accepts a low frequency crystal, which must be added externally. Upon initializing the microcontroller, the clock system must be configured to take advantage of this clock. The speed of instruction execution will depend on the clock. The PIC16F877 microcontroller is manned on the hardware of the clock. The microcontroller executes the instructions and display the resulting time on the four Seven-Segment (7-Seg) displays.

Keywords: Clock, Codes, Electronics, Instructions, Microcontroller, PIC16F877, Program

1.0 Introduction

Time is such a fundamental concept that is very difficult to repeat itself at regular intervals. The number of intervals counted gives a quantitative measure of the duration. The earliest references for the measurement of the time were moon and sun. When the sun and the moon were not visible, it was impossible to know the exact time. So, clocks were developed to measure out the hours between checks with the sun and the moon. The process of measuring time has progressively become more accurate. Many centuries have been spent devising method for the determination and measurement of time. Historically, clocks and watches of all sorts lie at an important crossroads of science, technology and society. Changes in time keeping technology have influenced the character of scientific observation, aided the development of other machine technologies and brought significant revisions in the way people think about and behave in time [1].

Electronic clocks have predominantly replaced the mechanical clocks. They are much reliable, accurate, maintenance free and portable. In general, there are two kinds of electronic clocks. They are analog clock and digital clock. But digital clocks are more common and independent of external source. Although peripherals do consume current, the CPU, when running, is in most cases the major offender. Current consumption usually varies linearly with clock speed and therefore one way to keep consumption to a minimum is to set the clock speed as low as possible (or turn it off completely when not needed). Microcontrollers generally use two categories of clocks, fast and slow [2]. The fast clocks source the CPU and most modules and vary usually from several hundred KHz to Several MHz.

There is strong need for communication between the user and the microcontroller. But the problem is, microcontroller doesn't understand our language. So there is need to generate codes (instruction sets) that the microcontroller understands, the codes are programmed on the microcontroller so that upon initialization of the clock, the microcontroller will begin to execute the instructions[3].

The circuit of Figure 1 was implemented with PIC16F877 microcontroller to execute the codes and instructions.

Corresponding author: Auwal Mustapha Imam, E-mail: mustaphaimamauwal@gmail.com, Tel.: +2348034647088

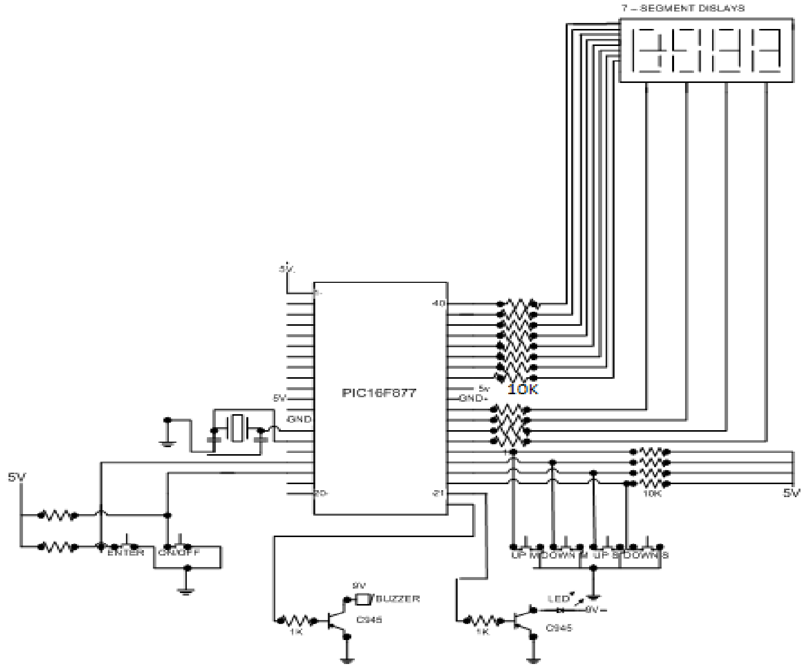

Figure 1: PIC16F877 Microcontroller-Based Timing System

2.0 Communication to a PIC16F877 Microcontroller and Instruction Set

The ability to communicate is of great importance. It is only possible if both communication partners know the same language (follow the same rules during communication). The communication between man and microcontroller is defined. The language that man and microcontrollers communicate is called "assembly language". Programs written in assembly language must be translated into '0s' and '1s' in order for the microcontroller to understand it [4]. A program is written according to the rules of the assembler to suit the desired effect. A translator interprets each instruction written as a series of '0s' and '1s' which have a meaning for the internal logic of a microcontroller. The process of communication between man and a microcontroller is illustrated in the Figure 2.

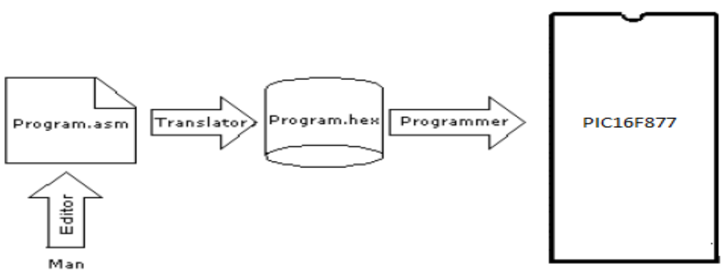

Figure 2: Process of communication between man and a microcontroller.

3.0 Instruction Sets

Each PIC16F877 instruction is a 14-bit word, divided into an OPCODE which specifies the instruction type and one or more operands which further specify the operation of the instruction. The PIC16F877 instruction set summary in the table below lists mnemonics operands with their description, byte-oriented, bit-oriented, and literal and control operations. For byte-oriented instructions, 'f' represents a file register designator and'd' represents a destination designator[5]. The file register designator specifies which file register is to be used by the instruction. The destination designator specifies where the result of the operation is to be placed. If'd' is zero, the result is placed in the W register. If'd' is one, the result is placed in the file register specified in the instruction.

For bit-oriented instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the address of the file in which the bit is located. For literal and control operations, 'k' represents an eight or eleven bit constant or literal value.

All instructions are executed within one single instruction cycle, unless a conditional test is true or the program counter is changed as a result of an instruction. In this case, the execution takes two instruction cycles with the second cycle executed as a NOP. One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 µs. If a conditional test is true, or the program counter is changed as a result of an instruction, the instruction execution time is 2 µs.

Table 1: PIC16F877 Instruction set [6]

Mnemonic, Operands		Description		14-Bit Opcode			
				MSb			LSb
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f		00	0111	dfff	ffff
ANDWF	f, d	AND W with f		00	0101	dfff	ffff
CLRF	f	Clear f		00	0001	1fff	ffff
CLRW	-	Clear W		00	0001	0xxx	xxxx
COMF	f, d	Complement f		00	1001	dfff	ffff
DECF	f, d	Decrement f		00	0011	dfff	ffff
DECFSZ	f, d	Decrement f, Skip if 0		00	1011	dfff	ffff
INCF	f, d	Increment f		00	1010	dfff	ffff
INCFSZ	f, d	Increment f, Skip if 0		00	1111	dfff	ffff
IORWF	f, d	Inclusive OR W with f		00	0100	dfff	ffff
MOVF	f, d	Move f		00	1000	dfff	ffff
MOVWF	f	Move W to f		00	0000	1fff	ffff
NOP	-	No Operation		00	0000	0xx0	0000
RLF	f, d	Rotate Left f through Carry		00	1101	dfff	ffff
RRF	f, d	Rotate Right f through Carry		00	1100	dfff	ffff
SUBWF	f, d	Subtract W from f		00	0010	dfff	ffff
SWAPF	f, d	Swap nibbles in f		00	1110	dfff	ffff
XORWF	f, d	Exclusive OR W with f		00	0110	dfff	ffff
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f		01	00bb	bfff	ffff
BSF	f, b	Bit Set f		01	01bb	bfff	ffff
BTFSC	f, b	Bit Test f, Skip if Clear		01	10bb	bfff	ffff
BTFSS	f, b	Bit Test f, Skip if Set		01	11bb	bfff	ffff
LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add literal and W		11	111x	kkkk	kkkk
ANDLW	k	AND literal with W		11	1001	kkkk	kkkk
CALL	k	Call subroutine		10	0kkk	kkkk	kkkk
CLRWDT	-	Clear Watchdog Timer		00	0000	0110	0100
GOTO	k	Go to address		10	1kkk	kkkk	kkkk
IORLW	k	Inclusive OR literal with W		11	1000	kkkk	kkkk
MOVLW	k	Move literal to W		11	00xx	kkkk	kkkk
RETFIE	-	Return from interrupt		00	0000	0000	1001
RETLW	k	Return with literal in W		11	01xx	kkkk	kkkk
RETURN	-	Return from Subroutine		00	0000	0000	1000
SLEEP	-	Go into standby mode		00	0000	0110	0011
SUBLW	k	Subtract W from literal		11	110x	kkkk	kkkk
XORLW	k	Exclusive OR literal with W		11	1010	kkkk	kkkk

4.0 Operation Procedures

a. Preset Procedure

The flowchart of Figure 3 describes the time preset procedures. The description of how the push-button switches for setting the time is elaborated. The responds to the procedures and execute them.

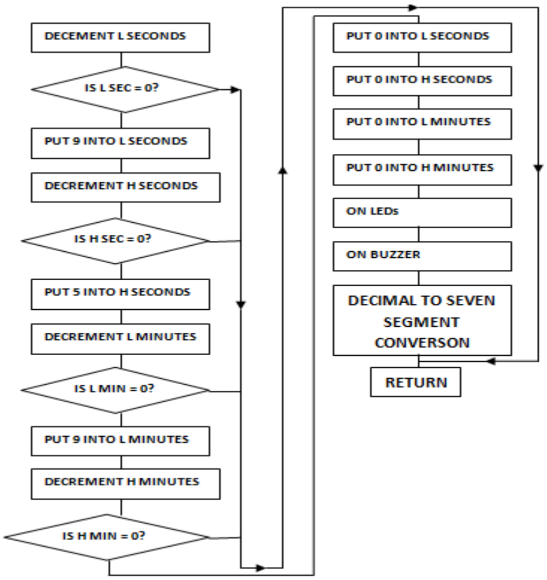

Figure 3: Time update sub-routine flowchart

b. Display sub-routine flowchart

Whenever a particular time is set, the program is written in such a way that a microcontroller will attend a sub-routine and pick a particular instruction to be displayed. All the LEDs in the four 7-SEG displays are attached to a particular instruction to set them according to desire. The flowchart of Figure 4 illustrates the steps and procedures for display on the 7-SEG displays.

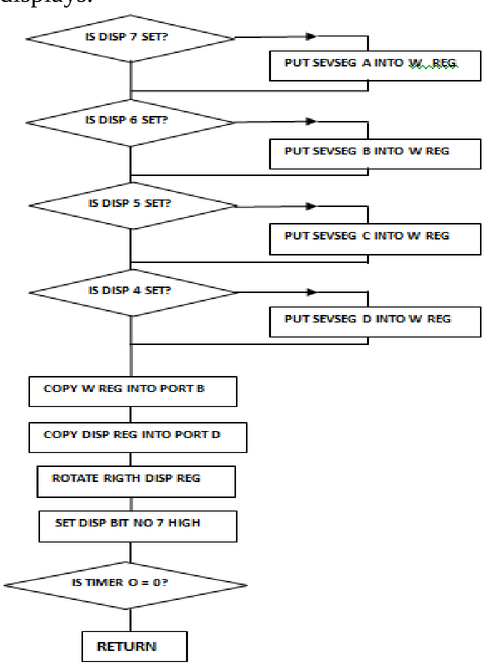

Figure 4: Display sub-routine flowchart

5.0 Timing System Operation Codes

As discussed earlier, there is need for communication between man and the microcontroller. But the two doesn't understand the language of each other. Codes are generated, which are translated and compiled by some components of the microcontroller. These codes are the programming languages. In this system, C language codes written on MPLAB programming environment are used for communication to the microcontroller to execute the desired actions according to the instructions contained in the codes.

```
;*****************************************************************************
;    Filename:          Advancetimingsys.asm                                    *
;    Date:         14th December, 2015                  *
;    File Version:  13/04                                              *
;    Author:       Auwal Mustapha Imam
;    Institution:   Federal university Birnin Kebbi         *
;*************************************************************************
;    Files required: list and include pic16f877A                *                                        *
;*************************************************************************
;    Notes: The following program is for Advanced Timing System
;          *
;          *
;        REMEMBER=the radix is in hex,                        *
;*************************************************************************
;======================================================================
        #INCLUDE "P16F877A.inc"
 __CONFIG     (_CP_ALL &_DEBUG_OFF &_WRT_OFF &_LVP_OFF & _BODEN_OFF &_WDT_OFF &_XT_OSC)
        RADIX     HEX
;======================================================================
```

; (a). Initialization and Allocation of Components to Registers and execution

; The input and output components of the system are assigned some special function registers. The components are addressed to the registers. The program mostly begins by this assignmentand initialization.

```
BEGIN
HDOT      EQU             0X00                          ; HDOT is allocated 0X00 register
CNTR      EQU             0X20                          ;CNTR goes to 0X20 register
CNTR1     EQU             0X21
CNTR2     EQU             0X22
FLAGS     EQU             0X23
SEVSEG_A   EQU     0X24
SEVSEG_B   EQU     0X25
SEVSEG_C   EQU     0X26                ;7-Seg C act based on the instruction in 0X26 register
SEVSEG_D   EQU     0X27
DISPLAY    EQU     0X28
SEC_NTH    EQU     0X29

L_SECOND   EQU     0X2A
H_SECOND   EQU     0X2B
L_MINUTE   EQU     0X2C
H_MINUTE   EQU     0X2D
INPUTS     EQU     0X2E                             ;AnyINPUT goes to 0X2E register
```

; (b).Definition and allocation of ports to the components and execution

; The components need to be defined and allocated to the ports of the microcontroller for effective addressing. The program below summarizes the addressing of the ports.

```
#DEFINE    ENTERBUT       PORTC, 2   ; ENTER BUTTON is put in port C of the MC
#DEFINE    POWERBUT       PORTC, 3
#DEFINE    DOWNSECBUT     PORTC, 4
#DEFINE    UPSECBUT       PORTC, 5
#DEFINE    DOWNMINBUT     PORTC, 6
#DEFINE    UPMINBUT       PORTC, 7

#DEFINE    SEGOUT_A       PORTD, 7
#DEFINE    SEGOUT_B       PORTD, 6
#DEFINE    SEGOUT_C       PORTD, 5
#DEFINE    SEGOUT_D       PORTD, 4
#DEFINE    LEDOUT         PORTA, 1
```

```
#DEFINE    BUZZOUT         PORTA, 0
#DEFINE    OVERFLOWFLAG      FLAGS, 6
#DEFINE    POWSTATEFLAG      FLAGS, 6
#DEFINE    ENTERFLAG        FLAGS, 5
#DEFINE    POWERFLAG        FLAGS, 4
#DEFINE    DOWNSECFLAG      FLAGS, 3
#DEFINE    UPSECFLAG        FLAGS, 2
#DEFINE    DOWNMINFLAG      FLAGS, 1
#DEFINE    UPMINFLAG        FLAGS, 0

        ORG      0X00                                ; Origin of the program is 0X00

        GOTO     START                       ; Return to the beginning of the program
```

; (c).Routine for selection of memory registers and execution

Below is the macro routine for the selection of memory registers on the microcontroller. The registers are named BANK, and the programs are addressed to a particular register to pick the instruction for execution.

```
BANK0     MACRO              ; bank 0 select macro routine
        BCF      STATUS, RP0
        BCF      STATUS, RP1
        ENDM
BANK1     MACRO              ; bank 1 select macro routine
        BSF      STATUS, RP0
        BCF      STATUS, RP1
        ENDM
BANK2     MACRO              ; bank 2 select macro routine
        BCF      STATUS, RP0
        BSF      STATUS, RP1
        ENDM
BANK3     MACRO              ; bank 0 select macro routine
        BSF      STATUS, RP0
        BSF      STATUS, RP1
        ENDM
```

; (d). Initialisation of the special function registers and execution of instructions

```
START                    BANK0;========this is where the special function registers are initialized================================================================================
================================================
       CLRF     PORTB         ; initialize PORTB by clearing output
       CLRF     PORTC         ; initialize PORTC by clearing output
       CLRF     PORTD         ; initialize PORTD by clearing output
       BANK1
       MOVLW    B'00000110'    ; set all input as digital not analog
       MOVWF    ADCON1
       MOVLW    B'00000011'    ; Prescale RTCC, 1:16
       MOVWF    OPTION_REG     ; set option register, transition on clock,
       MOVLW    B'11111111'    ; all PORTA pins as input
       MOVWF    TRISC
       MOVLW    B'11111100'
       MOVWF    TRISA
       MOVLW    B'00000000'    ; PORTB, PORTC and PORTD as outputs
       MOVWF    TRISD
       MOVWF    TRISB
       BANK0
```

```
        MOVLW     H'01'
            MOVWF              TMR0 ; set RTCC above zero so initial wait period occurs

        MOVLW     B'10000000'     ; set display switching
            MOVWF              DISPLAY
        MOVLW     0X01           ; put 60 seconds into display
        MOVWF     L_SECONDS
        MOVLW     0X01
        MOVWF     H_SECONDS
        MOVLW     0X01
        MOVWF     L_MINUTES
        MOVLW     0X07
        MOVWF     H_MINUTES
   GOTO      POWSUB
```

;(e). Sub-routine table for the representation of Os and 1s for ON/OFF of the LEDs in the display and execution
; This sub-routine shows the table for the representations of the LEDs on the 7-SEG display for the indication of the numbers 0 -9 on the display. 0 represents ON while 1 represents OFF.
;------------------------------------

```
TABLE     ADDWF      PCL, F
        RETLW      B'00000000'   ;--
        RETLW      B'00000011'   ; 0
        RETLW      B'10011111'   ; 1
        RETLW      B'00100101'   ; 2
        RETLW      B'00001101'   ; 3
        RETLW      B'10011001'   ; 4
        RETLW      B'01001001'   ; 5
        RETLW      B'01000001'   ; 6
        RETLW      B'00011111'   ; 7
        RETLW      B'00000001'   ; 8
        RETLW      B'00001001'   ; 9

        END
```

GO TO MAIN PROGRAM

;(f). ===================================this is a MACRO routine, which select BANK0,1,2and3==
==

```
BANK0     MACRO              ; bank 0 select macro routine
        BCF      STATUS, RP0
        BCF      STATUS, RP1
        ENDM
BANK1     MACRO              ; bank 1 select macro routine
        BSF      STATUS, RP0
        BCF      STATUS, RP1
        ENDM
BANK2     MACRO              ; bank 2 select macro routine
        BCF      STATUS, RP0
        BSF      STATUS, RP1
        ENDM
BANK3     MACRO                ; bank 0 select macro routine
        BSF      STATUS, RP0
        BSF      STATUS, RP1
        ENDM
```

```
START                BANK0;========this is where the special function registers are initialized==================================================================================================================================
        CLRF     PORTB         ; initialize PORTB by clearing output
        CLRF     PORTC         ; initialize PORTC by clearing output
        CLRF     PORTD         ; initialize PORTD by clearing output
        BANK1
        MOVLW    B'00000110'    ; set all input as digital not analog
        MOVWF    ADCON1
        MOVLW    B'00000011'    ; Prescale RTCC, 1:16
        MOVWF    OPTION_REG     ; set option register, transition on clock,
        MOVLW    B'11111111'    ; all PORTA pins as input
        MOVWF    TRISC
        MOVLW    B'11111100'
        MOVWF    TRISA
        MOVLW    B'00000000'    ; PORTB, PORTC and PORTD as outputs
        MOVWF    TRISD
        MOVWF    TRISB
        BANK0
        MOVLW    H'01'
        MOVWF      TMR0 ; set RTCC above zero so initial wait periods occurs

        MOVLW    B'10000000'    ; set display switching
            MOVWF           DISPLAY
        MOVLW    0X01          ; put 60 seconds into display
        MOVWF    L_SECONDS
        MOVLW    0X01
        MOVWF    H_SECONDS
        MOVLW    0X01
        MOVWF    L_MINUTES
        MOVLW    0X07
        MOVWF    H_MINUTES

        GOTO     POWSUB
;------------------------------------------------------------------------------------------------------------------
DEBOUNCE

        MOVLW    .10;;;2
        MOVWF    CNTR2
DEL1002 MOVLW    .65
        MOVWF    CNTR1
DEL1001 MOVLW    .255
        MOVWF    CNTR
        CLRWDT
DEL100  DECFSZ   CNTR,F
        GOTO     DEL100
        DECFSZ   CNTR1,F
        GOTO     DEL1001
        DECFSZ   CNTR2,F
        GOTO     DEL1002
        NOP
        RETURN
;------------------------------------------------
POWSUB    CLRWDT
         MOVLW   B'11111111'
         MOVWF   PORTB
         BTFSC   POWERBUT; if power SWITCH is pressed then
```

```
        GOTO    POWSUB          ; go and on display
        CALL    DEBOUNCE
        GOTO    POWON
DISP
        BTFSC   DISPLAY, 7      ; if display bit7 is low then
        MOVF    SEVSEG_A, W     ; display into the first 7 segment LED
        BTFSC   DISPLAY, 6      ; if display bit6 is low then
        MOVF    SEVSEG_B, W     ; display into the second 7 segment LED
        BTFSC   DISPLAY, 5      ; if display bit5 is low then
        MOVF    SEVSEG_C, W     ; display into the third 7 segment LED
        BTFSC   DISPLAY, 4      ; if display bit4 is low then
        MOVF    SEVSEG_D, W     ; display into the fourth 7 segment LED
        MOVWF   PORTB
        ; BTFSS   SEC_NTH,7
        ; BSF     PORTB,0
        MOVF    DISPLAY, W
        MOVWF   PORTD
        RRF     DISPLAY, F
        BCF     DISPLAY, 7

        BTFSC   DISPLAY, 3      ; if the fourth LED is displayed then
        BSF     DISPLAY, 7      ; make first led ready to display
        GOTO    RTCC_FILL

RTCC_FILL
        CLRWDT
        MOVF        TMR0, W
        BTFSS   STATUS, Z; note, RTCC is left free running to not lose clock cycles on writes
        GOTO    RTCC_FILL; DISP
        RETURN

TIME_UPDATE
        DECFSZ L_SECONDS, F
        GOTO    TU1
        MOVLW   .10
        MOVWF   L_SECONDS

        DECFSZ H_SECONDS, F
        GOTO    TU2
        MOVLW   .6
        MOVWF   H_SECONDS

        DECFSZ L_MINUTES, F
        GOTO    TU3
        MOVLW   .10
        MOVWF   L_MINUTES

        DECFSZ H_MINUTES, F
        GOTO    TU4
        MOVLW   .1
        MOVWF   L_SECONDS       ; time is zero, put zeros in all segments
        MOVWF   H_SECONDS
        MOVWF   L_MINUTES
        MOVWF   H_MINUTES
        BSF     LEDOUT
        BSF     BUZZOUT
```

```
TU1     NOP
        GOTO    HEXCON
TU2     NOP
        GOTO    HEXCON
TU3     NOP
        GOTO    HEXCON
TU4     NOP

HEXCON
        MOVF    L_SECONDS, W
        CALL    TABLE
        MOVWF   SEVSEG_D
        MOVF    H_SECONDS, W
        CALL    TABLE
        MOVWF   SEVSEG_C
        MOVF    L_MINUTES, W
        CALL    TABLE
        MOVWF   SEVSEG_B
        MOVF    H_MINUTES, W
        CALL    TABLE
        MOVWF   SEVSEG_A
        RETURN

; ----------POWER ON-----------------------------------------
POWON
MAIN
        MOVLW   .116;;;;; 12
        MOVWF   SEC_NTH
;------------------------------------------------
MAINLOOP
        NOP
SECIN   CALL    DISP        ; send output to display
        INCFSZ SEC_NTH, F
        GOTO    MAINLOOP
        MOVLW   .12
        MOVWF   SEC_NTH
        CALL    TIME_UPDATE     ; call and update time subroutine
        NOP
        BTFSS   ENTERBUT; if enter button is pressed then
        GOTO    SETTINGS        ; go to settings subroutine
        NOP
        BTFSC   POWERBUT; if power SWITCH is pressed then
        GOTO    SECIN       ; go and on display
        CALL    DEBOUNCE
        NOP
        GOTO    POWSUB
        GOTO    MAINLOOP

SETTINGS   MOVLW    .116
        MOVWF    SEC_NTH
        CALL    DEBOUNCE
        BCF     LEDOUT
        BCF     BUZZOUT
STTLOOP
        CALL    DISP        ; send output to display
        INCFSZ SEC_NTH, F
        GOTO    STTLOOP
```

```
        MOVLW   .190        ; variable for button change speed
        MOVWF   SEC_NTH

CHECKUB   BTFSC   UPSECBUT      ; if second up button is not incremented then
        GOTO    CHECKDB       ; go and check second down button
        INCF    L_SECONDS, F    ; button pressed, increment seconds
        MOVF    L_SECONDS, W
        XORLW   .11
        BTFSS   STATUS, Z
        GOTO    SETEND
        MOVLW   .1
        MOVWF   L_SECONDS

        INCF    H_SECONDS, F
        MOVF    H_SECONDS, W
        XORLW   .7
        BTFSS   STATUS, Z
        GOTO    SETEND
        MOVLW   .6
        MOVWF   H_SECONDS
        MOVLW   .10
        MOVWF   L_SECONDS
        GOTO    SETEND

CHECKDB   BTFSC   DOWNSECBUT      ; if second down button is not incremented then
        GOTO    CHECKUM         ; go and check minutes down button
        DECFSZ L_SECONDS, F     ; button pressed, decrement seconds
        GOTO    SETEND
        MOVLW   .10
        MOVWF   L_SECONDS
        DECFSZ H_SECONDS, F
        GOTO    SETEND
        MOVLW   .1
        MOVWF   H_SECONDS
        MOVLW   .1
        MOVWF   L_SECONDS
        GOTO    SETEND

CHECKUM   BTFSC   UPMINBUT      ; if minutes up button are not incremented then
        GOTO    CHECKDM        ; go and check minutes down button
        INCF    L_MINUTES, F    ; button pressed, increment minutes
        MOVF    L_MINUTES, W
        XORLW   .11
        BTFSS   STATUS, Z
        GOTO    SETEND
        MOVLW   .1
        MOVWF   L_MINUTES

        INCF    H_MINUTES, F
        MOVF    H_MINUTES, W
        XORLW   .7
        BTFSS   STATUS, Z
        GOTO    SETEND
        MOVLW   .6
        MOVWF   H_MINUTES
        MOVLW   .10
        MOVWF   L_MINUTES
```

```
        GOTO    SETEND

CHECKDM    BTFSC   DOWNMINBUT       ; if a minute up button is not incremented then
        GOTO    SETEND          ; go and check minutes up button
        DECFSZ L_MINUTES, F      ; button pressed, decrement minutes
        GOTO    SETEND
        MOVLW   .10
        MOVWF   L_MINUTES

        DECFSZ H_MINUTES, F
        GOTO    SETEND
        MOVLW   .1
        MOVWF   H_MINUTES
        MOVLW   .1
        MOVWF   L_MINUTES
        GOTO    SETEND

SETEND     NOP

        BTFSC   ENTERBUT
        GOTO    STTLOOP
        CALL    DEBOUNCE
        GOTO    SECIN
;------------------------------------

       END
```

6.0 Conclusion

As opposed to fixed digital circuitry, microcontrollers can be programmed to perform many applications and can be later changed when improvement are required. This saves both time and money when a field upgrade is required. However, these codes can be altered to improve the operation of the microcontroller. This paper therefore presents a more advanced design and implementation of timing system with a different method and approach of operation.

7.0 References

[1] Pan, T. (2008), *Development and Implementation of Microcontroller-based Digital Clock,*Journal of Academy ofn Science and Technology, Vol III, Myanmar.

[2] Matic, N. (2003), *Microcontrollers for beginners*, Mikroelektronika, 184 – 253, USA.

[3] G. S. M Galadanci, *etal*, Design and Development of a Microcontroller-based Timing System, Bayero Journal of Physics and Mathematical Sciences, Volume 6, No. 1, August 2015.

[4] Aggarwal, M. *etal*. (2012) *Comparative Implementation of Automatic Car Parking System with least distance parking space in Wireless Sensor Networks*, International Journal of Scientific and Research Publications, Volume 2, Issue 10.

[5] Augarten, Stan. (2008), *Oral History Panel on the Development and Promotion of the Intel 8048 Microcontroller,* Computer History Museum, Oral History, Retrieved 2011.

[6] Microchip (2001), PIC16F87X Data Sheet, Microchip technology, USA.