



BENEMÉRITA UNIVERSIDAD
AUTÓNOMA DE PUEBLA



FACULTAD DE CIENCIAS DE LA ELECTRÓNICA

Prácticas con PIC18F4550

Materia: Microcontroladores

Profesor: Ricardo Álvarez González

Equipo : Flora González David Salvador

Flores Jiménez Fernando

Quintero Quintana Rebeca del Consuelo

Quispe Condori Hanan Ronaldo

Curso Semestral

Periodo Primavera 2020

CONTENIDO

PRÁCTICA 1 EFECTO DE CORRIMIENTO CON LED'S PIC18F4550.....	7
OBJETIVO	7
MARCO TEÓRICO	7
BLOQUES DEL PIC18F4550	8
DIRECTIVAS DEL LENGUAJE	9
MEMORIA.....	9
Special Function Registers (SFR)	10
Bits de configuración.....	10
OSCILADOR.....	12
DESARROLLO.....	13
Simulación y Ensamblaje del Circuito.....	13
Código Fuente	16
CONCLUSIONES	21
PRÁCTICA 2 EFECTOS DE ROTACIÓN CON LED'S PIC18F4550.....	22
OBJETIVO	22
MARCO TEÓRICO	22
Modos de direccionamiento	22
Direccionamiento Inherente y Literal	22
Direccionamiento Directo	22
Direccionamiento Indirecto	23
Puertos de Entradas/Salidas	24
Configuración de los Puertos	24
Lectura de los Puertos.....	25
Escritura de Puertos	25
DESARROLLO.....	26
Simulación y Ensamblaje del Circuito.....	26

.....	27
Código Fuente	29
CONCLUSIONES	34
PRÁCTICA 3 INTERRUPCIONES PIC18F4550.....	35
OBJETIVO	35
.....	35
MARCO TEÓRICO	35
Interrupciones.....	35
INSTRUCCIONES PBP PARA INTERRUPCIONES.....	38
REGISTRO INTCON.....	38
REGISTRO INTCON2.....	40
DESARROLLO.....	41
Simulación y Ensamblaje del Circuito.....	41
Código Fuente	44
CONCLUSIONES	50
PRÁCTICA 4 CONTADOR DEL 0 AL 9 EN DISPLAY CON PIC18F4550.....	51
OBJETIVO	51
MARCO TEÓRICO	51
Ánodo Común y Cátodo Común.....	51
Control de los Segmentos	52
Temporizador/contador TIMERO	54
Módulo Temporizador/contador TIMER2.....	55
Operación:.....	55
Temporizador 0 Interrupción	56
DESARROLLO.....	57
Simulación y Ensamblaje del Circuito.....	57
Código Fuente	59
CONCLUSIONES	62
PRÁCTICA 5 CONTADOR DE 0 A 9 ASCENDENTE Y DESCENDENTE CON TIMER PIC18F4550	63

OBJETIVO	63
MARCO TEÓRICO	63
Prioridad de las interrupciones:	63
Proceso de tratamiento de una interrupción:	64
Interrupciones del puerto B:	64
Interrupción externa 1:	65
Interrupción externa 2:	65
Interrupción por cambio en pines RB7...RB4:	65
Tablas Por El Método Del 'Goto Calculado'	67
DESARROLLO.....	68
Simulación y Ensamblaje del Circuito.....	68
Código Fuente	70
CONCLUSIONES	73
PRÁCTICA 6 CONTADOR DE 0 A 9999 CON TIMER EN PIC18F4550.....	74
OBJETIVO	74
MARCO TEÓRICO	74
Interrupción del Temporizador 0:	74
Temporización con el registro TMROL.....	74
Temporizador 1	75
Temporizador 2:	76
Tablas De Búsqueda En Memoria De Programa Usando Instrucciones De Lectura De Tabla	77
DESARROLLO.....	78
Simulación y Ensamblaje del Circuito.....	78
.....	79
Código Fuente	81
CONCLUSIONES	85
PRÁCTICA 7 RELOJ 24 HORAS, 4 DISPLAY	86
OBJETIVO	86
MARCO TEÓRICO	86

Temporizador 1	86
Interrupción del Temporizador 1:.....	86
DESARROLLO.....	87
Simulación y Ensamblaje del Circuito.....	88
Código Fuente	89
CONCLUSIONES.....	97
PRÁCTICA 8 GENERACIÓN DE SEÑAL PWM.....	98
OBJETIVO	98
MARCO TEÓRICO	98
Modulo CCP:.....	98
Timer 2	98
Ciclo de Trabajo (Duty Cycle)	98
DESARROLLO.....	99
Implementación del Circuito.....	100
Código Fuente	101
CONCLUSIONES.....	104
PRÁCTICA 9 ADC.....	105
OBJETIVO	105
MARCO TEÓRICO	105
Modulo ADC:	105
GO/DONE	105
Tiempo de Adquisición.....	105
Uso del Disparo Del CCP2.....	105
DESARROLLO.....	106
Implementación del Circuito.....	107
Código Fuente	108
CONCLUSIONES.....	110
PRÁCTICA 10 TECLADO MATRICIAL Y LCD.....	111
OBJETIVO	111

MARCO TEÓRICO	111
Ensamblador	111
Compilador	111
Librería kbd_4x4.c	112
DESARROLLO.....	112
Implementación del Circuito.....	114
Código Fuente	114
Librería Modificada	115
CONCLUSIONES	119
Práctica 11 Cerradura Electrónica.....	120
OBJETIVO	120
.....	120
MARCO TEÓRICO	120
Librería kbd_4x4.c	120
DESARROLLO.....	120
Implementación del Circuito.....	121
Código Fuente	122
CONCLUSIONES	125
PRÁCTICA 12 INTRODUCCION AL IOT	126
OBJETIVO	126
MARCO TEÓRICO	126
Node MCU	126
Ubidots	127
DESARROLLO.....	127
Implementación del Circuito.....	128
Código Fuente	130
CONCLUSIONES	131

PRÁCTICA 1 EFECTO DE CORRIMIENTO CON LED'S PIC18F4550

OBJETIVO

Realizar el corrimiento y rotación de leds utilizando el PIC 18F4550 por medio de un código lenguaje ensamblador en el programa de simulación MPLAB, encender secuencialmente cada uno de ellos, para lo cual declararemos dicho puerto como salida, escribiendo la palabra adecuada de configuración en su registro de dirección de datos TRISD, tomando en cuenta que al escribir un 1 en un bit del registro TRISD, configurará su bit de puerto asociado como entrada, en el caso de escribir un cero se configurará su bit de puerto respectivo como salida.

MARCO TEÓRICO

La disposición de los puertos digitales nos permite hacer operaciones con un byte, nibble o bit, por ejemplo, intercambiar el nibble alto por el bajo o desplazar el valor de un bit hacia la izquierda o derecha, algunas de estas operaciones están disponibles en las funciones de algunos PICs como el caso del 18F4550, gracias a estas funciones es posible crear efectos de luces con LEDs al colocar ciertas configuraciones de 0 y 1 a la salida de dichos puertos.

En el siguiente gráfico (Figura 1) se muestran los puertos disponibles en el PIC18F4550, sus ubicaciones y las posibles funciones de cada pin:

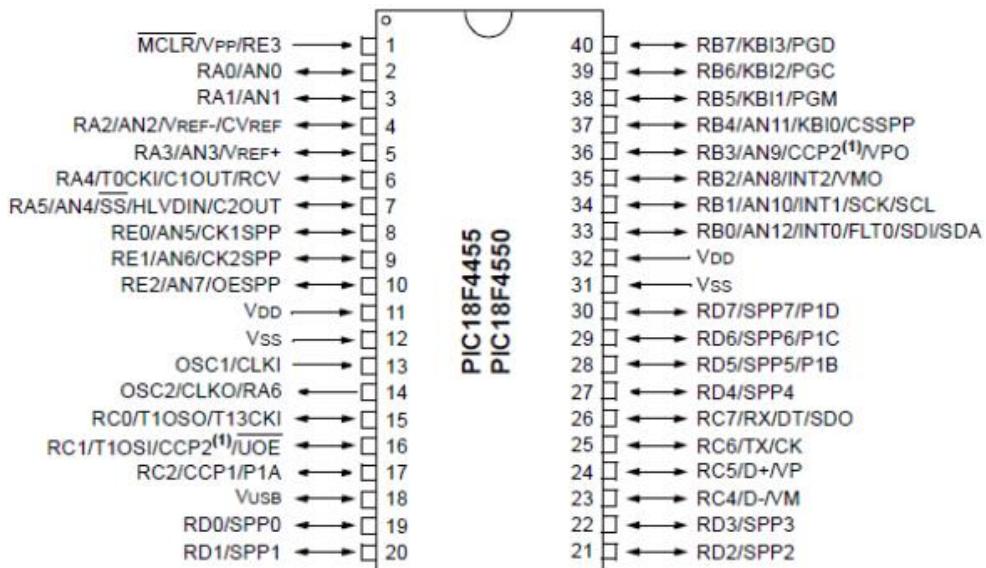


Figura 1.- Diagrama de los Pines del PIC18F4550

BLOQUES DEL PIC18F4550

A continuación, se muestra el diagrama de Bloques del microcontrolador PIC18F4550 (Figura 2):

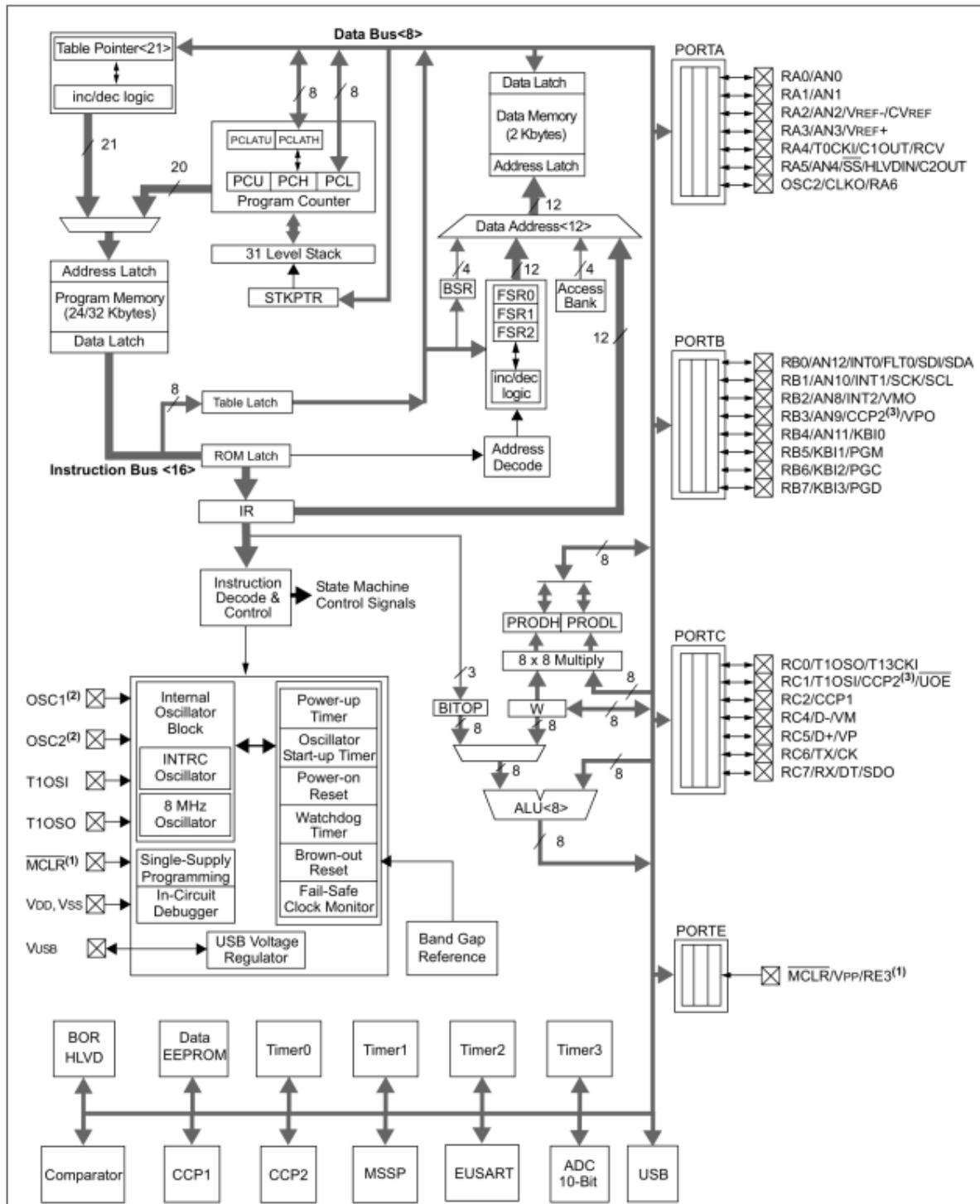


Figura 2.- Diagrama de Bloques del PIC18F4550

DIRECTIVAS DEL LENGUAJE

Las directivas son comandos que aparecen en el código fuente, pero no son traducidas directamente en código de operación. Son usadas para controlar el ensamblador: su entrada, salida y localización de datos. Muchas de las directivas de ensamblador tienen nombres y formatos alternos. Esos podrían existir para proporcionar compatibilidad con ensambladores previos de Microchip y para ser compatible con prácticas individuales de programación.

Existen cinco tipos básicos de directivas proporcionadas por MPASM:

- Directivas de control: Permiten secciones de código condicionalmente ensamblado.
- Directivas de datos: Controlan la localización de variables en memoria y proporcionan una manera para referirse a los datos simbólicamente mediante nombres significativos.
- Directivas de listado: Controlan el formato del archivo de listado (*.lst) generado por MPASM. Permitiendo la especificación de títulos, paginación y otros controles del listado.
- Macro directivas: Controlan la localización de datos y la ejecución mediante definiciones de macros.
- Directivas de archivo Objeto: Se usan solamente cuando se crea un archivo objeto.

MEMORIA

La Memoria de Datos (RAM) se divide en

- Registros de propósito general (GPRs): SRAM.
- Registros de funciones especiales (SFRs).

La sección denominada como “ACCESS BANK” comprende:

- Access Ram: los primeros 128 bytes de la memoria (00h-7Fh) en el banco 0 (GPRs).
- Los últimos 128 bytes (80h-FFh) en el banco 15 (SFRs).

La Memoria de Datos está dividida en 16 bancos diferentes, para seleccionar el área de trabajo se utilizan los siguientes bits:

Bit a (Access RAM bit):

- Si a = 0 \Leftrightarrow El registro BSR es ignorado y se accede directamente a los primeros 128 bytes (00h-7Fh) del banco 0 (GPRs), junto a los últimos 128 bytes (80h-FFh) del banco 15 (SFRs).
- Si a = 1 \Leftrightarrow El registro BSR especifica el banco usado por la instrucción.
- Bank Select Register (BSR): puntero de banco (4 bits).

Los registros pueden ser accedidos mediante direccionamiento directo, indirecto o indexado.

La instrucción MOVFF especifica los 12 bits de los registros fuente y destino, por lo tanto, ignora el BSR. El resto de instrucciones solo especifican los 8 bits menos significativos, por lo tanto, el bit a y/o el BSR serán tenidos en cuenta

Organización de la memoria de programa.

Un contador de programa de 21 bits es capaz de direccionar un espacio de memoria de programa de 2 Mbyte (MB). Accediendo a una localidad que este entre la memoria implementada físicamente y la dirección 2 MB causará una lectura de ceros (una instrucción NOP).

Special Function Registers (SFR)

Dentro de la memoria de datos, tenemos los SFR (Special Function Registers) a través de los cuales controlamos la mayor parte de las funciones del micro y sus periféricos. Cada periférico tiene asociado 2 o 3 SFR a través de los cuales se controla.

Ejemplos:

- Puertos B de entrada/salida: TRISB, PORTB, LATB
- TRISB determina si es entrada (1) o salida (1)
- En PORTB leemos valores del puerto.
- Usamos LATB para asignar valores a un puerto.
- Comunicaciones USART:
- TXREG, TXSTA: datos a transmitir, status/configuración de TX
- RCREG, RCSTA, datos recibidos, status/configuración de RX
- BRGH establece la velocidad del puerto (baudios) Se usan directamente con esos nombres dentro del compilador

Bits de configuración

También, dentro de la memoria del PIC hay un par de bytes que comprenden los llamados bits de configuración. Se definen dentro de nuestro programa C (o al crear el proyecto) y configuran aspectos básicos de la configuración del PIC al arrancar.

Ejemplos:

- Habilitación/deshabilitación del watchdog timer
- Decidir si tras un reset el puerto B se dedica a entradas analógicas o a un puerto normal de entrada/salida digital.

Si no los ponemos correctamente pueden ser fuente de errores frustrantes, sobre todo al principio:

- Si el puerto B arranca como entrada analógica no responderá a nuestros comandos para ponerlo a 0/1
- Si el “perro guardián” está activo pero nosotros no nos encargamos de “darle el hueso” el PIC se va a resetear cada poco tiempo y nuestro programa no va a funcionar.

Mostramos la tabla con funciones para la programación del PIC18F4550 (Figura 3):

TABLE 26-2: PIC18FXXXX INSTRUCTION SET

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes	
			MSb	Lsb					
BYTE-ORIENTED OPERATIONS									
ADDWF f, d, a	Add WREG and f	1	0010 01da	ffff ffff			C, DC, Z, OV, N	1, 2	
ADDWFC f, d, a	Add WREG and Carry bit to f	1	0010 00da	ffff ffff			C, DC, Z, OV, N	1, 2	
ANDWF f, d, a	AND WREG with f	1	0001 01da	ffff ffff			Z, N	1, 2	
CLRF f, a	Clear f	1	0110 101a	ffff ffff			Z	2	
COMF f, d, a	Complement f	1	0001 11da	ffff ffff			Z, N	1, 2	
CPFSEQ f, a	Compare f with WREG, Skip =	1 (2 or 3)	0110 001a	ffff ffff			None	4	
CPFSGT f, a	Compare f with WREG, Skip >	1 (2 or 3)	0110 010a	ffff ffff			None	4	
CPFSLT f, a	Compare f with WREG, Skip <	1 (2 or 3)	0110 000a	ffff ffff			None	1, 2	
DECf f, d, a	Decrement f	1	0000 01da	ffff ffff			C, DC, Z, OV, N	1, 2, 3, 4	
DECFSZ f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010 11da	ffff ffff			None	1, 2, 3, 4	
DCFSNZ f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100 11da	ffff ffff			None	1, 2	
INCf f, d, a	Increment f	1	0010 10da	ffff ffff			C, DC, Z, OV, N	1, 2, 3, 4	
INCFSZ f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011 11da	ffff ffff			None	4	
INFSNZ f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100 10da	ffff ffff			None	1, 2	
IORWF f, d, a	Inclusive OR WREG with f	1	0001 00da	ffff ffff			Z, N	1, 2	
MOVf f, d, a	Move f	1	0101 00da	ffff ffff			Z, N	1	
MOVFF f _s , f _d	Move f _s (source) to 1st word f _d (destination) 2nd word	2	1100 ffff 1111 ffff	ffff ffff ffff ffff			None		
MOVWF f, a	Move WREG to f	1	0110 111a	ffff ffff			None		
MULWF f, a	Multiply WREG with f	1	0000 001a	ffff ffff			None	1, 2	
NEGF f, a	Negate f	1	0110 110a	ffff ffff			C, DC, Z, OV, N		
RLCF f, d, a	Rotate Left f through Carry	1	0011 01da	ffff ffff			C, Z, N	1, 2	
RLNCF f, d, a	Rotate Left f (No Carry)	1	0100 01da	ffff ffff			Z, N		
RRCF f, d, a	Rotate Right f through Carry	1	0011 00da	ffff ffff			C, Z, N		
RRNCF f, d, a	Rotate Right f (No Carry)	1	0100 00da	ffff ffff			Z, N		
SETf f, a	Set f	1	0110 100a	ffff ffff			None	1, 2	
SUBFWB f, d, a	Subtract f from WREG with Borrow	1	0101 01da	ffff ffff			C, DC, Z, OV, N		
SUBWF f, d, a	Subtract WREG from f	1	0101 11da	ffff ffff			C, DC, Z, OV, N	1, 2	
SUBWFB f, d, a	Subtract WREG from f with Borrow	1	0101 10da	ffff ffff			C, DC, Z, OV, N		
SWAPF f, d, a	Swap Nibbles in f	1	0011 10da	ffff ffff			None	4	
TSTFSZ f, a	Test f, Skip if 0	1 (2 or 3)	0110 011a	ffff ffff			None	1, 2	
XORWF f, d, a	Exclusive OR WREG with f	1	0001 10da	ffff ffff			Z, N		
Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes	
			MSb	Lsb					
BIT-ORIENTED OPERATIONS									
BCF f, b, a	Bit Clear f	1	1001 bbba	ffff ffff			None	1, 2	
BSF f, b, a	Bit Set f	1	1000 bbba	ffff ffff			None	1, 2	
BTFC f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011 bbba	ffff ffff			None	3, 4	
BTFS f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010 bbba	ffff ffff			None	3, 4	
BTG f, d, a	Bit Toggle f	1	0111 bbba	ffff ffff			None	1, 2	
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110 0010	nnnn nnnn		None		
BN	n	Branch if Negative	1 (2)	1110 0110	nnnn nnnn		None		
BNC	n	Branch if Not Carry	1 (2)	1110 0011	nnnn nnnn		None		
BNN	n	Branch if Not Negative	1 (2)	1110 0111	nnnn nnnn		None		
BNOV	n	Branch if Not Overflow	1 (2)	1110 0101	nnnn nnnn		None		
BNZ	n	Branch if Not Zero	1 (2)	1110 0001	nnnn nnnn		None		
BOV	n	Branch if Overflow	1 (2)	1110 0100	nnnn nnnn		None		
BRA	n	Branch Unconditionally	2	1101 0nnn	nnnn nnnn		None		
BZ	n	Branch if Zero	1 (2)	1110 0000	nnnn nnnn		None		
CALL	n, s	Call Subroutine 1st word 2nd word	2	1110 110s	kkkk kkkk		None		
				1111 kkkk	kkkk kkkk				
CLRWD	—	Clear Watchdog Timer	1	0000 0000	0000 0100		TO, PD		
DAW	—	Decimal Adjust WREG	1	0000 0000	0000 0111		C		
GOTO	n	Go to Address 1st word 2nd word	2	1110 1111	kkkk kkkk		None		
				1111 kkkk	kkkk kkkk				
NOP	—	No Operation	1	0000 0000	0000 0000		None		
NOP	—	No Operation	1	1111 xxxx	xxxx xxxx		None	4	
POP	—	Pop Top of Return Stack (TOS)	1	0000 0000	0000 0110		None		
PUSH	—	Push Top of Return Stack (TOS)	1	0000 0000	0000 0101		None		
RCALL	n	Relative Call	2	1101 1nnn	nnnn nnnn		None		
RESET	—	Software Device Reset	1	0000 0000	1111 1111		All		
RETFIE	s	Return from Interrupt Enable	2	0000 0000	0001 000s		GIE/GIEH, PEIE/GIEL		
RETLW	k	Return with Literal in WREG	2	0000 1100	kkkk kkkk		None		
RETURN	s	Return from Subroutine	2	0000 0000	0001 001s		None		
SLEEP	—	Go into Standby mode	1	0000 0000	0000 0011		TO, PD		

Figura 3.- Conjunto de instrucciones del PIC18F4550

OSCILADOR

Un microcontrolador puede ser considerado como una computadora, pues posee una ALU (Unidad de Aritmética y Lógica), registros, buses y unidad de control, es decir tiene una CPU. La mayoría de los dispositivos de lógica secuencial, entre ellos los CPU, son de naturaleza síncrona. Es decir, están diseñados y operan en función de una señal de sincronización. Esta señal, conocida como señal de reloj, usualmente toma la forma de una onda cuadrada periódica. Calculando el tiempo máximo en que las señales eléctricas pueden moverse en las varias bifurcaciones de los muchos circuitos de un CPU, los diseñadores pueden seleccionar un período apropiado para la señal de reloj.

Para el correcto funcionamiento del microcontrolador se define la frecuencia del oscilador pues determinará los ciclos de instrucción; podemos utilizar un oscilador interno o externo, sin embargo, es recomendable utilizar un oscilador de cristal externo para evitar un mal funcionamiento frente a las variaciones de temperatura dentro del microcontrolador.

A continuación, mostramos la tabla del registro de configuración del oscilador (Figura 4):

REGISTER 2-2: OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0	R/W-1	R/W-0	R/W-0	R ⁽¹⁾	R-0	R/W-0	R/W-0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0
bit 7	bit 0						

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

-n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

bit 7	IDLEN: Idle Enable bit 1 = Device enters Idle mode on SLEEP instruction 0 = Device enters Sleep mode on SLEEP instruction
bit 6-4	IRCF2:IRCF0: Internal Oscillator Frequency Select bits 111 = 8 MHz (INTOSC drives clock directly) 110 = 4 MHz 101 = 2 MHz 100 = 1 MHz ⁽³⁾ 011 = 500 kHz 010 = 250 kHz 001 = 125 kHz 000 = 31 kHz (from either INTOSC/256 or INTRC directly) ⁽²⁾
bit 3	OSTS: Oscillator Start-up Time-out Status bit ⁽¹⁾ 1 = Oscillator Start-up Timer time-out has expired; primary oscillator is running 0 = Oscillator Start-up Timer time-out is running; primary oscillator is not ready
bit 2	IOFS: INTOSC Frequency Stable bit 1 = INTOSC frequency is stable 0 = INTOSC frequency is not stable
bit 1-0	SCS1:SCS0: System Clock Select bits 1x = Internal oscillator 01 = Timer1 oscillator 00 = Primary oscillator

Figura 4. Registro OSCCON para la Configuración del Oscilador

DESARROLLO

Presentamos a continuación los materiales a utilizar en la presente práctica:

- 8 Leds
- 8 Resistencias de 330 Ohms
- Cables
- Protoboard
- PIC18F4550 µvva
- Bus de Salida para PIC18F
- Ordenador para ejecutar nuestro programa

Simulación y Ensamblaje del Circuito

A continuación, mostramos el Diagrama el Circuito realizado en Proteus así como el ensamblaje del mismo en Protoboard:

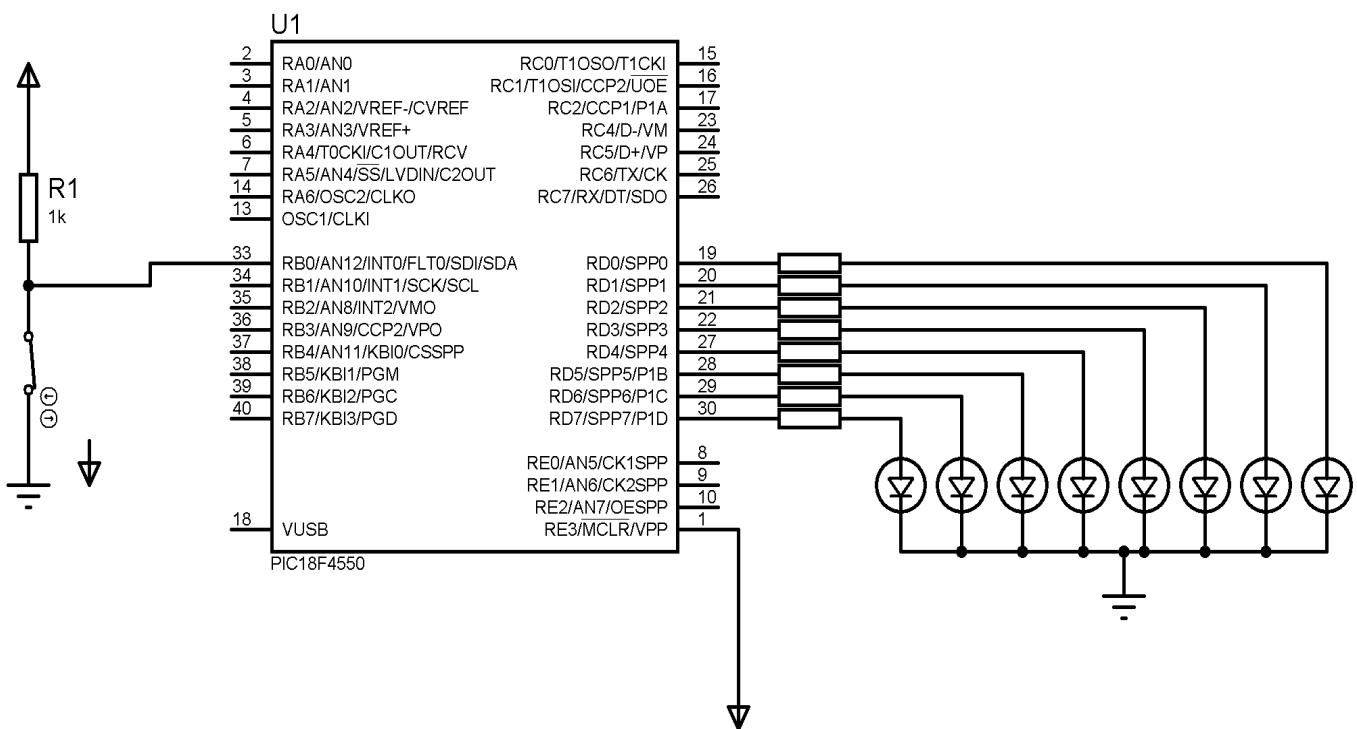


Figura 5.- Diagrama del circuito realizada en Proteus

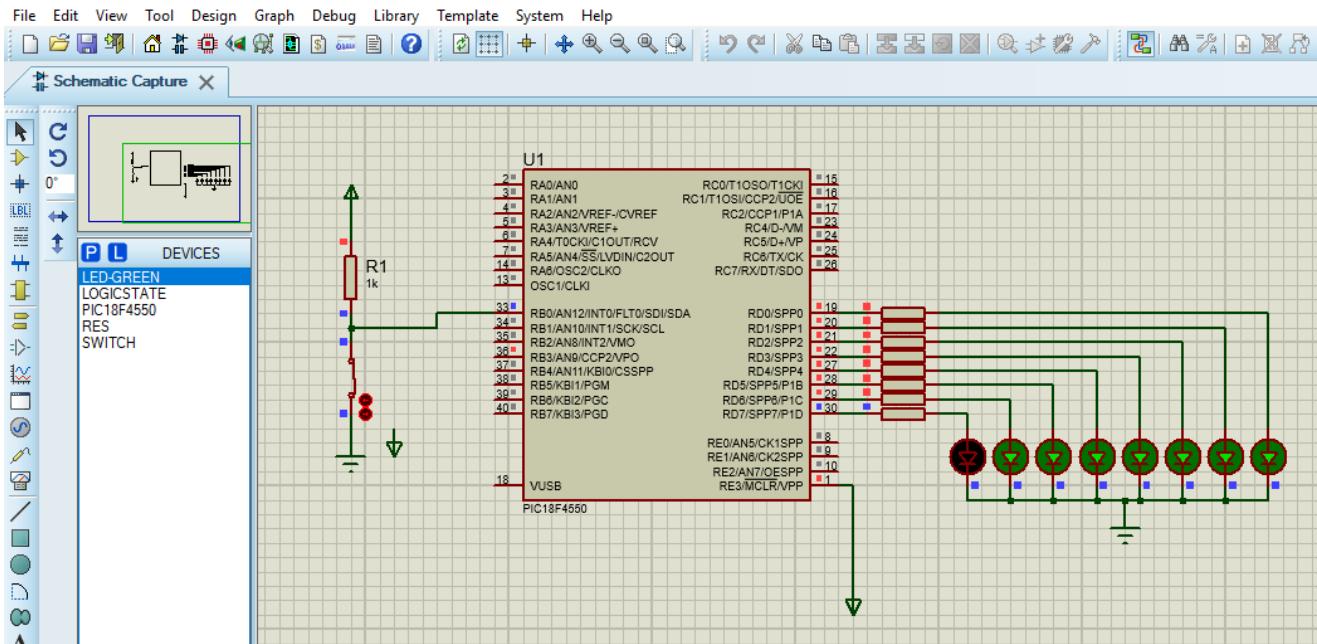


Figura 6.- Simulación del circuito realizada en Proteus

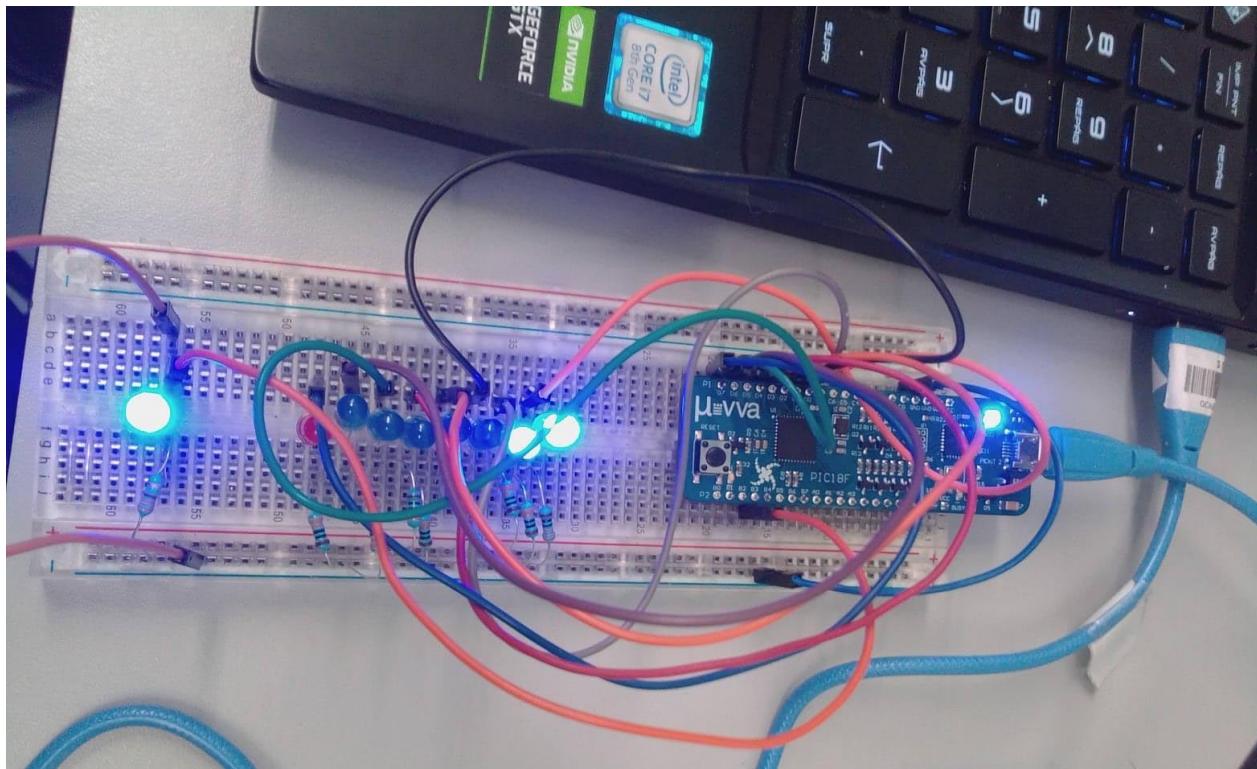


Figura 7.- Ensamblaje del circuito en protoboard con PIC18F4550

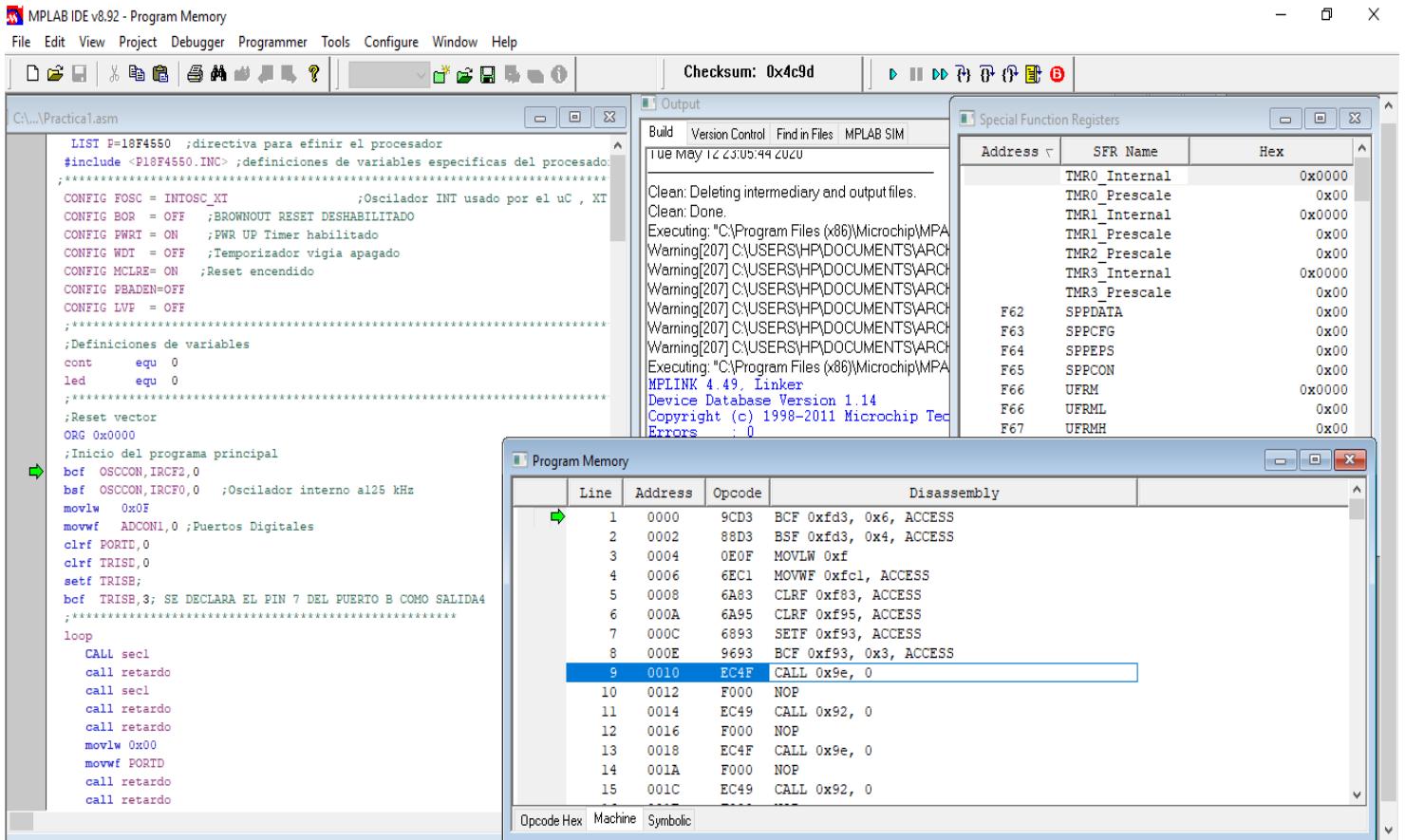


Figura 8.- Simulación del código Fuente en el Programa MPLAB IDE v8.92

Código Fuente

En el presente código se han colocado leds en cada uno de los pines del PUERTO D, con el propósito de encender secuencialmente cada uno de ellos, para lo cual debemos de declarar el puerto D como salida, esto lo hacemos escribiendo la palabra adecuada de configuración en su registro de dirección de datos TRISD. De este modo al escribir un 1 en un bit del registro TRISD, el puerto va a configurarse como entrada y al escribir un 0 el puerto se configurará como salida. Iniciamos nuestro oscilador interno con un valor de 125kHz, con la instrucción TRIS B,3 declaramos el pin 7 del puerto B como salida4.

```
LIST P=18F4550 ;directiva para definir el procesador
#include <P18F4550.INC> ;definiciones de variables específicas del procesador
;
*** ;Bits de configuración
CONFIG FOSC = INTOSC_XT           ;Oscilador INT usado por el uC , XT
usado por el USB
CONFIG BOR  = OFF    ;BROWNOUT RESET DESHABILITADO
CONFIG PWRT = ON     ;PWR UP Timer habilitado
CONFIG WDT  = OFF    ;Temporizador vigia apagado
CONFIG MCLRE= ON    ;Reset encendido
CONFIG PBADEN=OFF
CONFIG LVP   = OFF
;
*** ;Definiciones de variables
cont    equ  0
led     equ  0
;
*** ;Reset vector
ORG 0x0000
;Inicio del programa principal
bcf OSCCON,IRCF2,0
bsf OSCCON,IRCF0,0 ;Oscilador interno a125 kHz

movlw 0x0F
movwf ADCON1,0 ;Puertos Digitales
clrf PORTD,0
clrf TRISD,0
setf TRISB;
bcf TRISB,3; SE DECLARA EL PIN 7 DEL PUERTO B COMO SALIDA4
;
```

```

loop
    CALL sec1
    call retardo
    call sec1
    call retardo
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    call retardo
    call sec2
    call retardo
    call sec2
    call retardo
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    call retardo
    call sec3
    call retardo
    call sec3
    call retardo
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    call retardo
    call sec4
    call retardo
    call sec4
    call retardo
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    call retardo
    bra loop
;*****
***  

retardo    movlw 0xff
        movwf cont,0
nada      nop
        decfsz cont,1,0
        bra nada

```

```
return

sec1
    movlw 0x81
    movwf PORTD
    call retardo
    movlw 0X42
    movwf PORTD
    call retardo
    movlw 0X24
    movwf PORTD
    call retardo
    movlw 0X18
    movwf PORTD
    call retardo
    movlw 0X18
    movwf PORTD
    call retardo
    movlw 0x24
    movwf PORTD
    call retardo
    movlw 0X42
    movwf PORTD
    call retardo
    movlw 0X81
    movwf PORTD
    call retardo
    return

sec2
    movlw 0x01
    movwf PORTD
    call retardo
    movlw 0x03
    movwf PORTD
    call retardo
    movlw 0x07
    movwf PORTD
    call retardo
    movlw 0x1F
    movwf PORTD
    call retardo
    movlw 0x3F
    movwf PORTD
    call retardo
    movlw 0x7F
```

```
movwf PORTD
call retardo
movlw 0xFF
movwf PORTD
call retardo
movlw 0x7F
movwf PORTD
call retardo
movlw 0x3F
movwf PORTD
call retardo
movlw 0x7F
movwf PORTD
call retardo
movlw 0x1F
movwf PORTD
call retardo
movlw 0x0F
movwf PORTD
call retardo
movlw 0x07
movwf PORTD
call retardo
movlw 0x03
movwf PORTD
call retardo
movlw 0x01
movwf PORTD
call retardo
movlw 0x00
movwf PORTD
call retardo
return
sec3
    movlw 0x81
    movwf PORTD
    call retardo
    movlw 0xC3
    movwf PORTD
    call retardo
    movlw 0xE7
    movwf PORTD
    call retardo
    movlw 0xFF
    movwf PORTD
```

```
call retardo
movlw 0x7E
movwf PORTD
call retardo
movlw 0x3C
movwf PORTD
call retardo
movlw 0x18
movwf PORTD
call retardo
movlw 0x00
movwf PORTD
call retardo
return

sec4 movlw 0x80
movwf PORTD
call retardo
movlw 0x01
movwf PORTD
call retardo
movlw 0x40
movwf PORTD
call retardo
movlw 0x02
movwf PORTD
call retardo
movlw 0x20
movwf PORTD
call retardo
movlw 0x04
movwf PORTD
call retardo
movlw 0x20
movwf PORTD
call retardo
movlw 0x08
movwf PORTD
call retardo
movlw 0x10
movwf PORTD
call retardo
movlw 0x08
movwf PORTD
call retardo
```

```
    movlw 0x20
    movwf PORTD
    call retardo
    movlw 0x04
    movwf PORTD
    call retardo
    movlw 0x40
    movwf PORTD
    call retardo
    movlw 0x02
    movwf PORTD
    call retardo
    movlw 0x80
    movwf PORTD
    call retardo
    movlw 0x01
    movwf PORTD
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    return
END
```

CONCLUSIONES

Los microcontroladores PIC18F4550 nos permiten controlar de una manera fácil los diferentes tipos de displays y arreglos de leds gracias a la considerable cantidad de pines bidireccionales que contienen. Se conocen cada uno de los recursos con los que debe contar los diferentes módulos de entrenamiento de microcontroladores, para manejarse de acuerdo al conjunto de acciones a emprender en el desarrollo de las prácticas.

PRÁCTICA 2 EFECTOS DE ROTACIÓN CON LED'S PIC18F4550

OBJETIVO

Realizar distintos efectos de corrimiento y rotación de leds utilizando el PIC 18F4550 por medio de un código lenguaje ensamblador en el programa de simulación MPLAB, encender secuencialmente cada uno de ellos, para lo cual declararemos el puerto D como salida, aplicando instrucciones para aumentar el tipo de corrimiento y rotación de los leds.

MARCO TEÓRICO

Modos de direccionamiento

Direccionamiento Inherente y Literal

Inherente: no requieren argumentos. Ejemplo SLEEP, RESET, NOP. Literal: requieren un argumento explícito que actúa como literal u operando. Ejemplo 1: ADDLW, MOVLW, XORLW, las cuales operan sobre el registro W. Ejemplo 2: CALL, GOTO, las cuales incluyen un argumento que es una dirección de 20 bits relacionada con la memoria de programa.

Direccionamiento Directo

Especifica toda o una parte de las direcciones fuente/destino directamente desde el código de operación. Instrucciones como MOVF, ADDWF, usan un argumento (f) que especifica los 8 bits menos significativos del dato que van a leer/escribir, los 4 bits restantes están en el registro BSR.

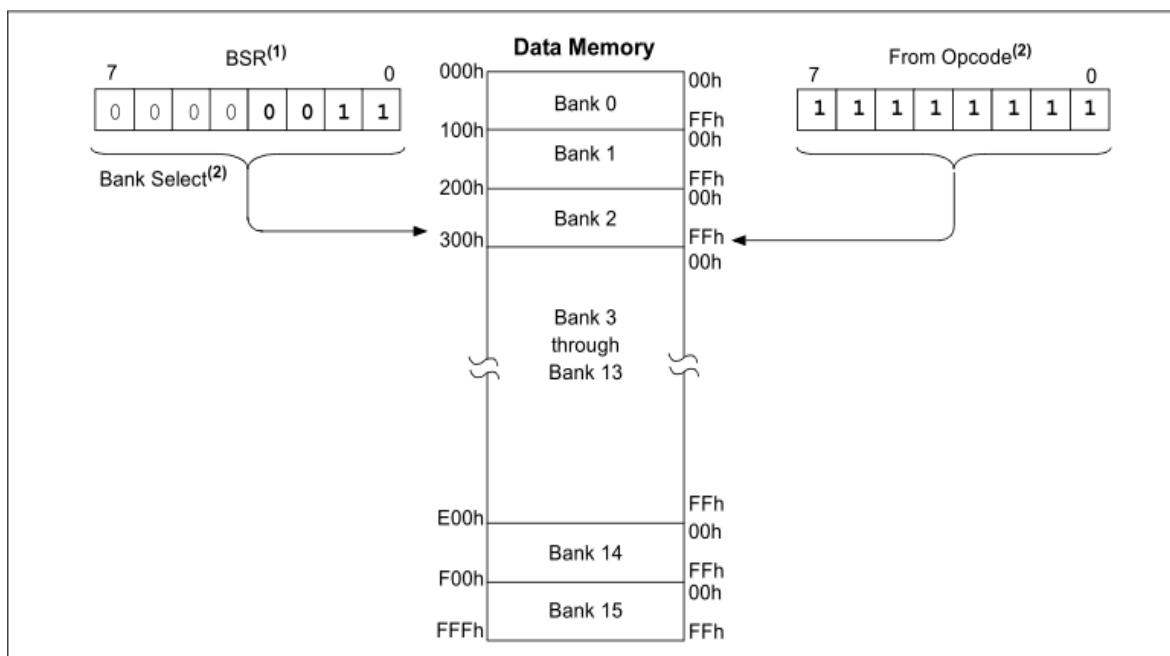


Figura 2.- Direccionamiento Directo

Direccionamiento Indirecto

Permite al usuario acceder a una posición en la memoria de datos sin necesidad de especificar una dirección en la instrucción.

- Los registros FSRs (File Selection Registers) actúan como punteros de la memoria de datos, pueden ser usados para operaciones de escritura o lectura.
- Los registros INDFs (Indirect File Operands) permiten leer/escribir el valor al que apunta el FSR, con opciones de auto-incremento, auto-decremento y offset con otro valor.
 - INDF: lee/escribe el valor al que apunta FSR, el valor de FSR no es modificado.
 - POSTINC: lee/escribe el valor al que apunta FSR, posteriormente el valor de FSR es incrementado en 1.
 - POSTDEC: lee/escribe el valor al que apunta FSR, posteriormente el valor de FSR es decrementado en 1.
 - PREINC: el valor del FSR es incrementado en 1, posteriormente lee/escribe el valor al que apunta FSR.
 - PLUSW: suma al FSR el valor con signo presente en el registro W (-128 a 127) y usa el nuevo valor en la operación.

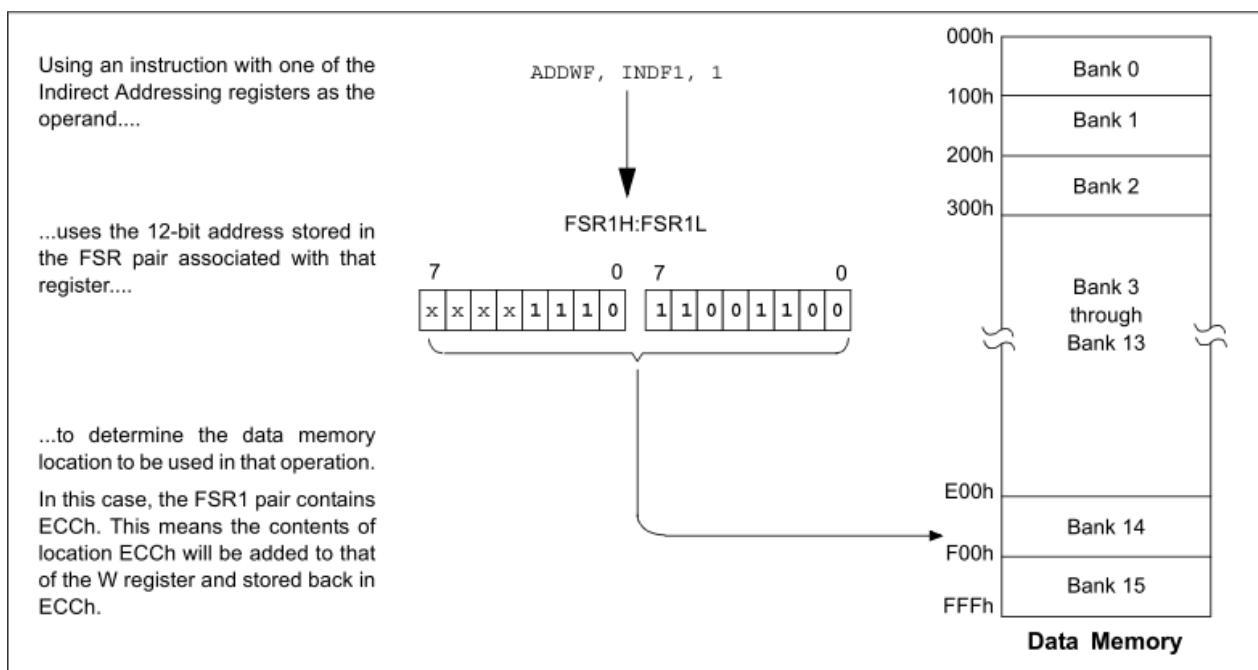


Figura 2.- Direccionamiento Indirecto

Puertos de Entradas/Salidas

Los Microcontroladores PIC tienen terminales de entrada/salida (I/O, Input/Output) divididos en puertos, que se encuentran nombrados alfabéticamente A, B, C, D, etc. Cada puerto puede tener hasta 8 terminales que se pueden comportar como una I/O digital. El PIC16F887 tiene hasta 35 I/O digitales de propósito general, y en dependencia de la configuración de sus periféricos pueden estar disponibles como I/O de propósito general. Por ejemplo, si utilizamos el módulo USART (Transmisión Serial) los pines correspondientes a este periférico serán utilizados para atender las operaciones del mismo y no podrán ser utilizados como I/O de propósito general.

Configuración de los Puertos

Los puertos pueden ser configurados como entradas o como salidas, es decir, dependen del programador si un pin servirá como entrada o como salida y en qué momento ocurrirá esto. Para realizar esto es necesario modificar el valor de los registros TRISx. Por ejemplo:

```
TRISA=%00000000      'Todos los pines de PORTA son salidas
TRISB=%11111111      'Todos los pines de PORTB son entradas
TRISC=%00001111      'Los pines de PORTC 0 al 3 son entradas y los pines 4 al 7 son
salidas
TRISD.3=0            'El pin 3 de PORTD se configura como salida sin afectar los demás
pines.
TRISD.5=1            'El pin 5 de PORTD se configura como entrada sin afectar los demás
pines.
```

Una forma fácil de recordar como configurar un pin como entrada o como salida es que el 1 se parece a la "I" de Input y el "0" se parece a la "O" de output, así todos los bits de los registros TRIS que tengan 1 serán entradas y los bits que tengan 0 serán salidas. Muchas veces deseamos configurar una gran cantidad de pines sin modificar el resto para ello se utilizan las operaciones bit a bit (and, or, xor), también llamadas operaciones máscara.

```
TRISD=TRISD or %00001111      'Los pines PORTC<0..3> se configuran como entradas sin
modificar los de más pines.
TRISD=TRISD and %10100110      'Los pines PORTD<0,3,4,6> se configuran como entradas
sin modificar el resto.
```

Lectura de los Puertos

Cuando configuramos un puerto o un pin como entrada podemos leerlo utilizando los registros PORTx, esto no significa que no podemos leer un puerto si lo configuramos como salida y deseamos saber su último valor para realizar modificaciones en dependencia de este. Por ejemplo:

```
TRISA=%11111111 'Configuramos todos los PINES de PORTA como entrada
TRISB=%10101010 'Configuramos los pines PORTB<0,2,4,6> como salida el resto como
entrada.
VAR1=PORTA          'El valor de PORTA se almacenará en la variable VAR1
VAR2=PORTB          'El valor de PORTB, incluyendo los pines de salida, se almacenan en
la variable VAR2
```

Escritura de Puertos

Cuando configuramos un puerto o pin como salida podemos escribirlo utilizando los registros PORTx, pero si algún pin del puerto no fue configurado como salida no se producirá ningún cambio al momento de escribirlo. Por ejemplo:

```
TRISA=%00000000      'Configuramos todos los pines de PORTA como salida
TRISB=%01010101      'Configuramos los pines PORTB<1,3,5,7> como salida el resto como
entrada.
...
PORTA=%11110000      'A PORTA se le asignara el valor $11110000
PORTB=%11111111      'Solo los pines PORTB<1,3,5,7> sufrirán cambio
```

DESARROLLO

Presentamos a continuación los materiales a utilizar en la presente práctica:

- 8 Leds
- 8 Resistencias de 330 Ohms
- Cables
- Protoboard
- PIC18F4550 µvva
- Bus de Salida para PIC18F
- Ordenador para ejecutar nuestro programa

Simulación y Ensamblaje del Circuito

A continuación, mostramos el Diagrama el Circuito realizado en Proteus así como el ensamblaje del mismo en Protoboard:

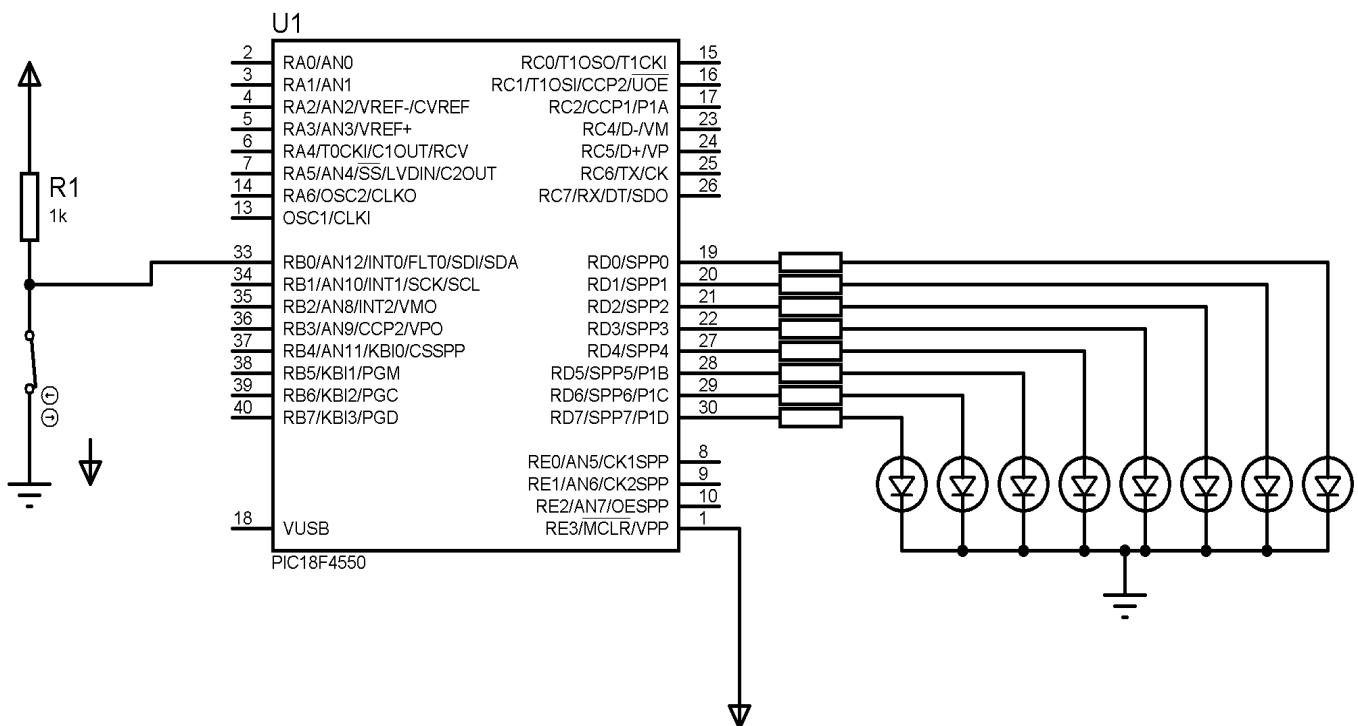


Figura 5.- Diagrama del circuito realizada en Proteus

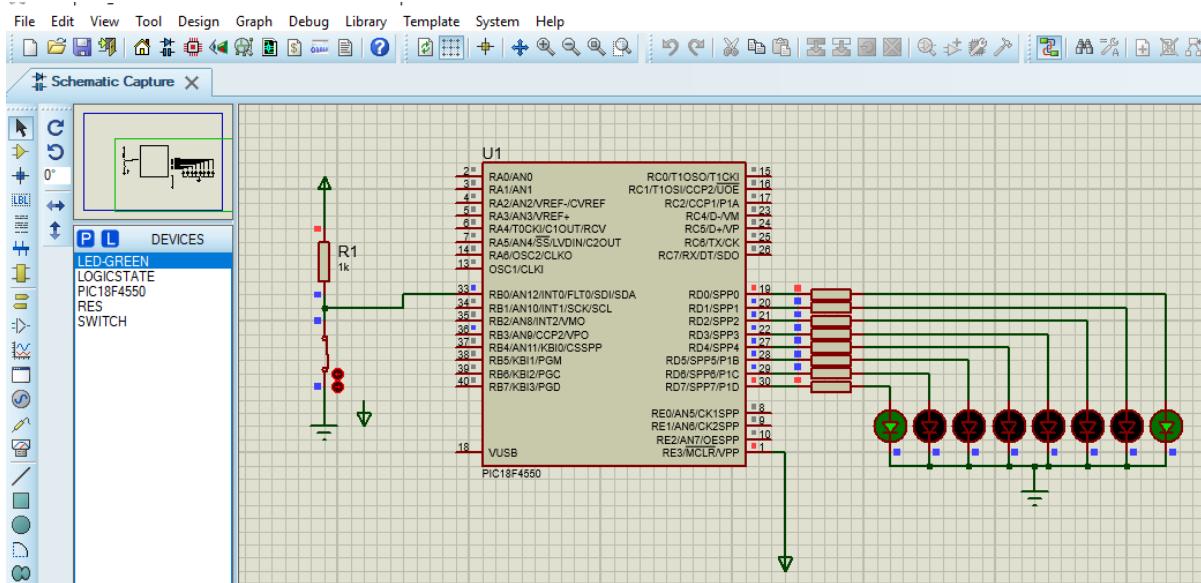


Figura 6.- Simulación del circuito realizada en Proteus

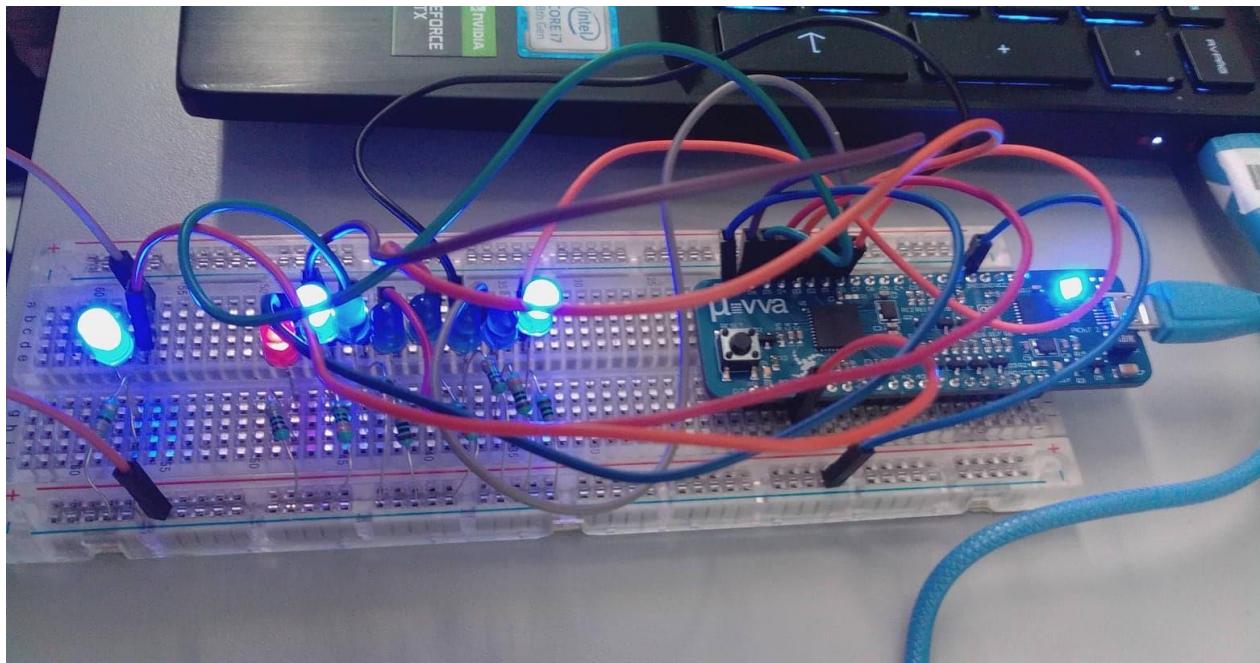


Figura 7.- Ensamblaje del circuito en protoboard con PIC18F4550

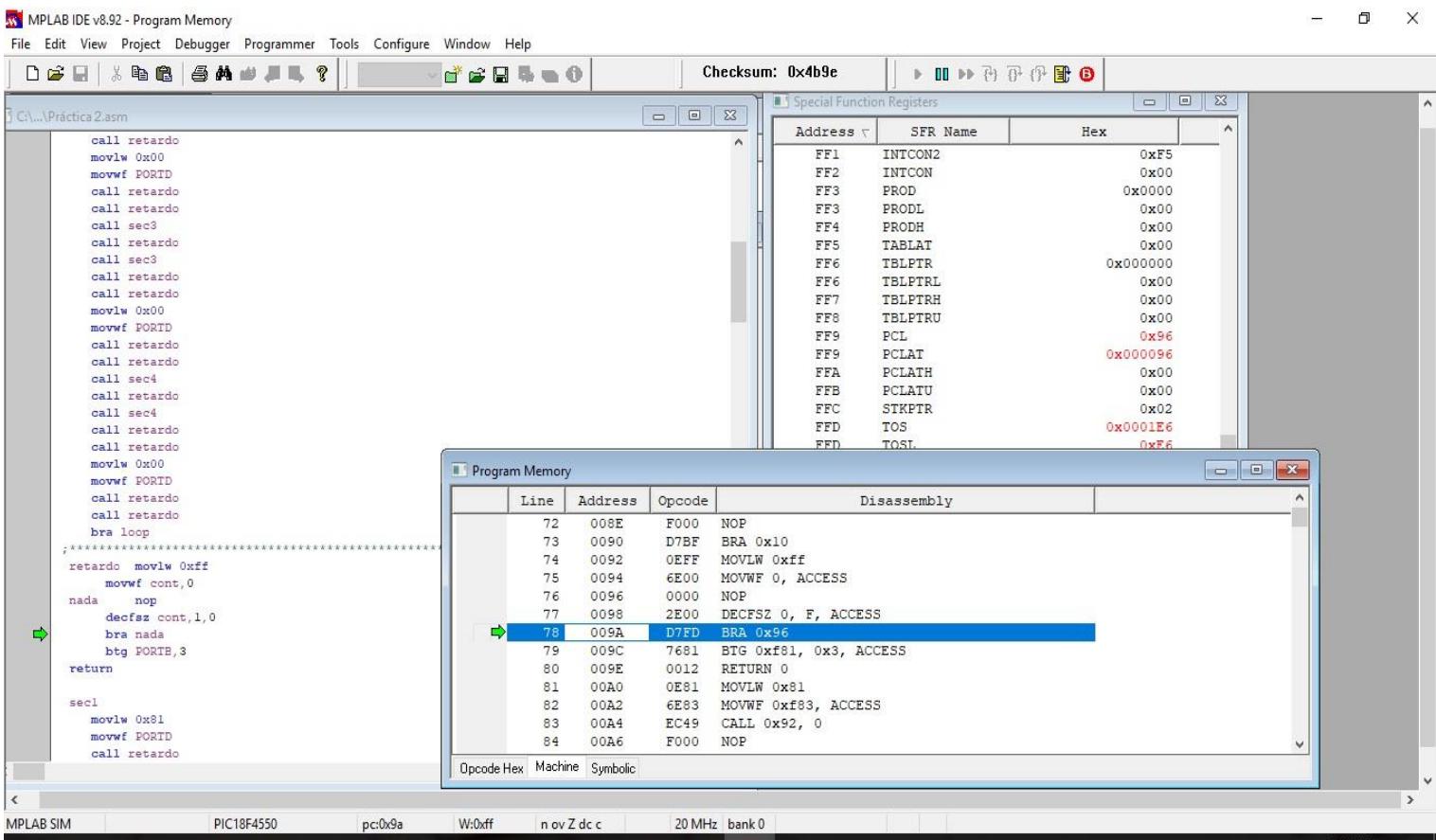


Figura 9.- Simulación del código Fuente en el Programa MPLAB IDE v8.92

Código Fuente

Para el siguiente código se han colocado leds en cada uno de los pines del PUERTO D, con el propósito de encender secuencialmente cada uno de ellos. El presente código es semejante a la práctica 1, solo que, a diferencia del mismo aquí le daremos aplicación a la instrucción BTG, que consiste en una instrucción orientada a bit que posee tres operandos, el registro de archivo (representado por “f”), el bit en el registro de archivo (representado por “b”) y la memoria accedida (representada por “a”). El operando de bit “b”, selecciona el número del bit afectado por la operación, mientras el de registro de archivo “f” representa el número del archivo en el cual está localizado el bit.

```
;*****  
LIST P=18F4550 ;directiva para definir el procesador  
#include <P18F4550.INC> ;definiciones de variables específicas del procesador  
  
;*****  
*** ;Bits de configuración  
CONFIG FOSC = INTOSC_XT ;Oscilador INT usado por el uC , XT usado  
por el USB  
CONFIG BOR = OFF ;BROWNOUT RESET DESHABILITADO  
CONFIG PWRT = ON ;PWR UP Timer habilitado  
CONFIG WDT = OFF ;Temporizador vigila apagado  
CONFIG MCLRE= ON ;Reset encendido  
CONFIG PBADEN=OFF  
CONFIG LVP = OFF  
;*****  
***  
;Definiciones de variables  
cont equ 0  
led equ 0  
;*****  
***  
;Reset vector  
ORG 0x0000  
;Inicio del programa principal  
bcf OSCCON,IRCF2,0  
bsf OSCCON,IRCF0,0 ;Oscilador interno a125 kHz  
  
movlw 0x0F  
movwf ADCON1,0 ;Puertos Digitales  
clrf PORTD,0  
clrf TRISD,0
```

```

setf TRISB;
bcf TRISB,3; SE DECLARA EL PIN 7 DEL PUERTO B COMO SALIDA4
;*****
loop
    CALL sec1
    call retardo
    call sec1
    call retardo
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    call retardo
    call sec2
    call retardo
    call sec2
    call retardo
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    call retardo
    call sec3
    call retardo
    call sec3
    call retardo
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    call retardo
    call sec4
    call retardo
    call sec4
    call retardo
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    call retardo
    bra loop
;*****
***  

retardo
    movlw 0xff

```

```

    movwf cont,0
nada
    nop
    decfsz cont,1,0
    bra nada
    btg PORTB,3
return

sec1
    movlw 0x81
    movwf PORTD
    call retardo
    movlw 0X42
    movwf PORTD
    call retardo
    movlw 0X24
    movwf PORTD
    call retardo
    movlw 0X18
    movwf PORTD
    call retardo
    movlw 0X18
    movwf PORTD
    call retardo
    movlw 0x24
    movwf PORTD
    call retardo
    movlw 0X42
    movwf PORTD
    call retardo
    movlw 0X81
    movwf PORTD
    call retardo
return

sec2
    movlw 0x01
    movwf PORTD
    call retardo
    movlw 0x03
    movwf PORTD
    call retardo
    movlw 0x07
    movwf PORTD
    call retardo
    movlw 0x1F

```

```
movwf PORTD
call retardo
movlw 0x3F
movwf PORTD
call retardo
movlw 0x7F
movwf PORTD
call retardo
movlw 0xFF
movwf PORTD
call retardo
movlw 0x7F
movwf PORTD
call retardo
movlw 0x3F
movwf PORTD
call retardo
movlw 0x7F
movwf PORTD
call retardo
movlw 0x1F
movwf PORTD
call retardo
movlw 0x0F
movwf PORTD
call retardo
movlw 0x07
movwf PORTD
call retardo
movlw 0x03
movwf PORTD
call retardo
movlw 0x01
movwf PORTD
call retardo
movlw 0x00
movwf PORTD
call retardo
return
sec3
    movlw 0x81
    movwf PORTD
    call retardo
    movlw 0xC3
    movwf PORTD
```

```
call retardo
movlw 0xE7
movwf PORTD
call retardo
movlw 0xFF
movwf PORTD
call retardo
movlw 0x7E
movwf PORTD
call retardo
movlw 0x3C
movwf PORTD
call retardo
movlw 0x18
movwf PORTD
call retardo
movlw 0x00
movwf PORTD
call retardo
return
```

```
sec4
movlw 0x80
movwf PORTD
call retardo
movlw 0x01
movwf PORTD
call retardo
movlw 0x40
movwf PORTD
call retardo
movlw 0x02
movwf PORTD
call retardo
movlw 0x20
movwf PORTD
call retardo
movlw 0x04
movwf PORTD
call retardo
movlw 0x20
movwf PORTD
call retardo
movlw 0x08
movwf PORTD
```

```
call retardo
movlw 0x10
movwf PORTD
call retardo
movlw 0x08
movwf PORTD
call retardo
movlw 0x20
movwf PORTD
call retardo
movlw 0x04
movwf PORTD
call retardo
movlw 0x40
movwf PORTD
call retardo
movlw 0x02
movwf PORTD
call retardo
movlw 0x80
movwf PORTD
call retardo
movlw 0x01
movwf PORTD
call retardo
movlw 0x00
movwf PORTD
call retardo
return
END
```

CONCLUSIONES

Para el presente código de rotación de luces, se usa una manera alterna para asignarles nombre a los registros en la memoria RAM, es mediante la directiva CBLOCK, la cual inicia en cero, después se enlistan el nombre de los registros y les asignará direcciones consecutivas, terminando con la directiva ENDC, también podríamos declararlos con la directiva EQU, como en el ejemplo anterior, pero esta es más elegante. El tiempo en el que durará encendido cada led, será aproximadamente, veinticinco veces más largo que el código de la práctica 1, esto se logra con la subrutina repite la cual llama a la rutina retardo veinticinco veces.

PRÁCTICA 3 INTERRUPCIONES PIC18F4550

OBJETIVO

Realizar distintos efectos de corrimiento y rotación de leds utilizando el PIC 18F4550 por medio de un código lenguaje ensamblador en el programa de simulación MPLAB, encender secuencialmente cada uno de ellos, para lo cual declararemos dicho puerto como salida. Aplicar la subrutina interrupción al programa irrumpiendo la secuencia del mismo, para ejecutar un programa especial llamando una rutina de servicio cuya característica principal es que al finalizar regrese al punto donde se interrumpió el programa.

MARCO TEÓRICO

Interrupciones

Los dispositivos PIC18F2455 / 2550/4455/4550 tienen múltiples fuentes de interrupción y una prioridad de interrupción característica que permite asignar cada fuente de interrupción un nivel de alta prioridad o un nivel de baja prioridad. Cada interrupción puede configurarse con prioridad alta o baja:

- Vector de Interrupción de alta prioridad es 00008h.
- Vector de interrupción de baja prioridad es 000018h.

Alta prioridad los eventos de interrupción interrumpirán cualquier interrupción de baja prioridad eso puede estar en progreso. Hay diez registros que se utilizan para controlar interrumpir la operación. Estos registros son:

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

Cada interrupción tiene tres bits de control de operación:

- **Bit de bandera (Flag bit):** indica que un evento de interrupción ocurrió.
- **Bit de habilitación (Enable bit):** permite al programa hacer un salto a la subrutina de atención de la interrupción cuando el bit de bandera (flag) es puesto a uno, en la dirección indicada por el vector de interrupción correspondiente.
- **Bit de prioridad (Priority bit):** selecciona alta o baja prioridad.

La prioridad de la interrupción es habilitada mediante el bit IPEN (RCON<7>).

Si la prioridad es habilitada (IPEN=1):

- El bit GIEH (INTCON<7>) habilita/deshabilita las interrupciones con prioridad alta.
- El bit GIEL (INTCON<6>) habilita/deshabilita las interrupciones con prioridad baja.

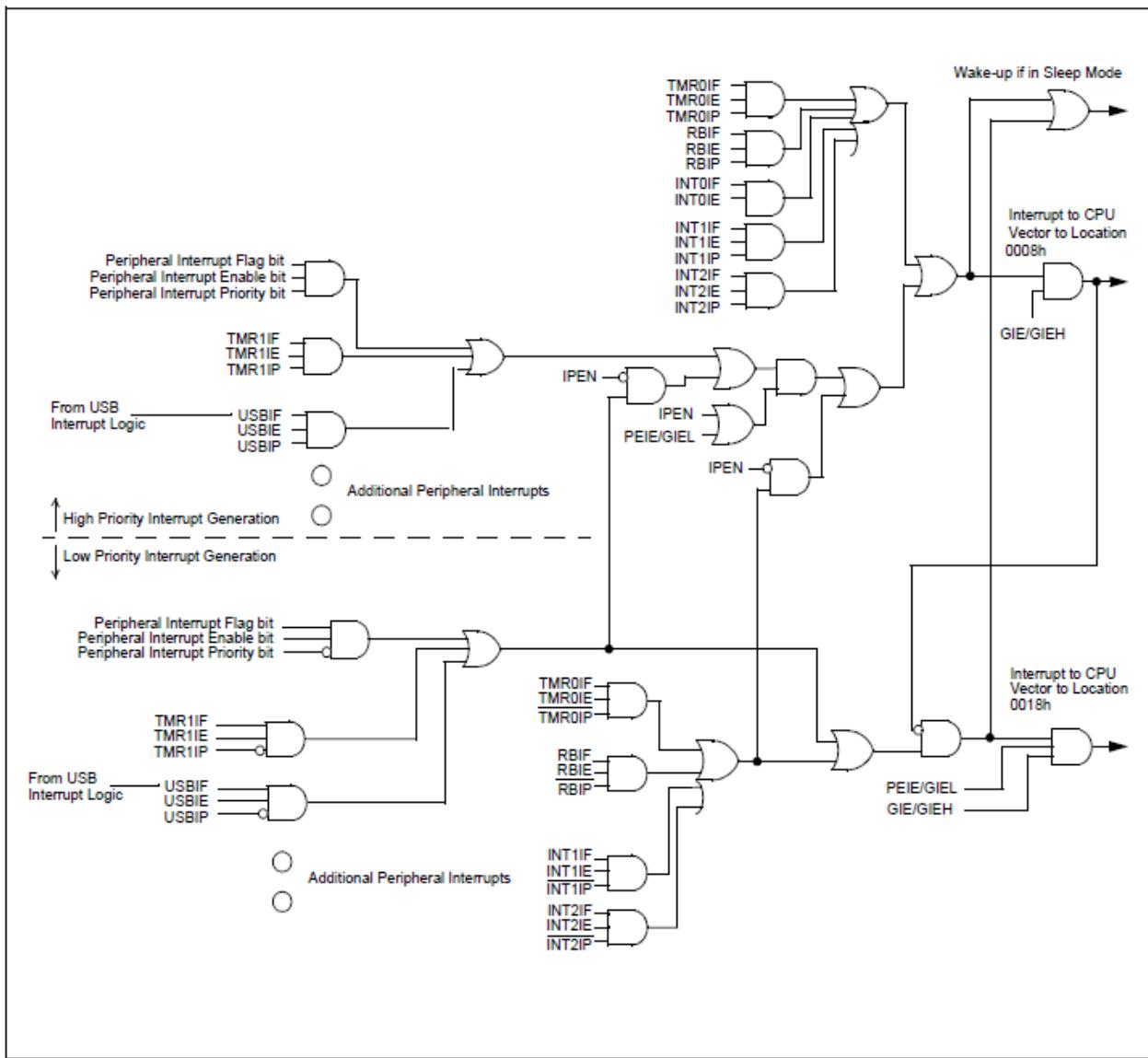


Figura 1.- Lógica de Interrupción del PIC 18F4550

Si la prioridad es deshabilitada (IPEN=0, valor por defecto):

- Modo de compatibilidad con dispositivos de rango medio (PIC16 y otros).
- El bit GIE (INTCON<7>) habilita/deshabilita las interrupciones.
- El bit PEIE (INTCON<6>) habilita/deshabilita las interrupciones por periféricos.
- Todas las interrupciones saltan a la dirección 000008h.

Registros sombra (Shadow Registers):

- Por defecto, la interrupción de alta prioridad de los PIC18F utiliza los registros internos de sombra (internal shadow registers) para guardar el valor de los registros W, STATUS y BSR.
- Para las interrupciones de baja prioridad todos los registros usados por la rutina de atención a la interrupción, deben ser guardados por software.

Una interrupción se define como un pedido de alta prioridad que un dispositivo exterior o un evento de programación solicita a la CPU para ejecutar otro programa.

El uC 18F4550 posee 21 fuentes de interrupciones. Se distinguen dos grupos de interrupciones:

1. Grupo general de interrupciones:

- Interrupción del Temporizador 0.
- Interrupción por cambio de estado en PORTB: Puertos B.4, B.5, B.6, B.7. En este caso cualquier cambio produce la misma interrupción.
- Interrupción externa 0: Puerto B.0
- Interrupción externa 1: Puerto B.1
- Interrupción externa 2: Puerto B.2

2. Grupo de interrupciones de periféricos:

- Interrupción del SPP
- Interrupción del A/D
- Interrupción de recepción de la EUSART
- Interrupción de transmisión de la EUSART
- Interrupción del MSSP
- Interrupción del CCP1
- Interrupción del Temporizador 2
- Interrupción del Temporizador 1
- Interrupción de fallo del oscilador
- Interrupción del comparador
- Interrupción del USB
- Interrupción de escritura en FLASH/EEPROM
- Interrupción de colisión del Bus (MSSP)
- Interrupción de detección de anomalías en VDD
- Interrupción del Temporizador 3
- Interrupción del CCP2

El grupo de registros que controlan las interrupciones son:

- INCONT
- INCONT2
- INCONT3
- RCON

INSTRUCCIONES PBP PARA INTERRUPCIONES

ON INTERRUPT GOTO handler

Activa la interrupción y salta a ejecutar el programa de la interrupción handler. Se puede dar cualquier nombre de etiqueta a este programa.

Seguido de esta instrucción se debe definir la fuente de interrupción mediante el registro INTCON.

- **INTCON = %10010000**; habilita la interrupción por el puerto B.0 en flanco positivo.
- **INTCON = %10001000**; habilita la interrupción por el puerto B.4---B.7 en flanco positivo.

Por defecto, la bandera **INTDEG** es 1 del registro **INTCON2** por lo que el cambio será detectado en flanco de subida. Para cambiar a flanco de bajada se debe cambiar la bandera INTDEG a 0, de la siguiente forma:

- **INTCON2.6 = 0**; censa en flanco de bajada.
- **INTCON2.6 = 1**; censa en flanco de subida.

DISABLE; deshabilita la interrupción, o sea, indica hasta que punto el programa atiende la interrupción.

RESUME; Retorna a la línea del programa en donde ocurrió la interrupción.

RESUME etiqueta; El programa continua en la subrutina o handler indicado por el nombre de la etiqueta.

ENABLE; Habilita la interrupción.

REGISTRO INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF ⁽¹⁾
bit 7							bit 0

GIE/GIEH: Bit de habilitación global de las interrupciones. Su función depende del valor de IPEN:

Cuando IPEN='0':

- GIE='0': Interrupciones deshabilitadas a nivel global
- GIE='1': Interrupciones habilitadas a nivel global

Cuando IPEN='1':

- GIEH='0': Interrupciones de alta prioridad deshabilitadas a nivel global
- GIEH='1': Interrupciones de alta prioridad habilitadas a nivel global

PEIE/GIEL: Bit de habilitación global de las interrupciones de periféricos. Su función depende del valor de IPEN:

Cuando IPEN='0':

- PEIE='0': Interrupciones de periféricos deshabilitadas a nivel global

- PEIE='1': Interrupciones de periféricos habilitadas a nivel global

Cuando IPEN='1':

- GIEL='0': Interrupciones de baja prioridad deshabilitadas a nivel global
- GIEL='1': Interrupciones de baja prioridad habilitadas a nivel global

TMROIE: Bit de habilitación de la interrupción de desbordamiento del Temporizador 0

- TMROIE='0': Interrupción de desbordamiento del Temporizador 0 deshabilitada
- TMROIE='1': Interrupción de desbordamiento del Temporizador 0 habilitada

INTOIE: Bit de habilitación de la interrupción externa 0

- INTOIE='0': Interrupción externa 0 deshabilitada
- INTOIE='1': Interrupción externa 0 habilitada

RBIE: Bit de habilitación de la interrupción por cambio en el Puerto B

- RBIE='0': Interrupción por cambio en el Puerto B deshabilitada
- RBIE='1': Interrupción por cambio en el Puerto B habilitada

TMROIF: Flag de la interrupción de desbordamiento del Temporizador 0

- TMROIF='0': No se ha producido desbordamiento del Temporizador 0
- TMROIF='1': Se ha producido desbordamiento del Temporizador 0

INTOIF: Flag de la interrupción externa 0

- INTOIF='0': No se ha producido un flanco en la línea RB0/INT0
- INTOIF='1': Se ha producido un flanco en la línea RB0/INT0

RBIF: Flag de la interrupción por cambio en el Puerto B

- RBIF='0': No se ha producido ningún cambio en ninguna de las líneas RB7..RB4
- RBIF='1': Se ha producido ningún cambio en ninguna de las líneas RB7..RB4

REGISTRO INTCON2

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

RBPU: Bit de habilitación de las resistencias de pull-up del Puerto B:

- RBPU='0': Las resistencias de pull-up del Puerto B están activadas
- RBPU='1': Las resistencias de pull-up del Puerto B están desactivadas

INTEDG0: Bit de selección de flanco de la interrupción externa 0:

- INTEDG0='0': La interrupción externa 0 se dispara por flanco de bajada
- INTEDG0='1': La interrupción externa 0 se dispara por flanco de subida

INTEDG1: Bit de selección de flanco de la interrupción externa 1:

- INTEDG1='0': La interrupción externa 1 se dispara por flanco de bajada
- INTEDG1='1': La interrupción externa 1 se dispara por flanco de subida

INTEDG2: Bit de selección de flanco de la interrupción externa 2:

- INTEDG2='0': La interrupción externa 2 se dispara por flanco de bajada
- INTEDG2='1': La interrupción externa 2 se dispara por flanco de subida

TMR0IP: Bit de selección de prioridad de la interrupción del Temporizador 0

- TMR0IP='0': Prioridad baja para la interrupción del Temporizador 0
- TMR0IP='1': Prioridad alta para la interrupción del Temporizador 0

RBIP: Bit de selección de prioridad de la interrupción por cambio en el Puerto B

- RBIP='0': Prioridad baja para la interrupción por cambio en el Puerto B
- RBIP='1': Prioridad alta para la interrupción por cambio en el Puerto B

DESARROLLO

Presentamos a continuación los materiales a utilizar en la presente práctica:

- 8 Leds
- 8 Resistencias de 330 Ohms
- Cables
- Protoboard
- PIC18F4550 Uva
- Bus de Salida para PIC18F
- Ordenador para ejecutar nuestro programa

Simulación y Ensamblaje del Circuito

A continuación, mostramos el Diagrama el Circuito realizado en Proteus así como el ensamblaje del mismo en Protoboard:

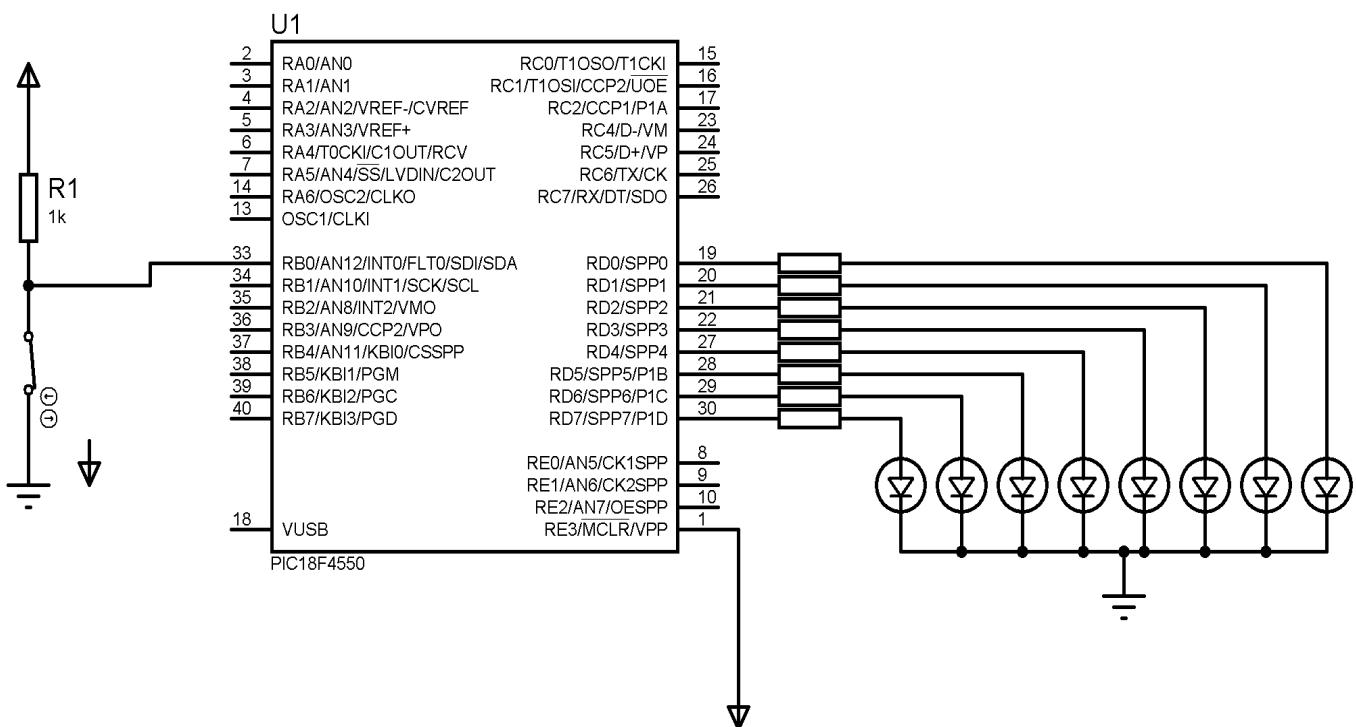


Figura 5.- Diagrama del circuito realizada en Proteus

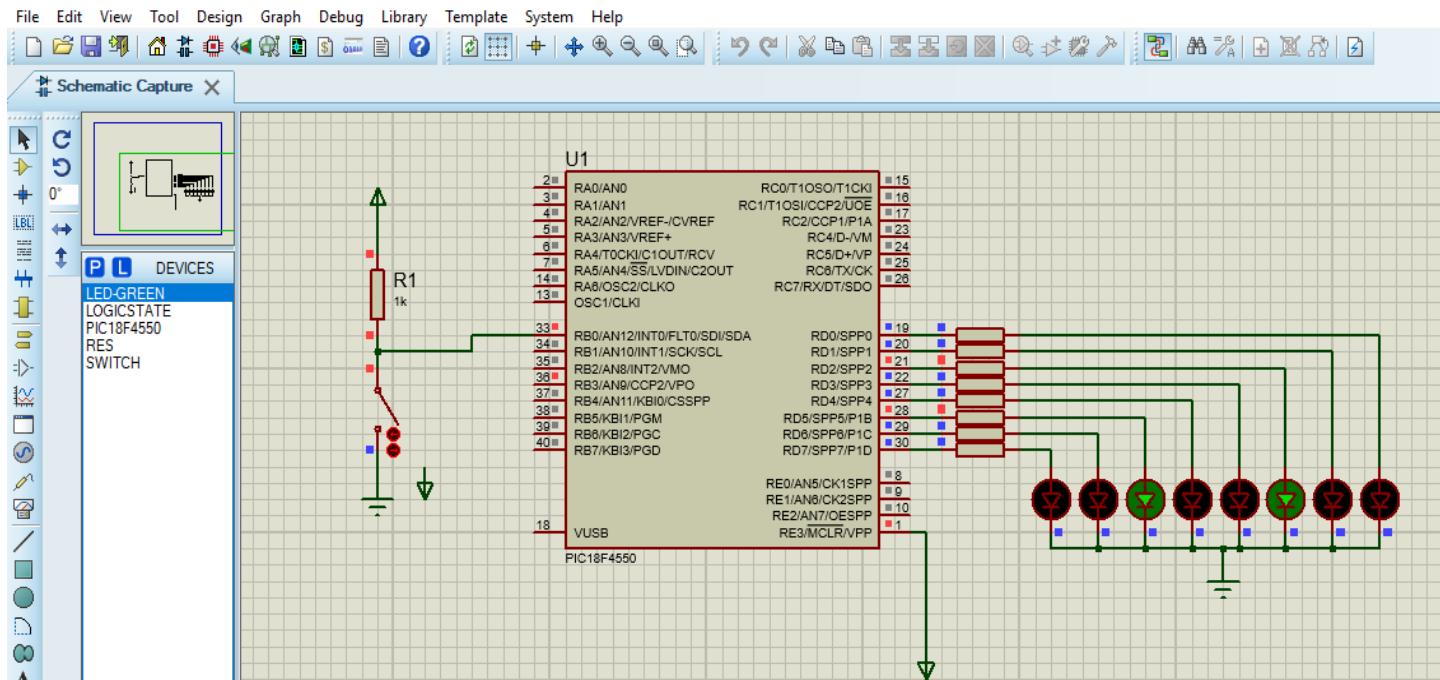


Figura 6.- Simulación del circuito realizada en Proteus

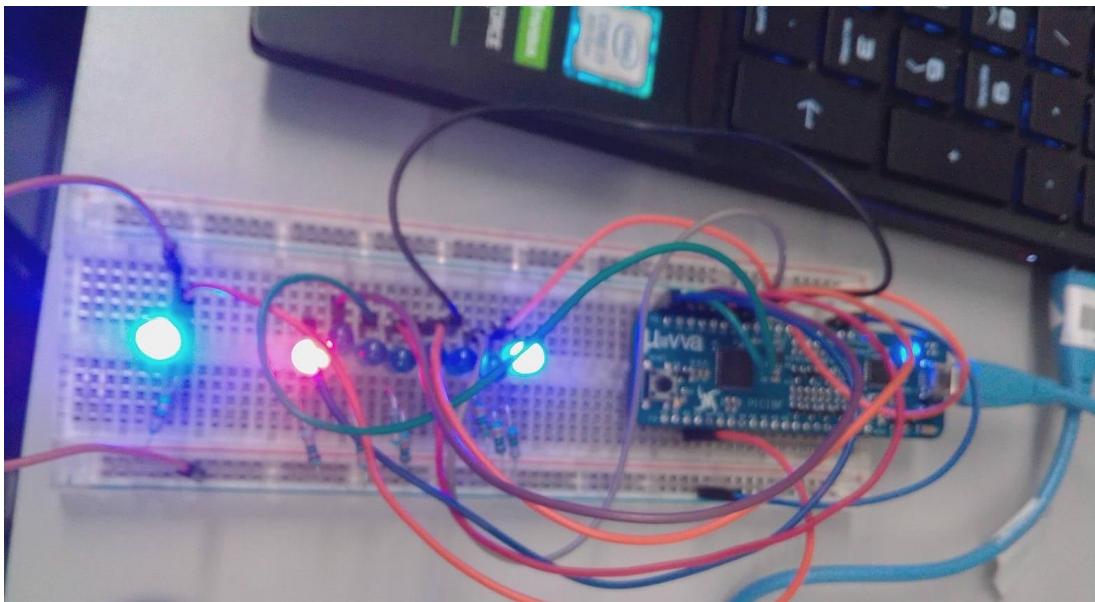


Figura 7.- Ensamblaje del circuito en protoboard con PIC18F4550

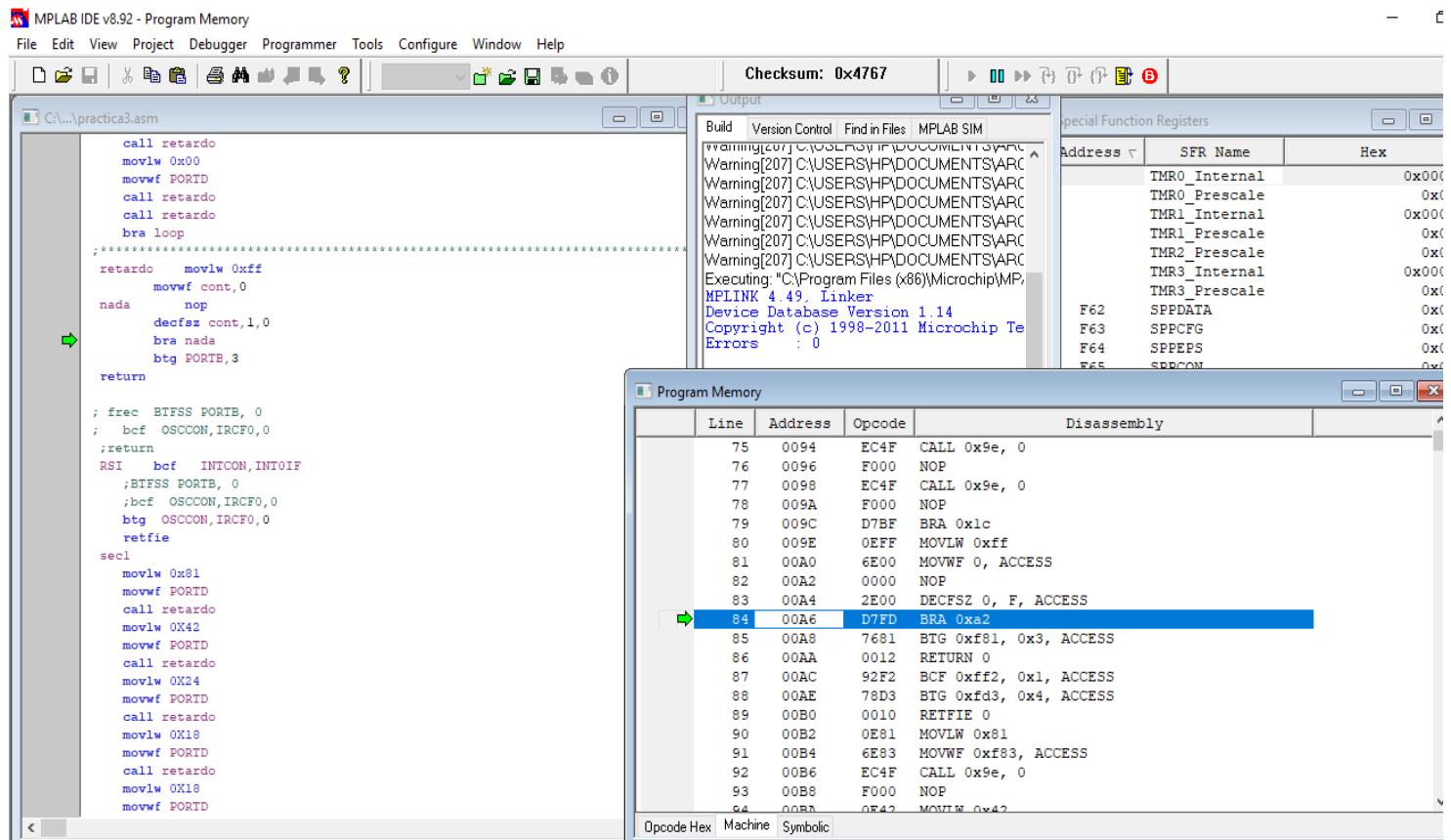


Figura 9.- Simulación del código Fuente en el Programa MPLAB IDE v8.92

Código Fuente

Para el siguiente código se han colocado leds en cada uno de los pines del PUERTO D, con el propósito de encender secuencialmente cada uno de ellos. El presente código es semejante a la práctica 1 y 2, en esta ocasión hacemos aplicación de las sub rutinas interrupciones, al activar las mismas, salta a ejecutar el programa de la interrupción handler. Acto seguido definimos la fuente de interrupción mediante el registro INTCON, este habilita la interrupción por el puerto B.0 en flanco positivo. Y para concluir la interrupción utilizamos la instrucción retfie.

```
LIST P=18F4550 ;directiva para definir el procesador
#include <P18F4550.INC> ;definiciones de variables específicas del procesador
;
;***** ;Bits de configuración
CONFIG FOSC = INTOSC_XT ;Oscilador INT usado por el uC , XT usado por el USB
CONFIG BOR = OFF ;BROWNOUT RESET DESHABILITADO
CONFIG PWRT = ON ;PWR UP Timer habilitado
CONFIG WDT = OFF ;Temporizador vigila apagado
CONFIG MCLRE= ON ;Reset APAGADO
CONFIG PBADEN=OFF
CONFIG LVP = OFF
;
;***** ;Definiciones de variables
cont equ 0
led equ 0
;
;***** ;Reset vector
ORG 0x0000
bra inicio
ORG 8
bra RSI
;Inicio del programa principal
Inicio bcf OSCCON,IRCF2,0
    bsf OSCCON,IRCF0,0 ;Oscilador interno a125 kHz

    movlw 0x0F
    movwf ADCON1,0 ;Puertos Digitales
    movlw 0x90
```

```
        movwf INTCON
        clrf PORTD,0
        clrf TRISE,0
        ;setf TRISB;
        ;setf PORTB;
        bcf TRISB,3; SE DECLARA EL PIN 3 DEL PUERTO B COMO SALIDA4

;*****loop
loop
        ;bcf OSCCON,IRCF2,0
        ;bsf OSCCON,IRCF0,0    ;Oscilador interno a125 kHz
        ;call freq
        CALL sec1
        call retardo
        call sec1
        call retardo
        call retardo
        call retardo
        movlw 0x00
        movwf PORTD
        call retardo
        call retardo
        call sec2
        call retardo
        call sec2
        call retardo
        call retardo
        movlw 0x00
        movwf PORTD
        call retardo
        call retardo
        call sec3
        call retardo
        call sec3
        call retardo
        call retardo
        movlw 0x00
        movwf PORTD
        call retardo
        call retardo
        call sec4
        call retardo
        call sec4
        call retardo
        call retardo
        movlw 0x00
```

```

movwf PORTD
call retardo
call retardo
bra loop
;*****
*** 
retardo    movlw 0xff
        movwf cont,0
nada      nop
decfsz cont,1,0
bra nada
btg PORTB,3
return

; freq BTFSS PORTB, 0
; bcf OSCCON,IRCF0,0
;return
RSI      bcf INTCON,INT0IF
        ;BTFSS PORTB, 0
        ;bcf OSCCON,IRCF0,0
        btg OSCCON,IRCF0,0
        retfie
sec1
        movlw 0x81
        movwf PORTD
        call retardo
        movlw 0X42
        movwf PORTD
        call retardo
        movlw 0X24
        movwf PORTD
        call retardo
        movlw 0X18
        movwf PORTD
        call retardo
        movlw 0X18
        movwf PORTD
        call retardo
        movlw 0x24
        movwf PORTD
        call retardo
        movlw 0X42
        movwf PORTD
        call retardo
        movlw 0X81

```

```
    movwf PORTD
    call retardo
    return
sec2
    movlw 0x01
    movwf PORTD
    call retardo
    movlw 0x03
    movwf PORTD
    call retardo
    movlw 0x07
    movwf PORTD
    call retardo
    movlw 0x1F
    movwf PORTD
    call retardo
    movlw 0x3F
    movwf PORTD
    call retardo
    movlw 0x7F
    movwf PORTD
    call retardo
    movlw 0xFF
    movwf PORTD
    call retardo
    movlw 0x7F
    movwf PORTD
    call retardo
    movlw 0x3F
    movwf PORTD
    call retardo
    movlw 0x7F
    movwf PORTD
    call retardo
    movlw 0x1F
    movwf PORTD
    call retardo
    movlw 0x0F
    movwf PORTD
    call retardo
    movlw 0x07
    movwf PORTD
    call retardo
    movlw 0x03
    movwf PORTD
```

```
call retardo
movlw 0x01
movwf PORTD
call retardo
movlw 0x00
movwf PORTD
call retardo
return

sec3
movlw 0x81
movwf PORTD
call retardo
movlw 0xC3
movwf PORTD
call retardo
movlw 0xE7
movwf PORTD
call retardo
movlw 0xFF
movwf PORTD
call retardo
movlw 0x7E
movwf PORTD
call retardo
movlw 0x3C
movwf PORTD
call retardo
movlw 0x18
movwf PORTD
call retardo
movlw 0x00
movwf PORTD
call retardo
return

sec4
movlw 0x80
movwf PORTD
call retardo
movlw 0x01
movwf PORTD
call retardo
movlw 0x40
movwf PORTD
call retardo
```

```
    movlw 0x02
    movwf PORTD
    call retardo
    movlw 0x20
    movwf PORTD
    call retardo
    movlw 0x04
    movwf PORTD
    call retardo
    movlw 0x20
    movwf PORTD
    call retardo
    movlw 0x08
    movwf PORTD
    call retardo
    movlw 0x10
    movwf PORTD
    call retardo
    movlw 0x08
    movwf PORTD
    call retardo
    movlw 0x20
    movwf PORTD
    call retardo
    movlw 0x04
    movwf PORTD
    call retardo
    movlw 0x40
    movwf PORTD
    call retardo
    movlw 0x02
    movwf PORTD
    call retardo
    movlw 0x80
    movwf PORTD
    call retardo
    movlw 0x01
    movwf PORTD
    call retardo
    movlw 0x00
    movwf PORTD
    call retardo
    return
END
```

CONCLUSIONES

En la presente práctica hicimos aplicación de la sub rutina interrupción, que consiste en que el procesador suspende temporalmente la tarea que está ejecutando para atender a algún periférico, mediante la ejecución de una rutina de servicio de interrupción, una vez que se concluye esta, el procesador continua con la tarea que estaba ejecutando antes de haber sido interrumpido.

Después el programa continua cíclicamente hasta que se produce alguna de las condiciones de interrupción, cuando eso ocurre el programa salta automáticamente a la subrutina de interrupción y cuando se dan las condiciones necesarias para terminar la subrutina vuelve al programa principal.

PRÁCTICA 4 CONTADOR DEL 0 AL 9 EN DISPLAY CON PIC18F4550

OBJETIVO

Reforzar el conocimiento y programación de los puertos de entrada y salida digital del microcontrolador PIC18F4550, así como la utilización del Timer0 en modo de temporizador. Para lo cual declararemos el puerto D como salida y de ese modo realizar la aplicación del timer en función de un contador decimal ascendente del dígito 0 al 9 con retardo por temporizador (Timer0).

MARCO TEÓRICO

Display de 7 Segmentos

El display 7 segmentos es un componente electrónico muy utilizado para representar visualmente números y letras, es de gran utilidad dado su simpleza para implementar en cualquier proyecto electrónico. Está compuesto por 7 dispositivos lumínicos (Led) que forman un “8”, de esta forma controlando el encendido y apagado de cada led, podremos representar el numero o letra que necesitamos.

Ánodo Común y Cátodo Común

Existen dos tipos de display de 7 segmentos, su principal diferencia es la conexión que debemos implementar para encenderlos, estos dos tipos se conocen como Ánodo común y Cátodo común. En los 7 segmentos de Cátodo Común, el punto circuital en común para todos los Led es el Cátodo (Gnd), cero volt, Mientras que el Ánodo común el punto de referencia es Vcc (5 volt). Teniendo en cuenta estas consideraciones la forma de encender los led debe realizase de diferente manera en función de que elemento tengamos (Ánodo o Cátodo común).

Cada Led trabaja con tensiones y corrientes bajas por lo tanto se pueden conectar directamente a compuertas lógicas o pines de salida de un micro controlador, igualmente siempre es recomendable para aumentar la vida útil de los mismos, conectarle una resistencia en serie entre el pin de salida del micro controlador y el de entrada de 7 segmentos, la intensidad lumínica en este caso dependerá del valor de la resistencia agregada.

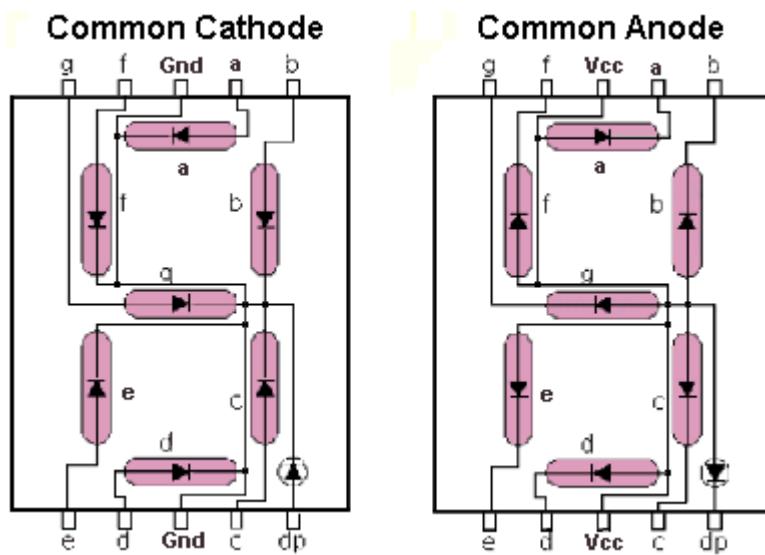


Figura 1.- Lógica de Interrupción del PIC 18F4550

Control de los Segmentos

Fácil, Lo primero que tenemos que identificar es con que tipo de display estamos trabajando (Cátodo o Ánodo común), una vez identificado nos basamos en la siguiente tabla de verdad dado el caso que corresponda.

		Catodo Común						
	Numero	A	B	C	D	E	F	G
Enable	0	1	1	1	1	1	1	0
0	1	0	1	1	0	0	0	0
0	2	1	1	0	1	1	0	1
0	3	1	1	1	1	0	0	1
0	4	0	1	1	0	0	1	1
0	5	1	0	1	1	0	1	1
0	6	1	0	1	1	1	1	1
0	7	1	1	1	0	0	0	0
0	8	1	1	1	1	1	1	1
0	9	1	1	1	1	0	1	1

		Anodo Común						
	Numero	A	B	C	D	E	F	G
Enable	0	0	0	0	0	0	0	1
1	1	1	0	0	1	1	1	1
1	2	0	0	1	0	0	1	0
1	3	0	0	0	0	1	1	0
1	4	1	0	0	1	1	0	0
1	5	0	1	0	0	1	0	0
1	6	0	1	0	0	0	0	0
1	7	0	0	0	1	1	1	1
1	8	0	0	0	0	0	0	0
1	9	0	0	0	0	1	0	0

Si queremos entonces mostrar el número cero tendríamos que encender los LED correspondientes a los segmentos A, B, C, D, E, F, para el numero uno serían entonces el segmento B y C.

Para encender los LED de un display de cátodo común se conectan a través de una resistencia a 5V, mientras que los del display de ánodo común se tienen que conectar a través de una resistencia a tierra.

Conectando un display 7 segmentos a un PIC

Los microcontroladores de MICROCHIP pueden manejar directamente ciertas cargas como los LED, gracias a esto podemos conectar directamente un display 7 segmentos a través de unas resistencias al PIC. Se muestra en la tabla siguiente, la asignación de los pines del puerto D para la conexión de los segmentos del display:

RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
Punto decimal	g	f	E	d	c	b	a

Los códigos de siete segmentos para el display de ánodo común, (considerando en este caso que encienden con cero) son los siguientes:

	Hexadecimal	Binario	Dígito
	C0	1100 0000	0
	F9	1111 1001	1
	A4	1010 0100	2
	B0	1011 0000	3
	99	1001 1001	4
	92	1001 0010	5
	82	1000 0010	6
	B8	1011 1000	7
	80	1000 0000	8
	98	1001 1000	9

Temporizador/contador TIMERO

Configurable como temporizador ($\text{TOCS} = 0$, reloj interno) o contador ($\text{TOCS} = 1$, reloj externo), en modos de operación de 8 y 16 bits.

- Preescala:

- Contador de 8 bits programable por software ($\text{PSA} = 1$):
- Preescalas disponibles: 1:2, 1:4, 1:8, 1:16, 1:32, 1:64, 1:128: 1:256.

Cuando se modifica el TMRO, se borrará el contador de la preescala, pero no cambiará la preescala asignada.

Puede producir una interrupción por desbordamiento del TMRO.

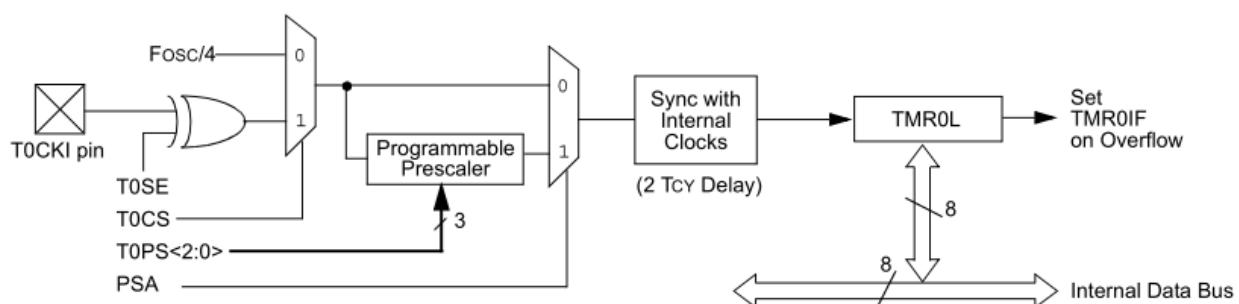
- Modo de 8 bits: cuando pasa de FFh a 00h.
- Modo de 16 bits: cuando pasa de FFFFh a 0000h.

Cuando se produce un desbordamiento, se pone a 1 el bit TMROIF.

- La interrupción puede ser enmascarada (habilitada/deshabilitada) a través del bit TMROIE.
- Antes de habilitar nuevamente la interrupción, el usuario debe borrar el bit TMROIF en la rutina de atención a la interrupción.
- Si el registro TMRO es escrito, el incremento es deshabilitado por dos ciclos de instrucción.
- En modo de operación de 8 bits, el TMRO puede ser escrito/leído por medio del registro TMROL.
- En modo de operación de 16 bits, la parte alta del TMRO no se puede leer ni escribir, sin embargo se puede leer/escribir indirectamente a través del buffer TMROH, así:

Cuando se hace una operación de lectura del TMROL, el TMROH es cargado con la parte alta del TMRO.

Para una operación escritura, se debe cargar el registro TMROH con el valor adecuado. La parte alta del TMRO es cargada con el contenido del registro TMROH cuando hay una escritura en el registro TMROL. Esto permite escribir los 16 bits del registro en un solo evento.



Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

Figura 2.- Diagrama de bloques TIMERO, modo de 8 bits

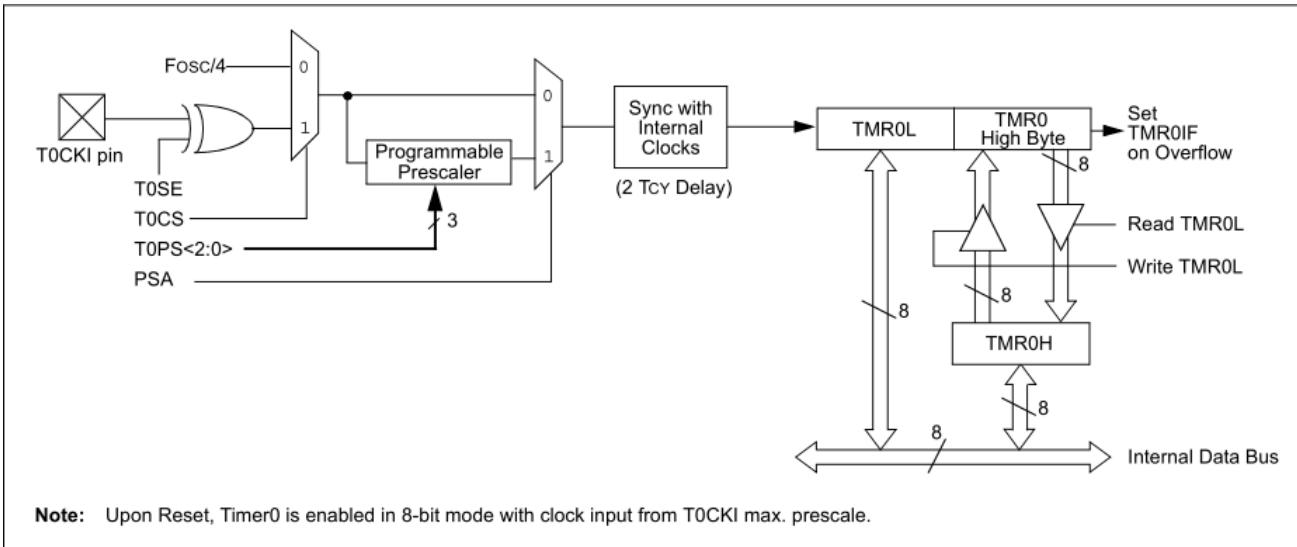


Figura 3.- Diagrama de bloques TIMER0, modo de 16 bits

Módulo Temporizador/contador TIMER2

- Temporizador de 8 bits en registro TMR2.
- Registro de periodo en PR2.
- Preescala programable a: 1:1, 1:4, 1:16.
- Postescala programable a: 1:1, 1:2, ..., 1:16.
- Permite la generación de interrupción cuando coinciden TMR2 y PR2 (depende de la postescala).
- Usado como base de tiempo para el módulo CCP en modo PWM. Uso opcional como reloj de desplazamiento para el módulo MSSP.
- Control del módulo mediante el registro T2CON.
- Los contadores de la preescala y la postescala son borrados cuando:
 - Se escribe el registro TMR2 (no cambiará la preescala asignada).
 - Se escribe el registro T2CON.
 - Cualquiera de los reset del dispositivo (POR, MCLR, WDT, BOR).

Operación:

Cuando la preescala es 1:1, el TMR2 es incrementado cada ciclo de instrucción (Fosc/4). La preescala puede modificar el funcionamiento para que trabaje cada 4 (1:4) o cada 16 (1:16) ciclos de instrucción. El TMR2 es comparado con el valor del PR2 en cada ciclo, de esta

menera cuando los valores coinciden, se genera una señal que inicia nuevamente el TMR2 en 00h, y genera un incremento en el contador de postescala.

Cuando el TMR2 y el PR2 coincide, se genera una señal que incrementa el contador de postescala, la cual dependiendo de su configuración (1:1 ... 1:16), puede generar una interrupción, mediante el bit TMR2IF (PIR<1>). Esta interrupción puede ser habilitada con el bit TMR2IE (PIE1<1>).

La salida (sin postescala) del TMR2 es usado principalmente para el módulo CCP, cuando es usado como base de tiempo para el modo PWM.

El TMR2 también puede ser usado como el reloj de desplazamiento para el módulo MSSP cuando esta en modo SPI.

Temporizador 0 Interrupción

La interrupción TMRO se genera cuando TMRO registrar desbordamientos de FFh a 00h en modo de 8 bits, o de FFFFh a 0000h en modo de 16 bits. Este desbordamiento establece el bit de bandera TMROIF. La interrupción puede ser enmascarada por borrando el bit TMROIE (INTCON <5>). Antes de volver a habilitar la interrupción, el bit TMROIF debe borrarse en software por la Rutina de servicio de interrupción. Como Timer0 se apaga en modo de Suspensión, el TMRO la interrupción no puede despertar el procesador del modo de suspensión.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TMROL	Timer0 Register Low Byte								52
TMROH	Timer0 Register High Byte								52
INTCON	GIE/GIEH	PEIE/GIEL	TMROIE	INTOIE	RBIE	TMROIF	INTOIF	RBIF	51
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMROIP	—	RBIP	51
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	52
TRISA	—	TRISA6 ⁽¹⁾	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	54

Figura 4.- Registros Asociados con Timer0

DESARROLLO

Presentamos a continuación los materiales a utilizar en la presente práctica:

- 1 Display de 7 segmentos
- 8 Resistencias de 330 Ohms
- 1 Resistencia de 1k Ohm
- Cables
- Protoboard
- PIC18F4550 Uva
- Bus de Salida para PIC18F
- Ordenador para ejecutar nuestro programa

Simulación y Ensamblaje del Circuito

A continuación, mostramos el Diagrama el Circuito realizado en Proteus así como el ensamblaje del mismo en Protoboard:

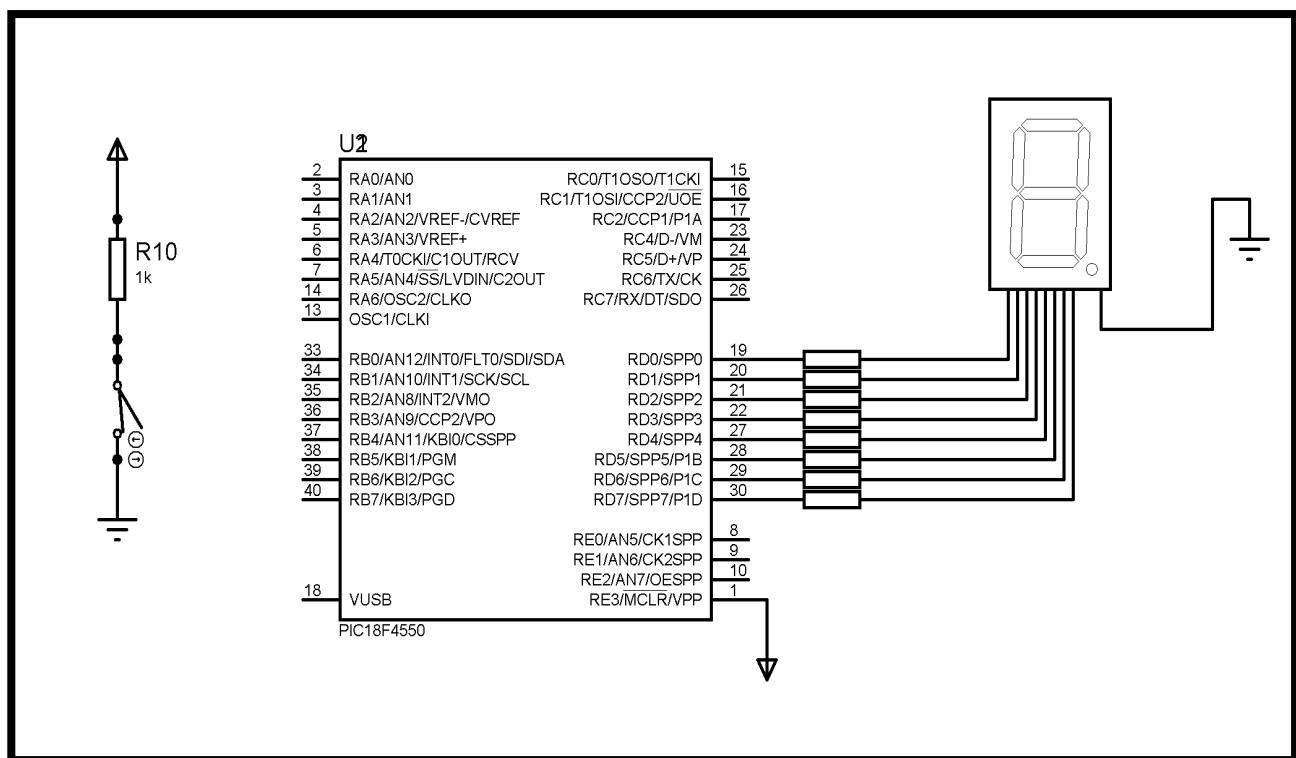


Figura 5.- Diagrama del circuito realizada en Proteus

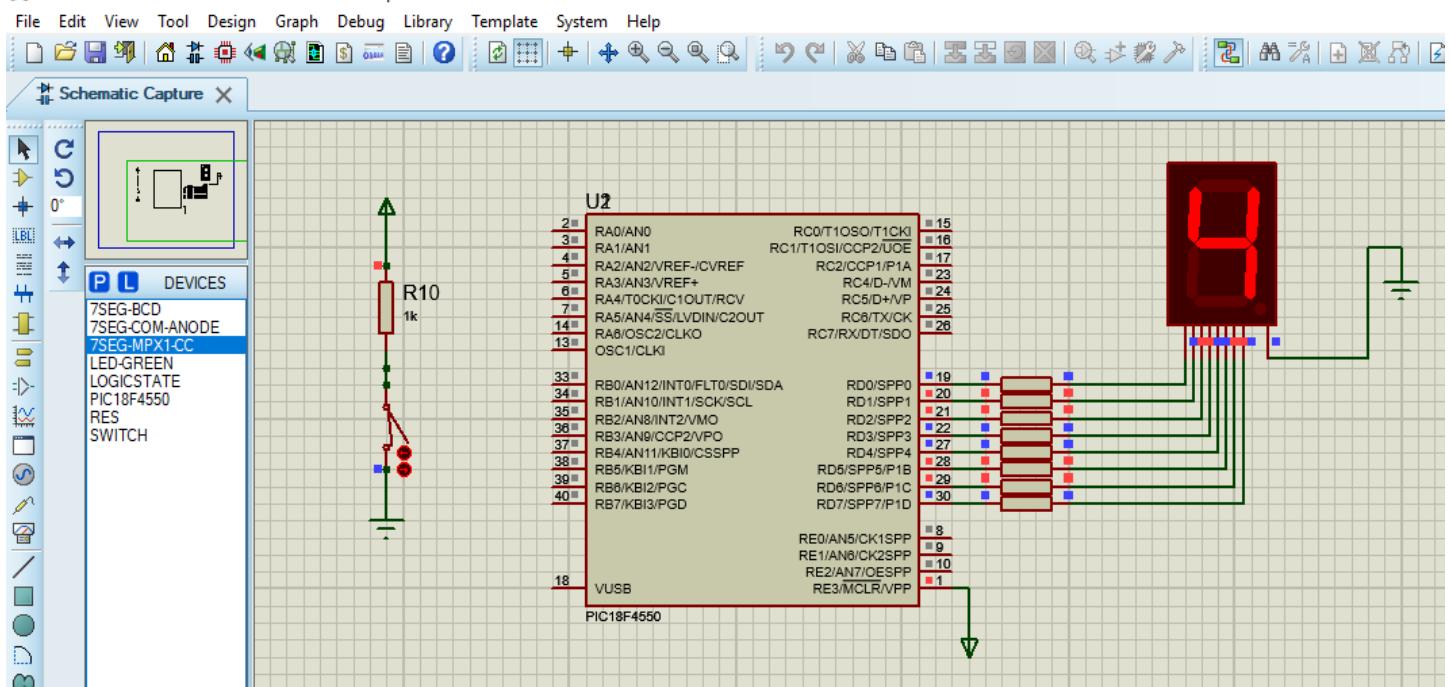


Figura 6.- Simulación del circuito realizada en Proteus

```

bra aa
ce movlw 0x39
movwf PORTD
call repite
btfs flags,0
bra be
dee movlw 0x5E
movwf PORTD
call repite
btfs flags,0
bra ce
ee movlw 0x79
movwf PORTD
call repite
btfs flags,0
bra dee
efe movlw 0x71
movwf PORTD
call repite
btfs flags,0
bra ee
;*****retardo***** movlw 0xff
        movwf cont,0
nada    nop
        decfsz cont,1,0
        bra nada
return
;*****repita***** movlw d'30' ;llama 25 veces a la rutina retardo
        movwf ciclo
llama   call retardo
        decfsz ciclo,f,0
        bra llama
return
;*****RSI***** bcf INTCON,INT0IF
        btg flags,0

```

Address	SFR Name
F64	SPPEPS
F65	SPCON
F66	UFRM
F66	UFRML
F67	UFRMH
F68	UIR
F69	UIE
F6A	UEIR
F6B	UEIE
F6C	USTAT
F6D	UCON
F6E	UCFG
F6F	UEPO
F70	UEP1
F71	UEP2
F72	UEP3
F73	UEP4
F74	UEP5
F75	UEP6
F76	UEP7
F77	UEP8
F78	UEP9
F79	UEP10
F7A	UEP11
F7B	UEP12
F7C	UEP13
F7D	UEP14
F7E	UEP15
F7F	PORTA
F80	PORTB
F81	PORTC
F82	PORTD
F83	PORTE

Figura 7.- Simulación del código Fuente en el Programa MPLAB IDE v8.92

Código Fuente

El siguiente código consiste en el control del display de 7 segmentos, el cual se basa solamente en enviar cada código por el puerto D y llamar a la rutina repite para que permanezca encendido el tiempo que dure la misma, esto se logra con las instrucciones ‘movlw 0xC0’ (código del cero) ‘movwf PORTD,0’ y ‘call repite’. De esa forma, las tres instrucciones anteriores se repiten diez veces, empezando por el código del cero y terminando con el código del nueve, después de esto se regresa el programa nuevamente al código del cero, con la instrucción de salto incondicional ‘bra cero’.

```
LIST P=18F4550 ;directiva para definir el procesador
#include <P18F4550.INC> ;definiciones de variables específicas del procesador
;
;***** ;Bits de configuración
CONFIG FOSC = INTOSC_XT ;Oscilador INT usado por el uC , XT
CONFIG BOR = OFF ;BROWNOUT RESET DESHABILITADO
CONFIG PWRT = ON ;PWR UP Timer habilitado
CONFIG WDT = OFF ;Temporizador vigila apagado
CONFIG MCLRE= ON ;Reset APAGADO
CONFIG PBADEN=OFF
CONFIG LVP = OFF
;*****CBLOCK

;Definiciones de variables
CBLOCK    0x000      ;ejemplo de definición de variables en RAM de acceso
cont
ciclo
flags
ENDC      ;fin del bloque de constantes ;*****
***** ;*****
***** ;Reset vector
ORG 0x0000
bra inicio
ORG 8
bra RSI
;Inicio del programa principal
inicio    bcf OSCCON,IRCF2,0
          bsf  OSCCON,IRCF0,0 ;Oscilador interno a125 kHz

          movlw  0x0F
          movwf  ADCON1,0 ;Puertos Digitales
```

```

        movlw 0x90 ;Bits del registro intcon
        movwf INTCON
        ;clrf PORTD,0
        clrf TRISD,0;Declara Puerto D como salida
        bsf flags,0
;*****
cero    movlw 0x3F
        movwf PORTD
        call repite
        btfss flags,0
        bra efe
uno    movlw 0x06
        movwf PORTD
        call repite
        btfss flags,0
        bra cero
dos    movlw 0x5B
        movwf PORTD
        call repite
        btfss flags,0
        bra uno
tres   movlw 0x4F
        movwf PORTD
        call repite
        btfss flags,0
        bra dos
cuatro movlw 0x66
        movwf PORTD
        call repite
        btfss flags,0
        bra tres
cinco  movlw 0x6D
        movwf PORTD
        call repite
        btfss flags,0
        bra cuatro
seis   movlw 0x7D
        movwf PORTD
        call repite
        btfss flags,0
        bra cinco
siete  movlw 0x07
        movwf PORTD
        call repite
        btfss flags,0

```

```

bra seis
ocho    movlw 0x7F
        movwf PORTD
        call repite
        btfss flags,0
        bra siete
nueve   movlw 0x67
        movwf PORTD
        call repite
        btfss flags,0
        bra ocho
aa       movlw 0x77
        movwf PORTD
        call repite
        btfss flags,0
        bra nueve
be       movlw 0x7C
        movwf PORTD
        call repite
        btfss flags,0
        bra aa
ce       movlw 0x39
        movwf PORTD
        call repite
        btfss flags,0
        bra be
dee     movlw 0x5E
        movwf PORTD
        call repite
        btfss flags,0
        bra ce
ee       movlw 0x79
        movwf PORTD
        call repite
        btfss flags,0
        bra dee
efe     movlw 0x71
        movwf PORTD
        call repite
        btfss flags,0
        bra ee
;*****
retardo  movlw 0xff
        movwf cont,0
nada    nop

```

```

decfsz cont,1,0
bra nada
return
;*****
repite    movlw d'30' ;llama 25 veces a la rutina retardo
          movwf ciclo
llama     call retardo
          decfsz ciclo,F,0
          bra llama
          return
;*****
RSI bcf  INTCON,INT0IF
btg  flags,0
retfie
END

```

CONCLUSIONES

En la presente práctica se realizó el contador decimal con el microcontrolador P18F4550 programable, que al presionar push button se generaba el conteo de 0 a 9. De esto modo, comprobando con la elaboración del circuito la veracidad del código con retardo por temporizador Timer0. Tomando en cuenta que la interrupción TMRO se genera cuando TMRO registrar desbordamientos de FFh a 00h en modo de 8 bits. Logrando así un conteo fluido y ordenado de los dígitos del 0 al 9.

PRÁCTICA 5 CONTADOR DE 0 A 9 ASCENDENTE Y DESCENDENTE CON TIMER PIC18F4550

OBJETIVO

Conocimiento y programación del puerto D del microcontrolador PIC18F4550, así como manejo de tablas y rutinas de retardo por programa, utilizando el Timer0 en modo de temporizador. Realizar la aplicación de un contador decimal ascendente/descendente de los dígitos del 0 al 9, por selección a través de la terminal RB0 del puerto B. Implementar el circuito basado en el microcontrolador en la base de prácticas protoboard.

MARCO TEÓRICO

Prioridad de las interrupciones:

El bit IPEN del registro RCON permite activar el sistema de prioridades:

- IPEN a '0': todas las interrupciones tienen la misma prioridad y se vectorizan en la dirección 0008H. Para habilitar globalmente las interrupciones del grupo general de interrupciones debe ponerse a '1' el GIE del registro INTCON. Para habilitar globalmente las interrupciones del bloque de interrupciones de periféricos deben ponerse a '1' los bits GIE y PEIE del registro INTCON. En este caso, ninguna interrupción puede interrumpir el tratamiento de otra interrupción.
- IPEN a '1': las interrupciones pueden configurarse con dos niveles de prioridad en función del bit de prioridad de interrupción (excepto la interrupción externa 0 que siempre tendrá nivel de prioridad alto):
 - Bit de prioridad de interrupción a '0' => Prioridad baja. Las interrupciones de prioridad baja se vectorizan en la dirección 0018H. Para habilitar globalmente las interrupciones de prioridad baja hay que poner a '1' el bit GIEL del registro INTCON.
 - Bit de prioridad de interrupción a '1' => Prioridad alta. Las interrupciones de prioridad alta se vectorizan en la dirección 0008H. Para habilitar globalmente las interrupciones de prioridad alta hay que poner a '1' el bit GIEH del registro INTCON.

Una interrupción de alta prioridad puede interrumpir el tratamiento de una interrupción de baja prioridad. Sin embargo las interrupciones de baja prioridad no pueden interrumpir a las interrupciones de alta prioridad.

Por otra parte las interrupciones de baja prioridad no pueden interrumpirse entre ellas. Lo mismo ocurre con las interrupciones de alta prioridad.

NOTA: Además de ser habilitadas globalmente, las interrupciones deben ser habilitadas a nivel particular mediante su bit de habilitación de interrupción.

Pasos a seguir para trabajar con interrupciones:

1. Configurar el dispositivo asociado a la interrupción (Temporizadores, EUSART, convertidor A/D, etc).

2. Habilitar a nivel global las interrupciones del grupo correspondiente mediante los bits GIE/GIEH y PEIE/GIEL del registro INTCON.
3. Habilitar a nivel individual la interrupción mediante el bit de habilitación.
4. Establecer la prioridad de la interrupción mediante el bit de prioridad (solo si previamente se ha activado el sistema de prioridad de interrupciones, IPEN='1')
5. En la dirección de vectorización correspondiente (0008H o 0018H, según el caso) añadir el código de tratamiento de la interrupción que debe incluir:
 - Identificación de la interrupción: dado que varias interrupciones pueden vectorizarse en la misma dirección, se deben comprobar los flags de las interrupciones habilitadas para saber cual de ellas ha provocado el salto a la dirección de vectorización.
 - Borrado del flag de interrupción: el flag de interrupción debe ser borrado por software una vez la interrupción ha sido identificada. De esta forma se evita que cuando finalice la rutina de tratamiento se vuelva a procesar la misma interrupción.
 - Código de procesado de la interrupción: por último hay que escribir el código de tratamiento de la interrupción que dependerá de cada aplicación.

El código de tratamiento de la interrupción siempre debe terminar con la instrucción RETFIE.

Proceso de tratamiento de una interrupción:

1. Se produce la condición de disparo de la interrupción (desbordamiento del temporizador, recepción de un dato en la EUSART, etc).
2. Si la interrupción está habilitada, el uC almacena en la pila la dirección de la instrucción que debía haberse ejecutado a continuación y escribe en PC la dirección de vectorización de la interrupción. Además el uC pone el bit de habilitación global de interrupciones (GIE, GIEH o GIEL según corresponda) a '0' para que ninguna interrupción del mismo nivel que la se está tratando pueda interrumpir el proceso.
3. Se ejecuta el código de tratamiento de la interrupción que deberá hallarse en la dirección de la vectorización (se comprueba que interrupción ha generado el salto a la dirección de vectorización, se borra el flag de interrupción correspondiente y se ejecuta el código asociado a dicha interrupción).
4. La última instrucción del código de tratamiento de la interrupción es RETFIE. Cuando se ejecuta esta instrucción, el uC vuelve a poner a '1' el bit de habilitación global de interrupciones (GIE, GIEH o GIEL) y escribe en el PC la dirección que había almacenado en la pila de forma que se continué con la ejecución del programa a partir del punto en el que había sido interrumpido.

Interrupciones del puerto B:

Interrupción externa 0:

- Flag de interrupción: bit INTOIF (INTCON)
- Bit de habilitación: bit INTOIE (INTCON)
- Selección de flanco: bit INTEDG0 (INTCON2): '0'>fl. bajada/'1'> fl. subida
- Prioridad: la interrupción 0 siempre tiene prioridad alta

Si se produce un flanco en la línea RB0 (flanco de subida o de baja dependiendo del valor del bit INTEDG0) se pone el flag INTOIF a '1'.

Si el bit de habilitación INTOIE está a '1' y las interrupciones están habilitadas a nivel global se genera una interrupción y el uC pasa a ejecutar el código situado a partir de la posición 0008H.

Interrupción externa 1:

- Flag de interrupción: bit INTCON3.INT1IF
- Bit de habilitación: bit INTCON3.INT1IE
- Selección de flanco: bit INTCON2.INTEDG1: '0'->fl. bajada/'1'-> fl. subida
- Prioridad: bit INT1IP (INTCON3): '0'->pr. baja/'1'->pr. Alta

Si se produce un flanco en la línea RB1 (flanco de subida o de baja dependiendo del valor del bit INTEDG1) se pone el flag INT1IF a '1'.

Si el bit de habilitación INT1IE está a '1' y las interrupciones están habilitadas a nivel global se genera una interrupción y el μC pasa a ejecutar el código situado a partir de la posición 0008H o 0018H (según el nivel de prioridad establecido).

Interrupción externa 2:

- Flag de interrupción: bit INTCON3.INT2IF
- Bit de habilitación: bit INTCON3.INT2IE
- Selección de flanco: bit INTCON2.INTEDG2: '0'->fl. bajada/'1'-> fl. subida
- Prioridad: bit INTCON3.INT2IP: '0'->pr. baja/'1'->pr. alta

Si se produce un flanco en la línea RB2 (flanco de subida o de baja dependiendo del valor del bit INTEDG2) se pone el flag INT2IF a '1'.

Si el bit de habilitación INT2IE está a '1' y las interrupciones están habilitadas a nivel global se genera una interrupción y el μC pasa a ejecutar el código situado a partir de la posición 0008H o 0018H (según el nivel de prioridad establecido).

Interrupción por cambio en pines RB7...RB4:

- Flag de interrupción: bit RBIF (INTCON)
- Bit de habilitación: bit RBIE (INTCON)
- Prioridad: bit RBIP (INTCON2): '0'->pr. baja/'1'->pr. Alta

Solo puede asociarse a las líneas RB7..RB4 que hayan sido configuradas como entrada. Cada vez que se produce un cambio en estas líneas se pone a '1' el flag RBIF. Si el bit de habilitación RBIE está a '1' y las interrupciones están habilitadas a nivel global se genera una interrupción y el uC pasa a ejecutar el código situado a partir de la posición 0008H ó 0018H (según el nivel de prioridad establecido).

Si se utiliza esta interrupción no es aconsejable hacer “**polling**” del puerto B. Antes de salirse de la rutina de tratamiento de la interrupción hay que hacer lo siguiente:

- Acceder al puerto B con cualquier instrucción excepto MOVFF, para deshacer la desigualdad entre valor antiguo y nuevo valor.
- Poner a '0' el bit RBIF.

REGISTRO INTCON

	L/E-0	L/E-0	L/E-0	L/E-0	L/E-0	L/E-0	L/E-0	L/E-0
INTCON	GIE/GIEH	PEIE/GIEL	TMROIE	INTOIE	RBIE	TMROIF	INTOIF	RBIF

- **GIE/GIEH:** Bit de habilitación global de las interrupciones. Su función depende del valor de IPEN:
 - IPEN='0':**
 - * GIE='0': Interrupciones deshabilitadas a nivel global
 - * GIE='1': Interrupciones habilitadas a nivel global
 - IPEN='1':**
 - * GIEH='0': Interrupciones de alta prioridad deshabilitadas a nivel global
 - * GIEH='1': Interrupciones de alta prioridad habilitadas a nivel global
- **PEIE/GIEL:** Bit de habilitación global de las interrupciones de periféricos. Su función depende del valor de IPEN:
 - IPEN='0':**
 - * PEIE='0': Interrupciones de periféricos deshabilitadas a nivel global
 - * PEIE='1': Interrupciones de periféricos habilitadas a nivel global
 - IPEN='1':**
 - * GIEL='0': Interrupciones de baja prioridad deshabilitadas a nivel global
 - * GIEL='1': Interrupciones de baja prioridad habilitadas a nivel global

REGISTRO INTCON2

	L/E-1	L/E-1	L/E-1	L/E-1	-0	L/E-1	-0	L/E-1
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	-	TMROIP	-	RBIP

- **RBPU:** Bit de habilitación de las resistencias de pull-up del Puerto B:
 - * RBPU='0': Las resistencias de pull-up del Puerto B están activadas
 - * RBPU='1': Las resistencias de pull-up del Puerto B están desactivadas
- **INTEDG0:** Bit de selección de flanco de la interrupción externa 0:
 - * INTEDG0='0': La interrupción externa 0 se dispara por flanco de bajada
 - * INTEDG0='1': La interrupción externa 0 se dispara por flanco de subida
- **INTEDG1:** Bit de selección de flanco de la interrupción externa 1:
 - * INTEDG1='0': La interrupción externa 1 se dispara por flanco de bajada
 - * INTEDG1='1': La interrupción externa 1 se dispara por flanco de subida
- **INTEDG2:** Bit de selección de flanco de la interrupción externa 2:
 - * INTEDG2='0': La interrupción externa 2 se dispara por flanco de bajada
 - * INTEDG2='1': La interrupción externa 2 se dispara por flanco de subida
- **TMROIP:** Bit de selección de prioridad de la interrupción del Temporizador 0
 - * TMROIP='0': Prioridad baja para la interrupción del Temporizador 0
 - * TMROIP='1': Prioridad alta para la interrupción del Temporizador 0
- **RBIP:** Bit de selección de prioridad de la interrupción por cambio en el Puerto B
 - * RBIP='0': Prioridad baja para la interrupción por cambio en el Puerto B
 - * RBIP='1': Prioridad alta para la interrupción por cambio en el Puerto B

Tablas Por El Método Del 'Goto Calculado'

Para crear tablas de búsqueda en memoria de programa con los microcontroladores PIC18, existen dos métodos:

- Goto calculado
- Instrucciones específicas de lectura de tabla

En este ejemplo, veremos el primer método. Para crear un goto calculado, es necesario primero cargar el valor de desplazamiento en el registro w y sumar un valor de desplazamiento (offset) al contador de programa, seguida de un grupo de instrucciones `retlw nn`.

El valor del desplazamiento indica el número de bytes que debe avanzar el contador de programa y debe ser un múltiplo de dos. Este método recibe el nombre de goto calculado, porque igual que una instrucción goto, realiza una ramificación, o equivalentemente un salto; pero se logra sumando un valor al contador de programa.

Debido a que únicamente se suma el desplazamiento al PCL, debe tenerse cuidado de que el grupo de instrucciones `retlw nn`, no cruce una página de 256 bytes, ya que el PCL es de 8 bits, y cuando cambie de FF a 00, no generará un acarreo hacia el registro PCH, ocasionando un salto fuera del grupo de instrucciones `retlw nn`, afectando la secuencia del programa, que percibiremos como un funcionamiento errático del mismo.

```
tabla    rlcf    indice,W,0 ;multiplica índice por 2
        addwf   PCL,F,0 ;ajusta el PCL de acuerdo al valor del índice
        retlw   0xC0 ;código del cero
        retlw   0xf9 ;código del uno
        retlw   0xA4 ;código del dos
        retlw   0xb0 ;código del tres
        retlw   0x99 ;código del cuatro
        retlw   0x92 ;código del cinco
        retlw   0x82 ;código del seis
        retlw   0xb8 ;código del siete
        retlw   0x80 ;código del ocho
        retlw   0x98 ;código del nueve
```

DESARROLLO

Presentamos a continuación los materiales a utilizar en la presente práctica:

- 8 Leds
- 8 Resistencias de 330 Ohms
- 1 Display de 7 segmentos
- Cables
- Protoboard
- PIC18F4550 µvva
- Bus de Salida para PIC18F
- Ordenador para ejecutar nuestro programa

Simulación y Ensamblaje del Circuito

A continuación, mostramos el Diagrama el Circuito realizado en Proteus así como el ensamblaje del mismo en Protoboard:

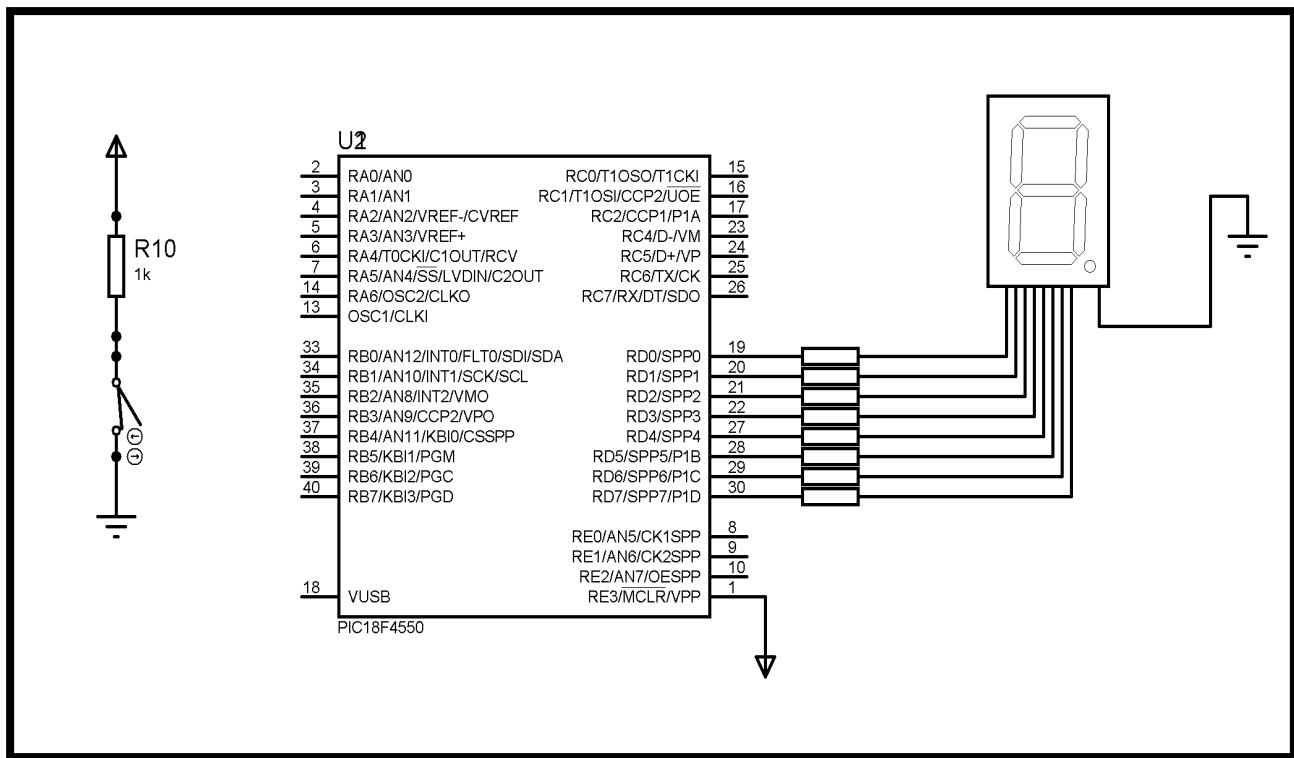


Figura 5.- Diagrama del circuito realizada en Proteus

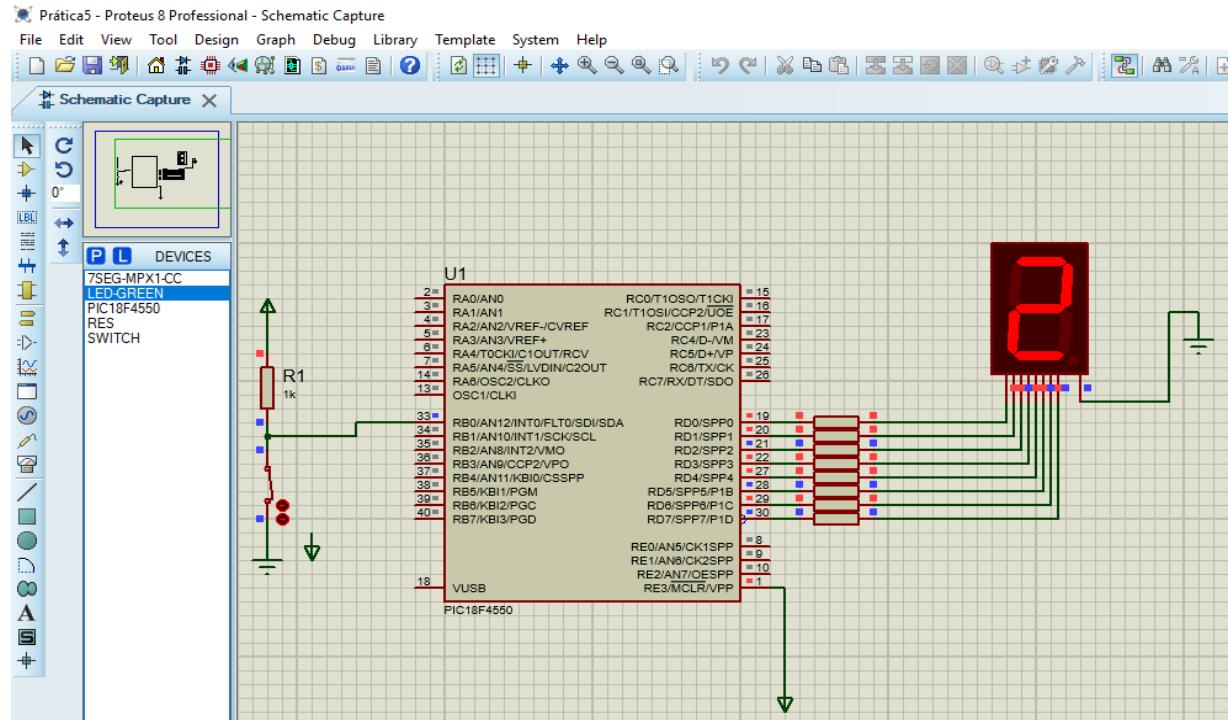


Figura 6.- Simulación del circuito realizada en Proteus

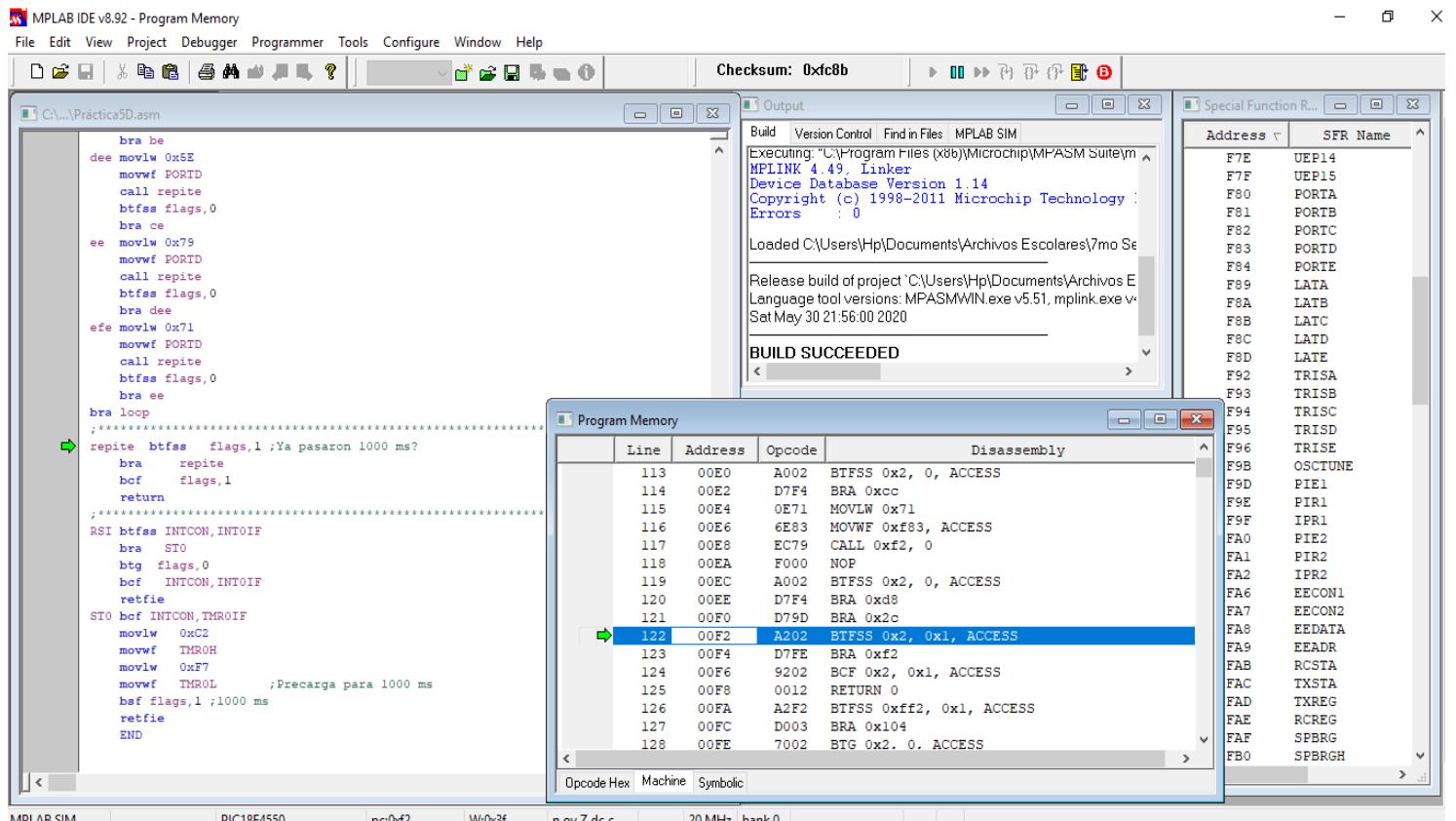


Figura 8.- Simulación del código Fuente en el Programa MPLAB IDE v8.92

Código Fuente

El programa que presentamos a continuación consiste en un contador ascendente/descendente, aquí la duración de cada número exhibido es de exactamente 500 ms. La rutina ‘repite’ ahora es controlada por el timer 0. Configuramos el oscilador interno del µc a 4 MHz, de acuerdo a los cálculos realizados. Con la instrucción ‘flags’ definimos la dirección 0 como registro de banderas y con ‘INTCON,TMROIF,0’ apagamos bandera timer0.

```
LIST P=18F4550 ;directiva para definir el procesador
#include <P18F4550.INC> ;definiciones de variables específicas del procesador
;
;***** ;Bits de configuración
CONFIG FOSC = INTOSC_XT           ;Oscilador INT usado por el uC , XT
T usado por el USB
CONFIG BOR  = OFF    ;BROWNOUT RESET DESHABILITADO
CONFIG PWRT = ON     ;PWR UP Timer habilitado
CONFIG WDT  = OFF    ;Temporizador vigila apagado
CONFIG MCLRE= OFF   ;Reset Encendido
CONFIG PBADEN=OFF
CONFIG LVP   = OFF
;*****CBLOCK
;
;Definiciones de variables
CBLOCK    0x000      ;ejemplo de definición de variables en RAM de acceso
cont
ciclo
flags
ENDC      ;fin del bloque de constantes ;*****
;*****
;*****
;Reset vector
ORG 0x0000
bra inicio
ORG 0x08
bra RSI
;Inicio del programa principal
inicio    bsf OSCCON,IRCF2,0
          bsf OSCCON,IRCF1,0
          bcf OSCCON,IRCF0,0    ;Oscilador interno a 4 MHz
          movlw  0x0F
          movwf  ADCON1,0 ;Puertos Digitales
          movlw  0xB0
```

```

        movwf  INTCON      ;ACTIVOinterrupciones INT0, TMR0
        movlw  0x95
        movwf  T0CON       ;Timer 16 bits, preescaler 64
        movlw  0xF7
        movwf  TMR0L       ;Precarga para 1000 ms
        movlw  0xC2
        movwf  TMR0H
        clrf  TRISD,0;Declara Puerto D como salida
        clrf  PORTD
        bcf   flags,1
        bsf   flags,0
;*****
loop
    cero  movlw 0x3F
        movwf PORTD
        movlw 0x3F
        movwf PORTD
        call  repite
        btfss flags,0
        bra  efe
    uno   movlw 0x06
        movwf PORTD
        call  repite
        btfss flags,0
        bra  cero
    dos   movlw 0x5B
        movwf PORTD
        call  repite
        btfss flags,0
        bra  uno
    tres  movlw 0x4F
        movwf PORTD
        call  repite
        btfss flags,0
        bra  dos
    cuatro  movlw 0x66
        movwf PORTD
        call  repite
        btfss flags,0
        bra  tres
    cinco  movlw 0x6D
        movwf PORTD
        call  repite
        btfss flags,0
        bra  cuatro

```

```
seis    movlw 0x7D
        movwf PORTD
        call repite
        btfss flags,0
        bra cinco
siete   movlw 0x07
        movwf PORTD
        call repite
        btfss flags,0
        bra seis
ocho    movlw 0x7F
        movwf PORTD
        call repite
        btfss flags,0
        bra siete
nueve   movlw 0x67
        movwf PORTD
        call repite
        btfss flags,0
        bra ocho
aa      movlw 0x77
        movwf PORTD
        call repite
        btfss flags,0
        bra nueve
be      movlw 0x7C
        movwf PORTD
        call repite
        btfss flags,0
        bra aa
ce      movlw 0x39
        movwf PORTD
        call repite
        btfss flags,0
        bra be
dee    movlw 0x5E
        movwf PORTD
        call repite
        btfss flags,0
        bra ce
ee      movlw 0x79
        movwf PORTD
        call repite
        btfss flags,0
        bra dee
```

```

efe movlw 0x71
    movwf PORTD
    call repite
    btfss flags,0
    bra ee
bra loop
;*****
repite btfss flags,1 ;Ya pasaron 1000 ms?
    bra repite
    bcf flags,1
    return
;*****
RSI btfss INTCON,INT0IF
    bra ST0
    btg flags,0
    bcf INTCON,INT0IF
    retfie
ST0 bcf INTCON,TMR0IF
    movlw 0xC2
    movwf TMR0H
    movlw 0xF7
    movwf TMR0L      ;Precarga para 1000 ms
    bsf flags,1 ;1000 ms
    retfie
END

```

CONCLUSIONES

En la presente práctica se realizó el contador decimal con el microcontrolador P18F4550 programable, ascendente y descendente de los dígitos del 0 al 9, logrando un conteo fluido de dichos dígitos. Además de haber incorporado la sub rutina Int0 implementando dicha interrupción a través de un push button con el que podemos cambiar la orientación del conteo, ya sea de forma progresiva o regresiva, además de cumplir con un tiempo de duración en cada número exhibido de 500 ms.

PRÁCTICA 6 CONTADOR DE 0 A 9999 CON TIMER EN PIC18F4550

OBJETIVO

Reforzar el conocimiento y programación de los puertos de entrada y salida digital (puertos A, B y D) del microcontrolador PIC18F4550, así como la utilización del Timer0 en modo de temporizador. Se realiza la aplicación de un contador decimal ascendente con retardo por temporizador (Timer0) de 4 dígitos del 0 al 9999.

MARCO TEÓRICO

Interrupción del Temporizador 0:

- Flag de interrupción: bit TMROIF (INTCON)
- Bit de habilitación: bit TMROIE (INTCON)
- Prioridad: bit TMROIP (INTCON2): '0'>pr. baja/'1'>pr. alta.

Si se produce el desbordamiento del Temporizador 0 se pone a '1' el flag TMROIF. Si el bit de habilitación TMROIE está a '1' y las interrupciones están habilitadas a nivel global se genera una interrupción y el uC pasa a ejecutar el código situado a partir de la posición 0008H o 0018H (según el nivel de prioridad establecido).

En modo Temporizador: El TMRO se incrementa con cada **Ciclo de Instrucción (Fosc/4)** y cuando el registro se desborde provocará una interrupción.

En modo Contador: El TMRO se incrementará con cada pulso que ingrese por el pin **RA4/T0CKI**, y por supuesto, cuando se desborde producirá la interrupción. **TOSE** es el Bit4 del Registro TOCON, en él se selecciona el flanco con el cual se incrementará el TMRO cuando haya un pulso por RA4.

Además, el micro dispone de dos temporizadores, el TMRO y **WDT (Watchdog)**. El primero es el que estamos tratando en esta sección, el segundo es el *perro guardián*, lo que hace es vigilar cada cierto tiempo que el micro no se quede colgado, por ejemplo cuando se queda detenido en un bucle infinito o en una larga espera de un acontecimiento que no se produce, entonces actúa resemando al micro.

El Prescalar es un previsor de frecuencia que se utiliza comúnmente para programar tiempos largos y se puede aplicarlo al TMRO o al WDT, esto se configura en PSA Bit3 del registro TOCON.

Temporización con el registro TMRO

El tiempo empleado en una temporización se puede calcular a partir de **un ciclo de instrucción** (es decir 1 instrucción por cada microsegundo, al trabajar con un oscilador de 4 Mhz), también se necesita el valor del **Divisor de Frecuencia** y finalmente con **el complemento del valor cargado en TMRO** (es decir 256-TMRO, si se trabaja en modo de 8 bits), la ecuación que permite realizar el cálculo es la que sigue.

Tiempo = 4 * Toscilador * (256 - TMRO) * Rango Divisor de Frecuencia

De donde el 4 indica que el micro ejecuta una instrucción normal en 4 ciclos de reloj.

Ejemplo: Se desea una temporización de 10 ms, trabajando con un oscilador de 4 Mhz.

El primer paso, es seleccionar el prescalar, es decir, PS2, PS1, PS0 = 1, 1, 1.

Por tanto, el Rango del divisor de frecuencia es 256.

El siguiente paso es calcular el valor que se debe cargar en TMRO...???

256 - TMRO= Tiempo / (4*Toscilador * Rango Divisor de Frecuencia).

256 - TMRO = 10000 us / (4 * 0.25 us * 256) = 39,0625 ciclos ≈ 39 ciclos.

Eso significa que en TMRO se debe cargar 256-39=217 y a partir de allí el TMRO contará los 39 ciclos que faltan para desbordarse y producir la interrupción, y el tiempo que tardará en hacerlo es aproximadamente 10000 us, o sea 10 ms.

Temporizador 1

Características fundamentales:

- Configurable como temporizador/contador de 16 bits
- Dispone de un oscilador propio que puede funcionar como:
 - Señal de reloj del temporizador 1.
 - Señal de reloj del μC en modos de bajo consumo.
 - Pre-escalador de 3 bits programable.
- Interrupción por desbordamiento

Existen dos modos de acceder al registro TMR1:

- RD16 (T1CON)=‘0’: se accede a TMR1L y TMR1H como dos registros independientes.
- RD16 (T1CON)=‘1’: la parte alta de TMR1 no es accesible directamente. Se accede a ella a través de TMR1H que funciona como un búfer:
 - Cuando se lee el valor de TMR1L, el valor de la parte alta de TMR1 pasa al registro TMR1H. Para leer el valor de TMR1 primero leemos TMR1L y luego leemos TMR1H.
 - Cuando se escribe en TMR1L, el valor de TMR1H pasa a la parte alta de TMR1. Para escribir en TMR1 primero escribimos la parte alta en TMR1H y luego escribimos la parte baja en TMR1L.

Interrupción del Temporizador 1:

- Flag de interrupción: bit TMR1IF (PIR1)
- Bit de habilitación: bit TMR1IE (PIE1)
- Prioridad: bit TMR1IP (IPR1): ‘0’->pr. baja/‘1’->pr. Alta

Si se produce el desbordamiento del Temporizador 1 se pone a ‘1’ el flag TMR1IF. Si el bit de habilitación TMR1IE está a ‘1’ y las interrupciones de periféricos están habilitadas a nivel global se genera una interrupción y el μC pasa a ejecutar el código situado a partir de la posición 0008H o 0018H (según el nivel de prioridad establecido).

Temporizador 2:

Características fundamentales:

- Temporizador de 8 bits (registro TMR2)
- Registro de periodo PR2
- Pre-escalar de 2 bits programable (1:1, 1:4, 1:16)
- Post-escalar de 4 bits (1:1...1:16)
- Interrupción por igualdad entre TMR2 y PR2
- Se puede utilizar junto con los módulos CCP y ECCP
- Se puede utilizar como señal de reloj del módulo MSSP en modo SPI

Los registros TMR2 y PR2 son de lectura escritura. Los contadores del pre-escalar y el post-escalar no son accesibles por el usuario. Se ponen a 0 automáticamente cuando:

- Se escribe en TMR2
- Se escribe en T2CON
- Se produce un reset

Funcionamiento del Temporizador 2:

El registro TMR2 se incrementa con cada pulso de reloj de una señal que se obtiene haciendo pasar la señal de frecuencia FOSC/4 por un pre-escalar. Los bits T2CKPS1 y T2CKPS0 permiten seleccionar si la frecuencia de la señal que incrementa TMR2 es de FOSC/4, FOSC/16 o FOSC/64.

Cuando el valor de TMR2 se iguala con el valor del registro de periodo PR2:

- Se reinicia el valor de TMR2
- Se activa la señal de salida del Temporizador 2

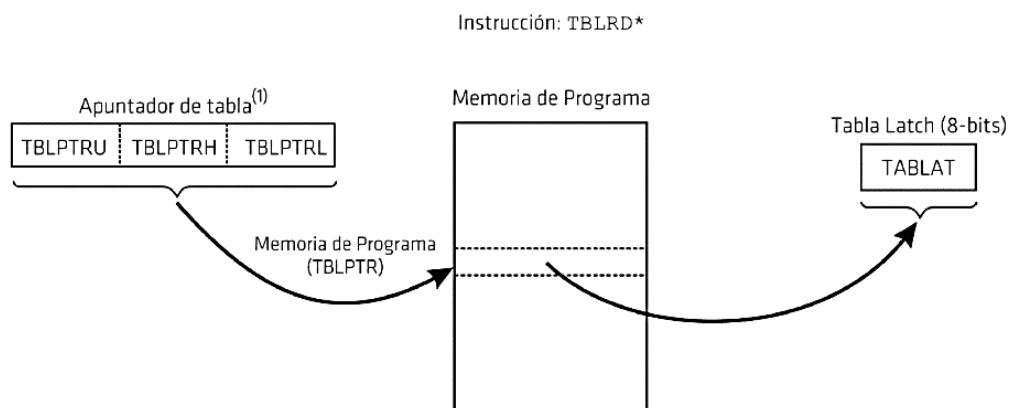
La señal de salida del Temporizador 2 puede utilizarse en el módulo CCP para generar señales PWM o en el módulo MSSP como señal de reloj del modo SPI. La señal de salida del Temporizador está conectada a un contador post-escalar que se incrementa cada vez que se produce la igualdad entre TMR2 y PR2. Cuando el contador post-escalar desborda se pone a '1' el flag de interrupción del Temporizador 2 (T2IF). El número de pulsos de la señal de salida del Temporizador 2 necesarios para provocar el desbordamiento del post-escalar puede configurarse de 1 a 16 en función de los T2OUTPS3..T2OUTPS0.

Tablas De Búsqueda En Memoria De Programa Usando Instrucciones De Lectura De Tabla

Ahora veremos el segundo método, que ya habíamos mencionado. Para tener un alcance suficiente, en los PIC18 que tengan mayor capacidad de memoria de programa, tenemos un apuntador de tabla de 24 bits, llamado TBLPTR (TaBLE PoinTeR) que está distribuido en tres registros de 8 bits:

- TBLPTRL
- TBLPTRH
- TBLPTRU

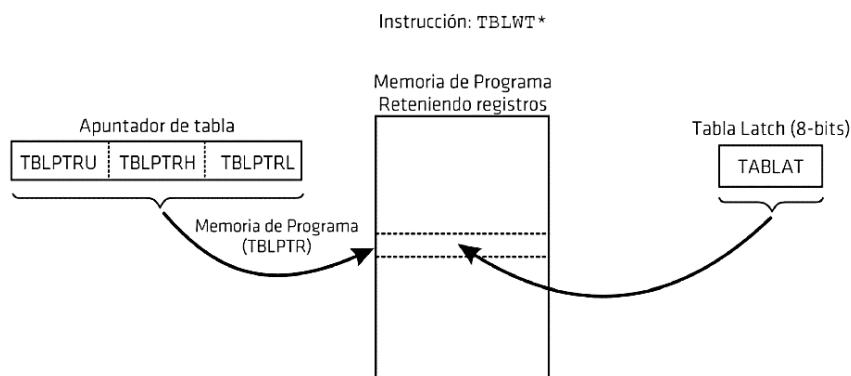
Apuntador de tabla parte superior TBLPTRU Apuntador de tabla parte alta TBLPTRH Apuntador de tabla parte baja TBLPTRL. También existe un registro llamado TABLAT (TABle LATch), para almacenar el dato leído, correspondiente a la dirección especificada por TBLPTR.



En la figura se muestra una operación de lectura de tabla TBLRD *, existen cuatro instrucciones de lectura de TABLA:

TBLRD *	Lectura de tabla
TBLRD **	Lectura de tabla con pos incremento del apuntador
TBLRD +*	Lectura de tabla con pre incremento del apuntador
TBLRD *-	Lectura de tabla con pos decremento del apuntador

También existen las instrucciones equivalentes para la escritura en memoria de tabla



DESARROLLO

Presentamos a continuación los materiales a utilizar en la presente práctica:

- 8 Resistencias de 330 Ohms
- 1 Display de 4 dígitos de 7 segmentos
- 4 Transistores
- Cables
- Protoboard
- PIC18F4550 µvva
- Bus de Salida para PIC18F
- Ordenador para ejecutar nuestro programa

Simulación y Ensamblaje del Circuito

A continuación, mostramos el Diagrama el Circuito realizado en Proteus así como el ensamblaje del mismo en Protoboard:

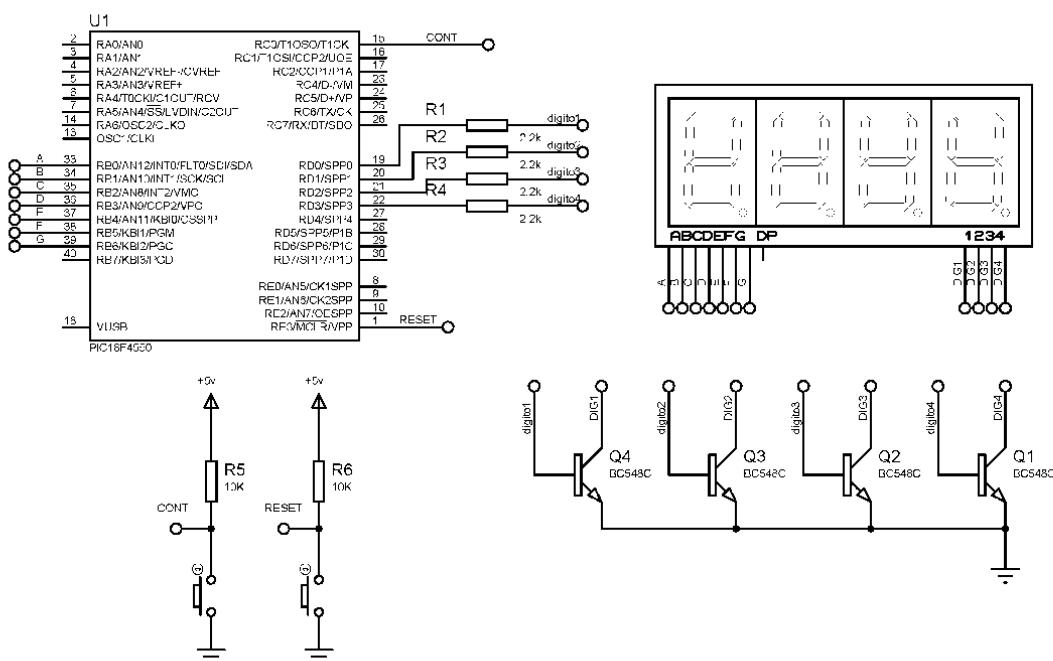


Figura 5.- Diagrama del circuito realizada en Proteus

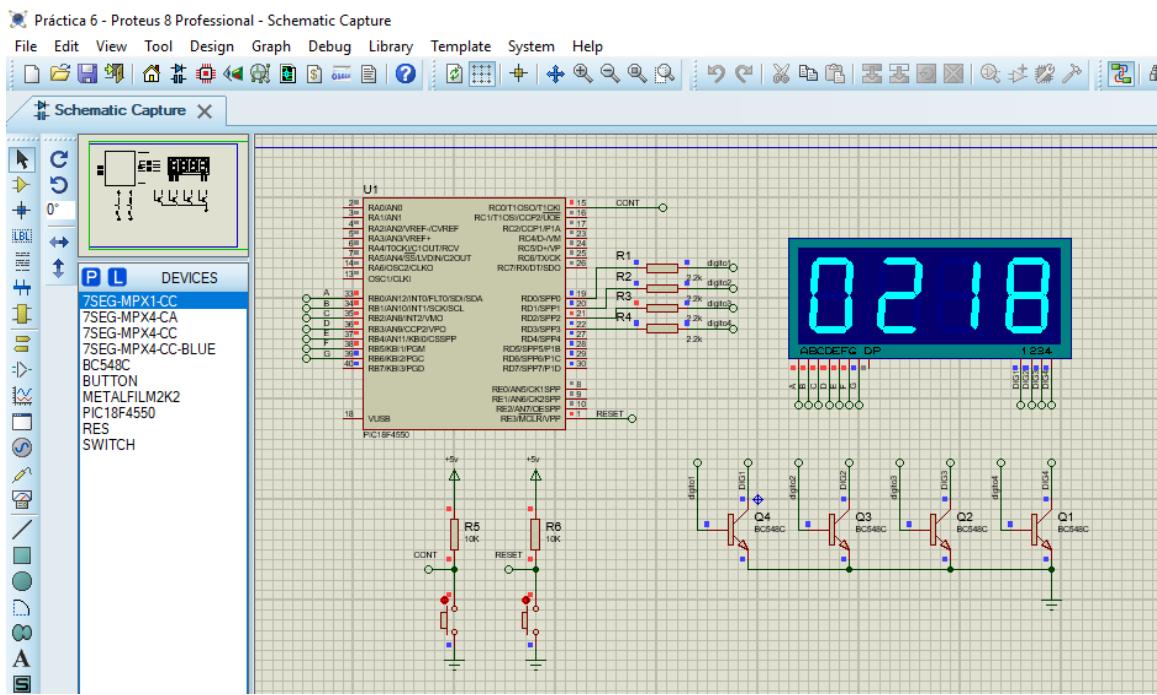


Figura 6.- Simulación del circuito realizada en Proteus

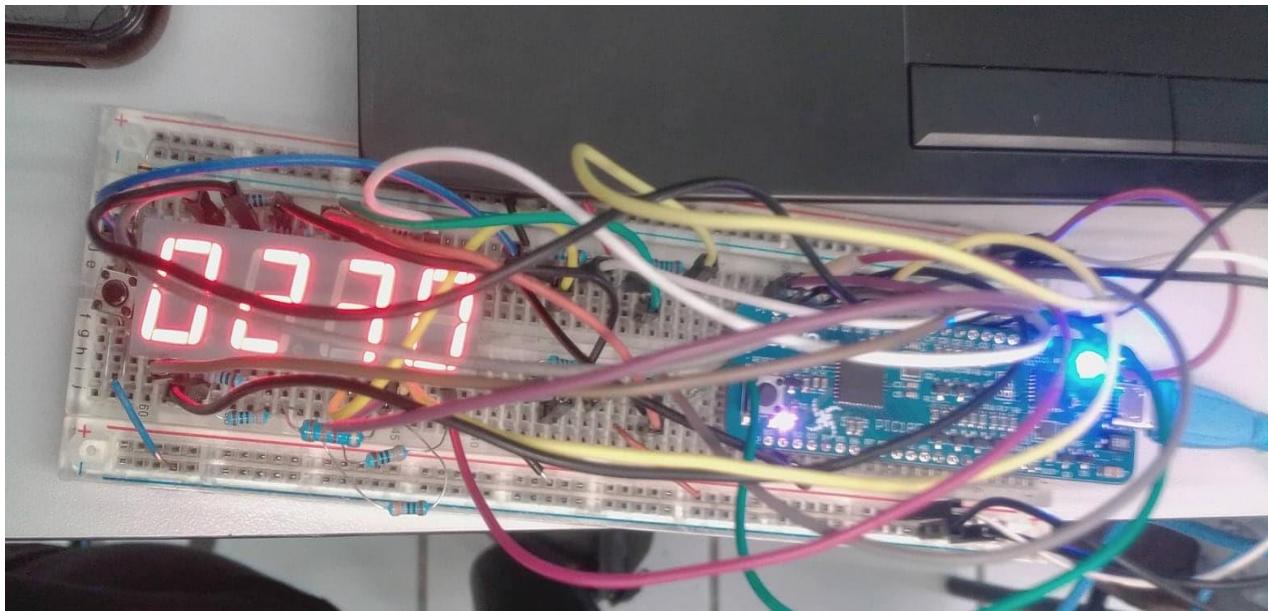


Figura 7.- Ensamblaje del circuito en protoboard con PIC18F4550

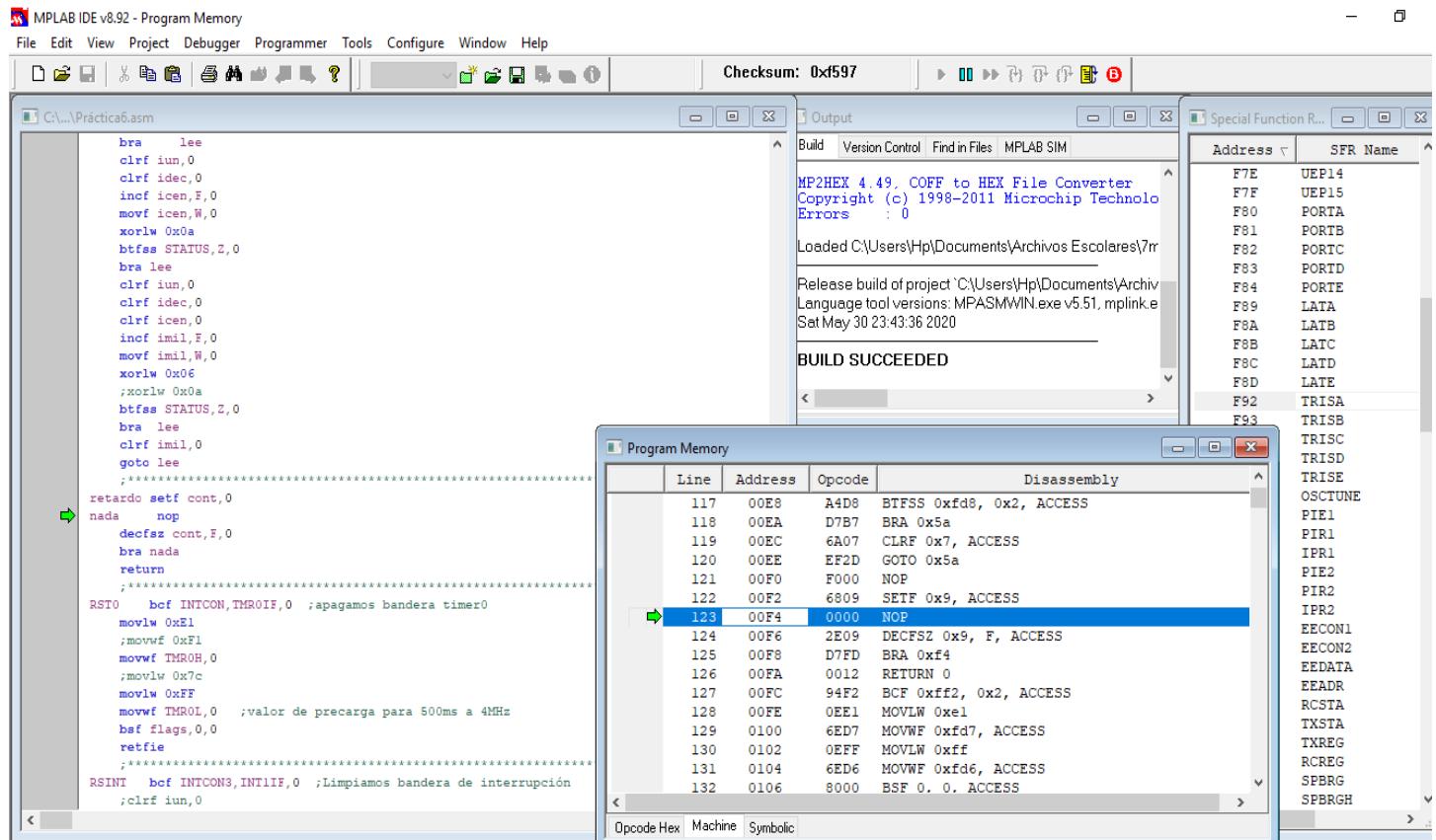


Figura 8.- Simulación del código Fuente en el Programa MPLAB IDE v8.92

Código Fuente

El siguiente programa que presentamos a continuación consiste en un contador de los dígitos de 0 al 9999 de forma ascendente, este mismo mantiene los índices dentro del valor adecuado, iun inicia en cero, y se incrementa exactamente cada 500 ms, hasta llegar a 99, una vez que iun tiene el valor de 9, en los siguientes 500 ms, iun se regresa a cero, pero a la vez se incrementa idec. Debido a que para encender los dígitos necesitamos los códigos de 7 segmentos, tenemos también dos registros que los almacenan: Cuni código de 7 segmentos de unidades y Cdec código de 7 segmentos de decenas. El programa se basa es cargar el código de siete segmentos en el PORTD, el tiempo que dura la rutina retardo, cada 500ms se incrementa iun (y cuando regresa a cero, se incrementa idec) y se actualiza el código de siete segmentos con las instrucciones.

```
;*****  
***  
LIST P=18F4550           ;directiva para definir el procesador  
#include <P18F4550.INC> ;definiciones de variables específicas del procesador  
  
;*****  
*** ;Bits de configuración  
CONFIG FOSC = INTOSC_XT    ;Oscilador INT usado por el uC , XT usado por el USB  
CONFIG BOR  = OFF          ;BROWNOUT RESET DESHABILITADO  
CONFIG PWRT = ON           ;PWR UP Timer habilitado  
CONFIG WDT  = OFF          ;Temporizador vigía apagado  
CONFIG MCLRE=OFF          ;Reset apagado  
CONFIG PBADEN=OFF  
CONFIG LVP   = OFF  
;*****  
***  
;Definiciones de variables  
CBLOCK 0x000   ;ejemplo de definición de variables en RAM de acceso  
flags      ;banderas  
iun       ;índice de unidades  
cuni      ;código de 7 segmentos de unidades  
idec      ;índice de decenas  
cdec      ;código de 7 segmentos de decenas  
icen  
ccen  
imil  
cmil  
cont  
ENDC      ;fin del bloque de constantes  
;*****  
***
```

```

;Reset vector
ORG 0x0000
bra inicio
org 0x08 ;vector de alta prioridad
bra RST0 ;ramifica servicio interrupción T0
org 0x18 ;vector de baja prioridad
bra RSINT
org 0x0020

inicio bsf OSCCON,IRCF2,0
bsf OSCCON,IRCF1,0
bcf OSCCON,IRCF0,0 ;Oscilador interno a 4 MHz

movlw 0x0F
movwf ADCON1,0 ;Puertos Digitales
clrf PORTD,0
clrf TRISD,0 ;Puerto D Configurado como salida
;movlw 0xE0
;movwf TRISC,0 ;RC0 y RC1 como salidas
clrf TRISC
clrf TRISA
clrf PORTC,0
movlw 0x95
movwf T0CON,0 ;timer 16 bits prescalerX64
movlw 0XE0
movwf INTCON,0 ;interrupciones TMR0,prioridad habilitada
bsf RCON,IPEN,0 ;habilitamos prioridades de interrupción
movlw 0xE1
movwf TMR0H,0
movlw 0x7c
movwf TMR0L,0 ;valor de precarga para 500ms a 4MHz
movlw 0x88
movwf INTCON3,0 ;habilitamos int1 en baja prioridad
clrf TBLPTRL,0
movlw 0x03
movwf TBLPTRH,0
clrf TBLPTRU,0 ;tblptr=0x000300
clrf iun,0
clrf idec,0 ;iniciamos en 0
clrf icen,0
clrf imil,0

lee movff iun,TBLPTRL ;ajusta apuntador
tblrd * ;lee la tabla sin modificar apuntador
movff TABLAT,cuni ;cuni tiene código 7 segmentos
movff idec,TBLPTRL ;ajusta apuntador
tblrd * ;lee la tabla sin modificar apuntador

```

```

movff TABLAT,cdec ;cdec tiene código 7 segmentos
movff icen,TBLPTRL
tblrd *
movff TABLAT,ccen
movff imil,TBLPTRL
tblrd *
movff TABLAT,cmil
loop    movlw 0x01
        movwf PORTA,0 ;encendemos display unidades
        movff cuni,PORTD
        call retardo
        movlw 0x02
        movwf PORTA,0 ;encendemos display decenas
        movff cdec,PORTD
        call retardo
        movlw 0x04
        movwf PORTA,0;encendemos display de las centenas
        movff ccen,PORTD
        call retardo
        movlw 0x08
        movwf PORTA,0;encendemos display de las centenas
        movff cmil,PORTD
        call retardo
        btfss flags,0,0
        bra loop
        bcf flags,0,0
        incf iun,F,0
        movf iun,W,0
        xorlw 0x03
        btfss STATUS,Z,0;verifica límite de tabla
        bra    lee
        clrf iun,0
        incf idec,F,0
        movf idec,W,0
        xorlw 0x03
        btfss STATUS,Z,0
        bra    lee
        clrf iun,0
        clrf idec,0
        incf icen,F,0
        movf icen,W,0
        xorlw 0x0a
        btfss STATUS,Z,0
        bra lee
        clrf iun,0

```

```

        clrf idec,0
        clrf icen,0
        incf imil,F,0
        movf imil,W,0
        xorlw 0x06
        ;xorlw 0x0a
        btfss STATUS,Z,0
        bra lee
        clrf imil,0
        goto lee
        ;*****
*****  

retardo setf cont,0
nada      nop
decfsz cont,F,0
bra nada
return
;*****  

*****  

RST0    bcf INTCON,TMR0IF,0 ;apagamos bandera timer0
        movlw 0xE1
        ;movwf 0xF1
        movwf TMR0H,0
        ;movlw 0x7c
        movlw 0xFF
        movwf TMR0L,0 ;valor de precarga para 500ms a 4MHz
        bsf flags,0,0
        retfie
        ;*****  

*****  

RSINT   bcf INTCON3,INT1IF,0 ;Limpiamos bandera de interrupción
        ;clrf iun,0
        ;clrf idec,0
        ;clrf icen,0
        ;clrf imil,0 ;bit monitor de interrupción
        incf imil,F,0
        retfie
        ;*****  

        org 0x300 ;DB directiva que Define Byte
        DB 0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x98
END

```

CONCLUSIONES

La presente práctica consistió en controlar cuatro exhibidores, ya que el contador es de 4 dígitos, por lo que necesitamos cuatro bits de puerto configurados como salida, ya que serán los encargados de encender y apagar los transistores que manejan los displays. En este programa, no se usamos la instrucción TBLRD *+, ya que tenemos un par de registros en donde se guardan los índices necesarios para el conteo el ‘Iun’ índice de unidades y ‘Idec’ índice de decenas. El programa mantiene los índices dentro del valor adecuado, iun inicia en cero, y se incrementa exactamente cada 500 ms, hasta llegar a 9, una vez que iun tiene el valor de 9, en los siguientes 500 ms, iun se regresa a cero, pero a la vez se incrementa idec. Debido a que para encender los dígitos necesitamos los códigos de 7 segmentos, tenemos también dos registros que los almacenan que son ‘Cuni’ código de 7 segmentos de unidades ‘Cdec’ código de 7 segmentos de decenas. De este modo se logra un conteo de fluido ascendente del 0 al 9999.

PRÁCTICA 7 RELOJ 24 HORAS, 4 DISPLAY

OBJETIVO

Aplicar los conceptos en programación de los puertos de entrada y salida digital (puertos C y D) del microcontrolador PIC18F4550, así como la utilización del Timer0 en modo de temporizador. Se realizará una aplicación en función a la práctica anterior, esta vez se modificará el límite superior que tendrá el contador para realizar una aplicación de reloj, teniendo este que resetearse en el valor 2359.

MARCO TEÓRICO

Esta práctica es una aplicación real de la anterior, el conocimiento necesario para ejecutarla es el mismo.

Temporizador 1

Características fundamentales:

- Configurable como temporizador/contador de 16 bits
- Dispone de un oscilador propio que puede funcionar como:
 - Señal de reloj del temporizador 1.
 - Señal de reloj del μ C en modos de bajo consumo.
 - Pre-escalar de 3 bits programable.
- Interrupción por desbordamiento

Existen dos modos de acceder al registro TMR1:

- RD16 (T1CON)=‘0’: se accede a TMR1L y TMR1H como dos registros independientes.
- RD16 (T1CON)=‘1’: la parte alta de TMR1 no es accesible directamente. Se accede a ella a través de TMR1H que funciona como un búfer:
 - Cuando se lee el valor de TMR1L, el valor de la parte alta de TMR1 pasa al registro TMR1H. Para leer el valor de TMR1 primero leemos TMR1L y luego leemos TMR1H.
 - Cuando se escribe en TMR1L, el valor de TMR1H pasa a la parte alta de TMR1. Para escribir en TMR1 primero escribimos la parte alta en TMR1H y luego escribimos la parte baja en TMR1L.

Interrupción del Temporizador 1:

- Flag de interrupción: bit TMR1IF (PIR1)
- Bit de habilitación: bit TMR1IE (PIE1)
- Prioridad: bit TMR1IP (IPR1): ‘0’->pr. baja/‘1’->pr. Alta

Si se produce el desbordamiento del Temporizador 1 se pone a ‘1’ el flag TMR1IF. Si el bit de habilitación TMR1IE está a ‘1’ y las interrupciones de periféricos están habilitadas a nivel

global se genera una interrupción y el µC pasa a ejecutar el código situado a partir de la posición 0008H o 0018H (según el nivel de prioridad establecido).

DESARROLLO

Se procedió a realizar las siguientes modificaciones en el código de la practica 6

```
xorlw 0x02
btfs STATUS,Z,0
bra lee
call veint
clr f icen,0
clr f imil,0

veint
leei    movff iun,TBLPTRL ;ajusta apuntador
tblrd * ;lee la tabla sin modificar apuntador
movff TABLAT,cuni ;cuni tiene código 7 segmentos
movff idec,TBLPTRL ;ajusta apuntador
tblrd * ;lee la tabla sin modificar apuntador
movff TABLAT,cdec ;cdec tiene código 7 segmentos
movff iumin, TBLPTRL
tblrd *
movff TABLAT,cumin
movff idmin, TBLPTRL
tblrd *
movff TABLAT,cdmin
movff icen,TBLPTRL
tblrd *
movff TABLAT,ccen
movff imil,TBLPTRL
tblrd *
movff TABLAT,cmil
```

Se hizo una nueva subrutina controla el caso del reinicio del conteo cuando este llega a las 23 horas, cuando el display de las centenas llega al valor de 2 se llama a la subrutina que tiene como nuevo límite en el dígito de las decenas a el valor de 3 y se procede a reiniciar el reloj

Para la implementación física de la práctica se usaron los siguientes materiales

Presentamos a continuación los materiales a utilizar en la presente práctica:

- Display de 4 Dígitos y 7 Segmentos
- Display de 2 Dígitos y 7 Segmentos
- Resistencias de 330 Ohms
- BC547
- Cables
- Protoboard
- Push Button
- PIC18F4550 Uva
- Ordenador para ejecutar nuestro programa

Simulación y Ensamblaje del Circuito

A continuación, mostramos el Diagrama el Circuito realizado en Proteus, así como el ensamblaje del mismo en Protoboard:

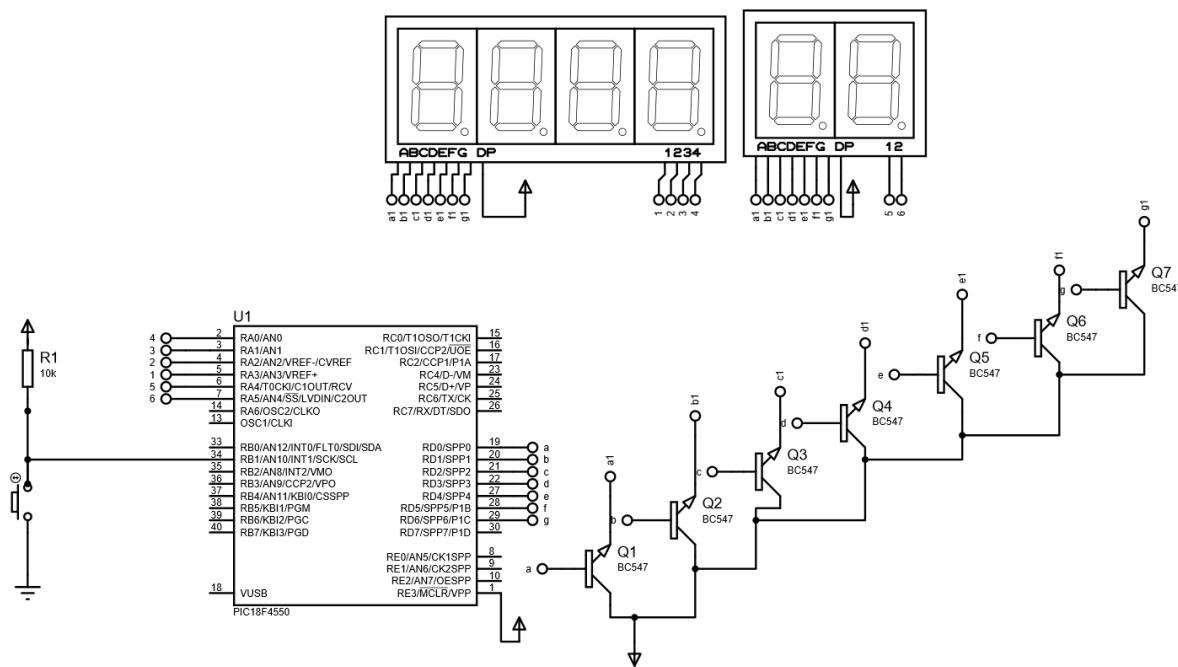


Figura 1: Esquemático Usado para la simulación del Circuito

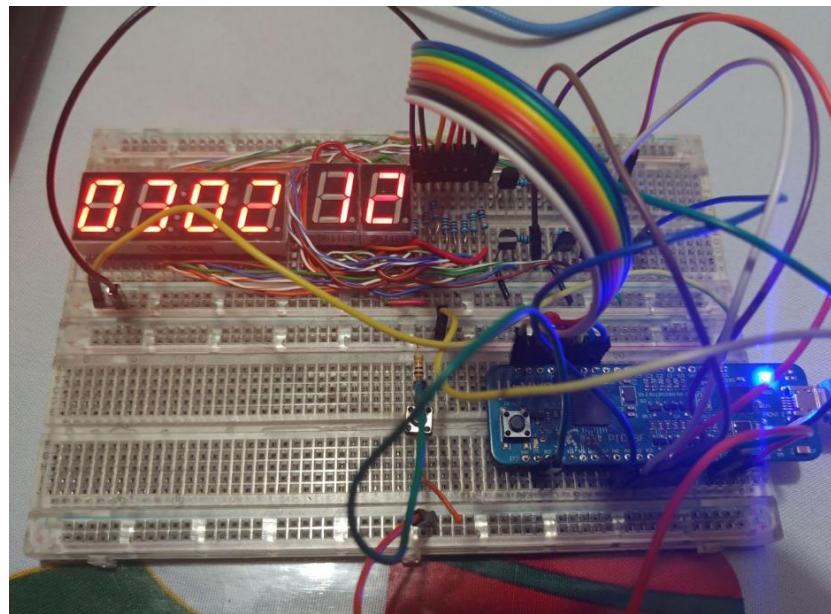


Figura 2: Implementación Física del circuito

Código Fuente

```
;*****  
***  
  
LIST P=18F4550           ;directiva para definir el procesador  
#include <P18F4550.INC> ;definiciones de variables específicas del procesa-  
dor  
  
;*****  
*** ;Bits de configuración  
CONFIG FOSC = INTOSC_XT    ;Oscilador INT usado por el uC , XT usado por el  
USB  
CONFIG BOR = OFF           ;BROWNOUT RESET DESHABILITADO  
CONFIG PWRT = ON            ;PWR UP Timer habilitado  
CONFIG WDT = OFF            ;Temporizador vigía apagado  
CONFIG MCLRE=ON             ;Reset apagado  
CONFIG PBADEN=OFF  
CONFIG LVP = OFF  
;*****  
***
```

```

;Definiciones de variables
CBLOCK 0x000 ;ejemplo de definición de variables en RAM de acceso
flags ;banderas
iun ;índice de unidades
cuni ;código de 7 segmentos de unidades
idec ;índice de decenas
cdec ;código de 7 segmentos de decenas
iumin ;indice de las unidades de minuto
cumin ;codigo de las unidades minutos
idmin
cdmin
icen
ccen
imil
cmil
cont
aux
ENDC ;fin del bloque de constantes
;*****
***

;Reset vector
    ORG 0x0000
    bra inicio
    org 0x08 ;vector de alta prioridad
    bra RST0 ;ramifica servicio interrupción T0
    org 0x18 ;vector de baja prioridad
    goto RSINT
    org 0x0020

inicio bsf OSCCON,IRCF2,0
    bsf OSCCON,IRCF1,0
    bcf OSCCON,IRCF0,0 ;Oscilador interno a 4 MHz

    movlw 0x0F
    movwf ADCON1,0           ;Puertos Digitales
    clrf PORTD,0
    clrf TRISD,0 ;Puerto D Configurado como salida
    ;setf TRISB,0
    ;movlw 0xE0
    ;movwf TRISC,0 ;RC0 y RC1 como salidas
    clrf TRISC
    clrf PORTC,0
    clrf TRISA
    bcf TRISB,0
    movlw 0x95
    movwf T0CON,0 ;timer 16 bits prescalerX64

```

```

movlw 0XE0
movwf INTCON,0 ;interrupciones TMR0,prioridad habilitada
bsf RCON,IPEN,0 ;habilitamos prioridades de interrupción
movlw 0xC2
movwf TMR0H,0
movlw 0xF7
movwf TMR0L,0 ;valor de precarga para 1000ms a 4MHz
movlw 0x88
movwf INTCON3,0 ;habilitamos int1 en baja prioridad
clrf TBLPTRL,0
movlw 0x03
movwf TBLPTRH,0
clrf TBLPTRU,0 ;tblptr=0x000300
clrf iun,0
clrf idec,0 ;iniciamos en 0
clrf icen,0
clrf imil,0
clrf iumin,0
clrf idmin,0
clrf aux,0

lee    movff iun,TBLPTRL ;ajusta apuntador
tblrd * ;lee la tabla sin modificar apuntador
movff TABLAT,cuni ;cuni tiene código 7 segmentos
movff idec,TBLPTRL ;ajusta apuntador
tblrd * ;lee la tabla sin modificar apuntador
movff TABLAT,cdec ;cdec tiene código 7 segmentos
movff iumin, TBLPTRL
tblrd *
movff TABLAT,cumin
movff idmin, TBLPTRL
tblrd *
movff TABLAT,cdmin
movff icen,TBLPTRL
tblrd *
movff TABLAT,ccen
movff imil,TBLPTRL
tblrd *
movff TABLAT,cmil

loop   movlw 0x01
      movwf PORTA,0 ;encendemos display unidades
      movff cuni,PORTD
;call INCREMENTO
      call retardo
      movlw 0x02

```

```

movwf PORTA,0 ;encendemos display decenas
movff cdec,PORTD
call retardo
movlw 0x04
movwf PORTA,0 ;encendemos display unidades
movff cumin,PORTD
call retardo
movlw 0x08
movwf PORTA,0 ;encendemos display unidades
movff cdmin,PORTD
call retardo
movlw 0x10
movwf PORTA,0;encendemos display de las centenas
movff ccen,PORTD
call retardo
movlw 0x20
movwf PORTA,0;encendemos display de las centenas
movff cmil,PORTD
call retardo
btfs flags,0,0
bra loop
bcf flags,0,0
incf iun,F,0
movf iun,W,0
xorlw 0x0a
btfs STATUS,Z,0;verifica límite de tabla
bra lee
clrf iun,0
incf idec,F,0
movf idec,W,0
xorlw 0x06
btfs STATUS,Z,0
bra lee
clrf iun,0
clrf idec,0
incf iumin,F,0
movf iumin,W,0
xorlw 0x0a
btfs STATUS,Z,0
bra lee
clrf iun,0
clrf idec,0
clrf iumin,0
incf idmin,F,0
movf idmin,W,0

```

```

xorlw 0x06
btfs STATUS,Z,0
bra lee
clrf iun,0
clrf idec,0
clrf iumin,0
clrf idmin,0
incf icen,F,0
movf icen,W,0
movwf aux,0
xorlw 0x0a
btfs STATUS,Z,0
bra lee
clrf iun,0
clrf idec,0
clrf icen,0
incf imil,F,0
movf imil,W,0
xorlw 0x02
btfs STATUS,Z,0
bra lee
call veint
clrf icen,0
clrf imil,0
goto lee
*****;
*****
veint
leei    movff iun,TBLPTRL ;ajusta apuntador
tblrd * ;lee la tabla sin modificar apuntador
movff TABLAT,cuni ;cuni tiene código 7 segmentos
movff idec,TBLPTRL ;ajusta apuntador
tblrd * ;lee la tabla sin modificar apuntador
movff TABLAT,cdec ;cdec tiene código 7 segmentos
movff iumin, TBLPTRL
tblrd *
movff TABLAT,cumin
movff idmin, TBLPTRL
tblrd *
movff TABLAT,cadmin
movff icen,TBLPTRL
tblrd *
movff TABLAT,ccen
movff imil,TBLPTRL
tblrd *

```

```

        movff TABLAT,cmil
loopi    movlw 0x01
        movwf PORTA,0 ;encendemos display unidades
        movff cuni,PORTD
        call retardo
        movlw 0x02
        movwf PORTA,0 ;encendemos display decenas
        movff cdec,PORTD
        call retardo
        movlw 0x04
        movwf PORTA,0 ;encendemos display unidades
        movff cumin,PORTD
        call retardo
        movlw 0x08
        movwf PORTA,0 ;encendemos display unidades
        movff cdmin,PORTD
        call retardo
        movlw 0x10
        movwf PORTA,0;encendemos display de las centenas
        movff ccen,PORTD
        call retardo
        movlw 0x20
        movwf PORTA,0;encendemos display de las centenas
        movff cmil,PORTD
        call retardo
        btfss flags,0,0
        bra loopi
        bcf flags,0,0
        incf iun,F,0
        movf iun,W,0
        xorlw 0x0a
        btfss STATUS,Z,0;verifica límite de tabla
        bra leei
        clrf iun,0
        incf idec,F,0
        movf idec,W,0
        xorlw 0x06
        btfss STATUS,Z,0
        bra     leei
        clrf iun,0
        clrf idec,0
        incf iumin,F,0
        movf iumin,W,0
        xorlw 0x0a
        btfss STATUS,Z,0

```

```

bra leei
clrf iun,0
clrf idec,0
clrf iumin,0
incf idmin,F,0
movf idmin,W,0
xorlw 0x06
btfs STATUS,Z,0
bra leei
clrf iun,0
clrf idec,0
clrf iumin,0
clrf idmin,0
incf icen,F,0
movf icen,W,0
movwf aux,0
xorlw 0x04
btfs STATUS,Z,0
bra leei
return

;*****
***  

retardo setf cont,0
nada    nop
    decfsz cont,F,0
    bra nada
    return
;*****  

*****  

RST0    bcf INTCON,TMR0IF,0 ;apagamos bandera timer0
        movlw 0xC2
        ;movwf 0xF1
        movwf TMR0H,0
        ;movlw 0x7C
        movlw 0xF7
        movwf TMR0L,0 ;valor de precarga para 1000ms a 4MHz
        bsf flags,0,0
        retfie
;*****  

RSINT   bcf INTCON3,INT1IF,0 ;Limpiamos bandera de interrupción
        incf iumin,F,0
        movf iumin,W,0
        xorlw 0xa
        btfs STATUS,Z,0

```

```

bra nada1
clrf iumin
incf idmin,F,W
movf idmin,W,0
xorlw 0x06
btfs STATUS,Z,0
bra nada1
clrf iumin
clrf idmin
incf icen,F,0
movf icen,W,0
movwf aux,0
xorlw 0x0a
btfs STATUS,Z,0
bra nada1
clrf icen,0
incf imil,F,0
movf imil,W,0
xorlw 0x02
btfs STATUS,Z,0
bra nada1
clrf iumin
clrf idmin
clrf icen,0
clrf imil,0
nada1    nop
retfie
INCREMENTO
btfs PORTB,0
nop
incf iumin,F,0
return
;*****
org 0x300      ;DB directiva que Define Byte
DB 0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x98
END

```

CONCLUSIONES

Se vio que el uso de subrutinas puede ser muy conveniente a la hora de implementar estructuras condicionales, también se pudo ver que la exactitud del timer a la hora de implementar aplicaciones que usen el tiempo como variable de importancia, se vio también una aplicación de las tablas dentro de subrutinas para mostrar el paso del tiempo con gran exactitud, cabe resaltar también la importancia del control sobre el flujo de trabajo a la hora de ejecutar el código, ya que este permite que se logren los resultados deseados

PRÁCTICA 8 GENERACIÓN DE SEÑAL PWM

OBJETIVO

Configurar el módulo CCP1 en modo PWM, para variar la velocidad de un motor de corriente directa usando la interrupción externa para modificar el ciclo de trabajo de PWM.

MARCO TEÓRICO

Modulo CCP:

Puede ser configurado de tres modos de operación posibles dependiendo del valor del registro de control CCP1CON para el primer módulo o CCP2CON para el segundo módulo.

- Captura
- Comparación
- PWM

Este módulo tiene dos registros asociados estos son CCPR1H y CCPR1L para el CP1 y en el caso del CP2 son CCPR1H y CCPR1L, en el modo de PWM el registro CCPR1L funciona como registro maestro y el registro CCPR1H funciona como esclavo, además de esto será necesario usar el Timer 2 ya que este es de 8 bits. Los modos de captura y comparación trabajan con el Timer 1 o Timer 3 y se pueden usar para interactuar con un encoder o para la generación de diversas frecuencias. Para el desarrollo de esta práctica se usará el módulo CP1 que tiene asociado el pin RC1, si se desea usar el módulo CCP2 el pin a utilizar sería el RC2.

Timer 2

Este timer tiene la particularidad de poseer un postdivisor y no desbordarse, se usará principalmente como temporizador, al ser de 8 bits puede almacenar valores de 0 a 255, tiene asociado al registro PR2 también de 8 bits, este tiene la función de almacenar el límite superior de TMR2 una vez los valores en los registros TMR2 y PR2 sean iguales el valor de TMR2 se reseteará a cero volviendo a iniciar el ciclo de incremento para el registro TMR2.

Ciclo de Trabajo (Duty Cycle)

Este ciclo se especifica usando 10 bits, van repartidos en los registros CCPR1H y CCPR1L de la siguiente manera, los 8 bits más significativos van escritos en el registro CCPR1L y los 2 bits menos significativos van en los bits 5 y 4 del registro CCPR1, su funcionamiento se basa en comparar el valor de estos 10 bits con el registro TMR2, cada vez que el conteo en el timer 2 se reinicie se reiniciara también el periodo del PWM. El periodo del PWM y el ciclo de trabajo se calcularán usando las siguientes expresiones.

$$PWM = [PR2 + 1] * 4 * Tosc * prescalerTMR2$$

$$PWM\ Duty\ Cicle = (CCPRxL:CCPxCON < 5:4 >) * Tosc * (prescalerTMR2)$$

DESARROLLO

La práctica se desarrolló en base al ejemplo de los apuntes del curso, adicionalmente como se especificó la interrupción INT2 al ser activada tendrá que modificar el ciclo de trabajo de la señal PWM en 10%, con este fin se calcularon los valores correspondientes a cada porcentaje estos se guardaron en una tabla con el fin de hacer el código más conciso.

```
ciclo ;TABLA QUE GUARDA LOS VALORES PARA EL CICLO DE TRABAJO
    rlcf contador,W,0 ;multiplica índice por 2
    addwf PCL,1 ;ajusta el PCL de acuerdo al valor del índice
    retlw 0xC0 ;código del caso no existente
    retlw 0x13 ;código del 10% del pulso ←
    retlw 0x27 ;código del 20% del pulso
    retlw 0x3A ;código del 30% del pulso
    retlw 0x4E ;código del 40% del pulso
    retlw 0x61 ;código del 50% del pulso
    retlw 0x75 ;código del 60% del pulso
    retlw 0x89 ;código del 70% del pulso
    retlw 0x9C ;código del 80% del pulso
    retlw 0xB0 ;código del 90% del pulso
    return
end
```

El cálculo de estos valores se realizó de la siguiente manera

Caso 10%

$$Fosc = 125\text{KHz}, Tosc = 8\mu\text{s}$$

$$(CCPRxL:CCPxCO / N < 5:4 >) = \frac{PWMDutyCycle}{Tosc * (\text{prescalerTMR2})}$$

$$(CCPRxL:CCPxCO / N < 5:4 >) = \frac{10 \cdot 10^{-3}}{8 \cdot 10^{-6} \cdot 16} = 78.125 \approx 78$$

78 en binario es

$$78 = 0001001110$$

Los 8 bits más significativos son

$$00010011 = 13$$

El resto de los porcentajes del ciclo de trabajo se realizó de la misma forma en la que se desarrollo para el caso de 10%.

Presentamos a continuación los materiales a utilizados en la práctica:

- Display de 4 Dígitos y 7 Segmentos
 - Resistencias de 330 Ohms
 - BC547
 - Cables
 - Protoboard
 - Motor DC
 - Fuente de Tension
 - Push Button
 - PIC18F4550 Uva
 - Ordenador para ejecutar nuestro programa

Implementación del Circuito

Se muestra el esquemático del circuito usado para la práctica :

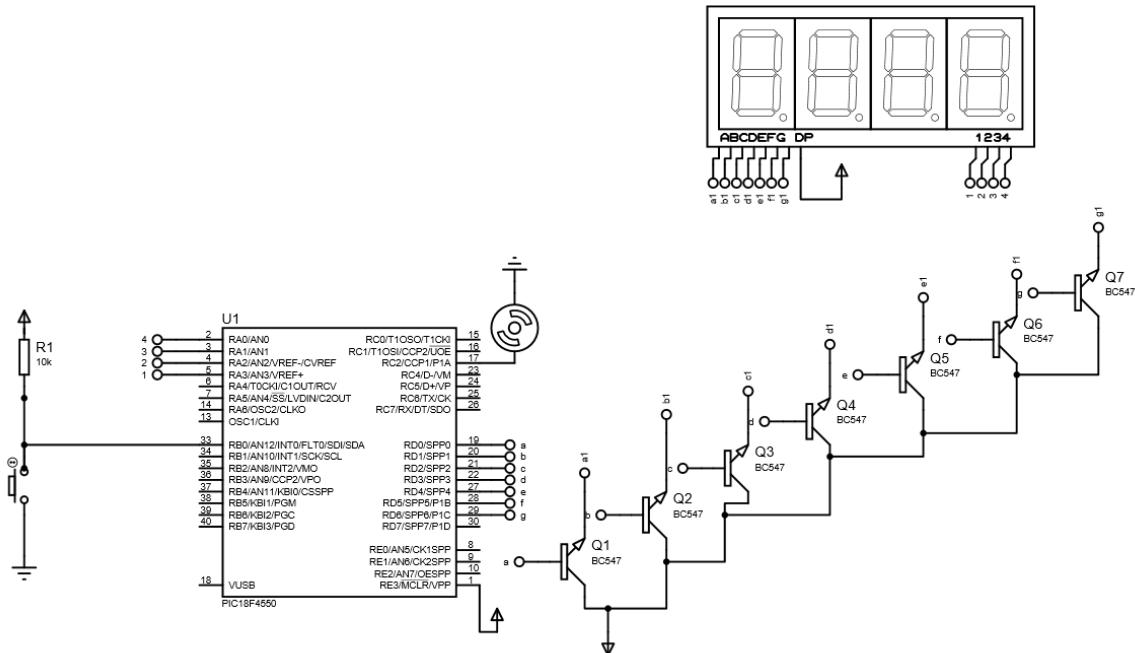


Figura 1: Esquemático Usado para la simulación del Circuito

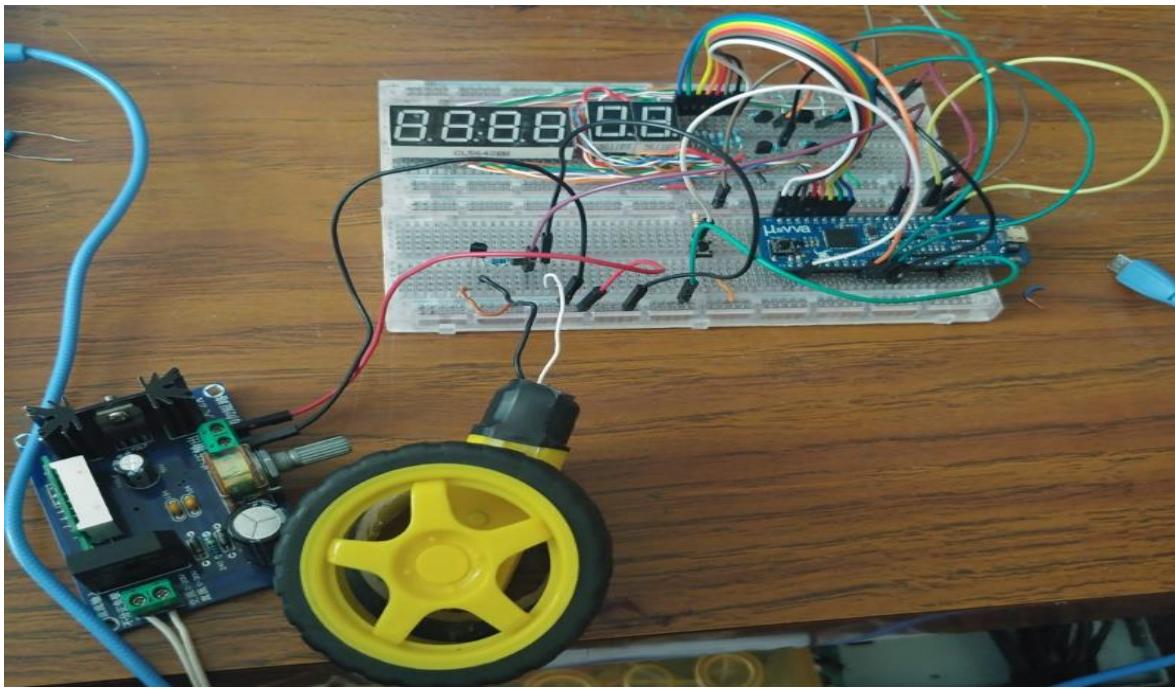


Figura 2: Implementación Física del circuito

Código Fuente

```
;*****
***  
LIST P=18F4550 ;directiva para definir el procesador  
#include <P18F4550.INC> ;definiciones de variables específicas del procesador  
r  
;*****  
***  
;Bits de configuración  
CONFIG FOSC = INTOSC_XT ;Oscilador INT usado por el uC , XT usado por el USB  
CONFIG BOR = OFF ;BROWNOUT RESET DESHABILITADO  
CONFIG PWRT = ON ;PWR UP Timer habilitado  
CONFIG WDT = OFF ;Temporizador vigia apagado  
CONFIG MCLRE= ON ;Reset apagado  
CONFIG PBADEN=OFF  
  
;*****  
***  
;Definiciones de variables  
CBLOCK 0x000 ;ejemplo de definición de variables en RAM de acceso  
contador  
cont  
ENDC ;fin del bloque de constantes
```

```

;*****  

***  

ORG 0x0000 ;vector de reset  

bra inicio  

inicio  

    bcf OSCCON,IRCF2,0  

    bsf OSCCON,IRCF0,0 ;Oscilador interno a 125 kHz  

    movlw 0x0F  

    movwf ADCON1,0 ;Puertos Digitales  

    clrf PORTA,0  

    clrf TRISA,0  

    clrf PORTD,0  

    clrf TRISD,0 ;Puerto D Configurado como salida  

    ;configurar PWM  

    movlw 0x1C  

    movwf CCP1CON  

    movlw 0x07  

    movwf T2CON,0 ;PRESCALER TIMER 2 X16  

    movlw d'194'  

    movwf PR2,0 ;CONFIGURACIÓN DE PERIODO pwm 100ms  

    bcf TRISC,2,0; Pin ccp1 como salida  

    movlw 0x01 ;declarar puerto B PIN 0 COMO ENTRADA  

    movwf TRISB  

    clrf PORTB  

    movlw 0x00  

    movwf contador  

    movf contador,0  

    call decodificador  

    movwf PORTD  

    movlw 0x13  

    movwf CCPR1L,0  

loop  

    btfss PORTB,0  

    bra sig  

    incf contador,1  

    movf contador,W,0  

    xorlw 0x0A  

    btfss STATUS,Z,0  

    bra nada1  

    movlw 0x01  

    movwf contador  

nada1    nop  

antirrebote  

    call ciclo

```

```

movwf CCPR1L,0 ;CONFIGURAR EL CICLO DE TRABAJO
call decodificador; MOSTRAR EL CICLO DE TRABAJO EN EL DISPLAY
movwf PORTD
movlw 0x02
movwf PORTA
call retardo
movlw 0xC0
movwf PORTD
movlw 0x01
movwf PORTA
call retardo
movlw 0xC6
movwf PORTD
movlw 0x04
movwf PORTA,0
call retardo
movlw 0x08
movwf PORTA,0
movlw 0xA1
movwf PORTD
call retardo
btfs PORTB,0 ;RUTINA PARA SELECCIONAR CORRECTAMENTE EL CICLO
goto antirrebote
sig movf contador,0
    call decodificador
    goto loop
;*****
retardo movlw 0x25 ;RETARDO PARA MOSTRAR DATOS EN EL DISPLAY
    movwf cont
nada    nop
    decfsz cont,F,0
    bra nada
    return
decodificador ;TABLA QUE GUARDA LOS VALORES DEL DISPLAY
    rlcf contador,W,0 ;multiplica índice por 2
    addwf PCL,1 ;ajusta el PCL de acuerdo al valor del índice
    retlw 0xC0 ;código del cero
    retlw 0xF9 ;código del uno
    retlw 0xA4 ;código del dos
    retlw 0xB0 ;código del tres
    retlw 0x99 ;código del cuatro
    retlw 0x92 ;código del cinco
    retlw 0x82 ;código del seis
    retlw 0xF8 ;código del siete
    retlw 0x80 ;código del ocho

```

```

retlw 0x98 ;código del nueve
return
ciclo ;TABLA QUE GUARDA LOS VALORES PARA EL CICLO DE TRABAJO
rlcf contador,W,0 ;multiplica índice por 2
addwf PCL,1 ;ajusta el PCL de acuerdo al valor del índice
retlw 0xC0 ;código del caso no existente
retlw 0x13 ;código del 10% del pulso
retlw 0x27 ;código del 20% del pulso
retlw 0x3A ;código del 30% del pulso
retlw 0x4E ;código del 40% del pulso
retlw 0x61 ;código del 50% del pulso
retlw 0x75 ;código del 60% del pulso
retlw 0x89 ;código del 70% del pulso
retlw 0x9C ;código del 80% del pulso
retlw 0xB0 ;código del 90% del pulso
return
end

```

CONCLUSIONES

Se observó el uso del módulo CPP configurado en modo PWM, en este uso pudimos comprobar la exactitud de las expresiones matemáticas dadas en la hoja de datos del microcontrolador, también vimos el uso del display de 7 segmentos para la visualización de los ciclos de trabajo y la variación de velocidad del motor de corriente directa usado en la práctica, dicho esto se concluye que la generación de PWM usando el PIC18F4550 es bastante fiable para diversas aplicaciones.

PRÁCTICA 9 ADC

OBJETIVO

Activar el módulo ADC del microcontrolador para leer la entrada analógica de un sensor de temperatura LM35 y mostrar el resultado en los displays de 7 segmentos.

MARCO TEÓRICO

Modulo ADC:

Este módulo permite la conversión de una señal analógica en un número digital de 10 bits, estos se almacenan en los registros ADRESH y ADRESL, la configuración se realiza mediante los registros de control ADCON0, ADCON1, ADCON2, según la configuración de estos registros el resultado puede ir con los 8 MSB en el registro ADRESH o en ADRESL, en estos registros de control podemos establecer cuantas entradas analógicas usaremos y donde estarán las referencias de voltaje.

El voltaje de referencia a usar también puede ser configurado, según se desee, se pueden usar los pines de la fuente o el voltaje de la fuente, RA3/AN3/VREF+ y RA2/AN2/VREF

El resultado es generado mediante la técnica de aproximaciones sucesivas, una vez esta conversión se completa, el resultado se carga en los registros ADRESH: ADRESL, el bit GO/DONE se limpia y la bandera de interrupción ADIF se activa.

GO/DONE

Este bit es una bandera nos indica el estado de la conversión, en el inicio de la conversión este bit tiene el valor de 1 cuando la conversión termina su valor se pone en cero, esto es útil ya que se puede usar este bit para configurar la interrupción ADIF.

Tiempo de Adquisición

Este es el tiempo mínimo antes de iniciar una nueva conversión, este valor se selecciona usando el registro ADCON2 cada vez que el bit GO/DONE es puesto, así como también, se tiene la opción de usar un tiempo de adquisición predeterminado, estos tiempos predeterminados se encuentran en un rango de 2 a 20 TAD (tiempo de conversión por bit), se recomienda que el reloj de conversión A/D(TAD) debe ser tan corto como sea posible pero mayor que el tiempo mínimo TAD.

Uso del Disparo Del CCP2

El módulo CCP2 se puede usar para generar un evento especial de disparo para iniciar la conversión A/D, cuando el disparo ocurre se pondrá el bit GO/DONE y el timer 1 o timer3 se reiniciarán para repetir el proceso de adquisición.

DESARROLLO

En esta práctica se usó el sensor LM35 para la adquisición de datos analógicos, luego se procedió a configurar el ADC del microcontrolador como se muestra.

```
movlw 0x0E
movwf ADCON1 ;voltajes de referencia de alimentación y configuración AN0
movlw 0x00
movwf ADCON0 ;configuración del canal AN0
movlw 0x08
movwf ADCON2 ;configuración adc
```

Al cargar 0x0E en el registro ADCON1 estamos indicando que los voltajes de referencia del ADC serán los de alimentación del microcontrolador.

Al cargar 0x00 en el registro ADCON0 estamos seleccionando el pin AN0 para como entrada de la señal analógica a convertir

Al cargar 0x08 en el registro ADCON2 estamos indicando que el resultado de la conversión se almacenará justificado a la izquierda en los registros ADRESH y ADRESL, además de esto, se establece también que el tiempo de adquisición es el 0 TAD y el reloj de conversión del ADC es derivado del oscilador RC de este mismo.

BUCLE

```
bsf ADCON0, GO_DONE ;inicio de la conversión
btfsr ADCON0, GO_DONE ;terminó la conversión?
goto BUCLE ;NO, regresa al bucle
clrf centenas ;si limpia registros de datos
clrf decenas
clrf unidades
movf ADRESH,W ;mueve el registro ADRESH a W
addwf ADRESH,W ;aproxima el valor de ADRESH al valor en grados duplicando su
valor
movwf result ;guarda el valor duplicado en result
```

El proceso de adquisición de datos del ADC se realiza de la siguiente manera, primero se revisa el estado del bit GO/DONE recordemos que este bit indica si la conversión se terminó o no. Si esta adquisición ya se terminó, se limpia los registros de almacenamiento de datos y se almacena en el registro “result” para ser mostrado usando los displays de 7 segmentos

Presentamos a continuación los materiales a utilizados en la práctica:

- Display de 4 Dígitos y 7 Segmentos
- Resistencias de 330 Ohms
- BC547

- Cables
- Protoboard
- LM35
- Fuente de Tension
- PIC18F4550 Uva
- Ordenador para ejecutar nuestro programa

Implementación del Circuito

Se muestra el esquemático del circuito usado para la práctica

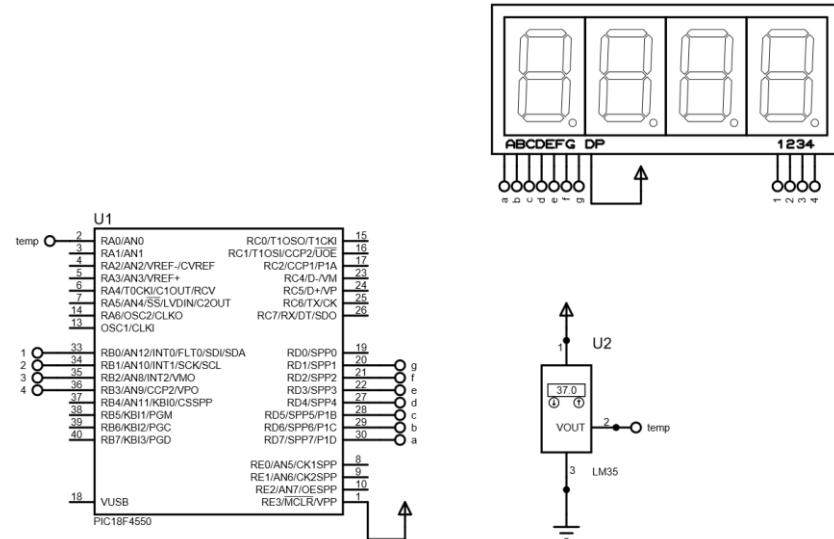


Figura 1: Esquemático Usado para la simulación del Circuito

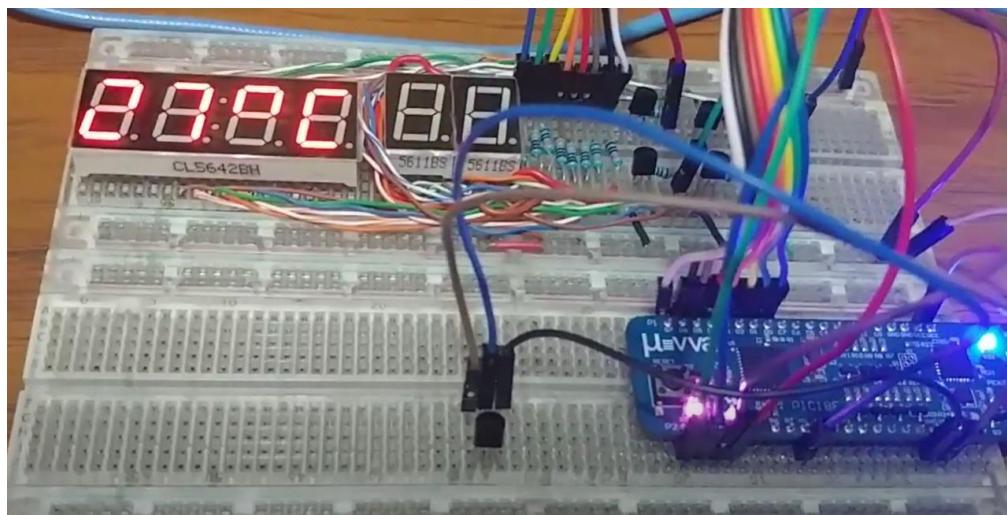


Figura 2: Implementación Física

Código Fuente

```
LIST P=18F4550           ;directiva para definir el procesador
#include <P18F4550.INC> ;definiciones de variables específicas del procesador
;
;***** ;Bits de configuración
CONFIG FOSC = INTOSC_XT ;Oscilador INT usado por el uC , XT usado por el US
B
CONFIG BOR  = OFF        ;BROWNOUT RESET DESHABILITADO
CONFIG PWRT = ON         ;PWR UP Timer habilitado
CONFIG WDT  = OFF        ;Temporizador vigia apagado
CONFIG MCLRE= ON         ;Reset APAGADO
CONFIG PBADEN=OFF
CONFIG LVP  = OFF
;*****CBLOCK
;Código
CBLOCK 0x000
    result ;variable para realizar operaciones aritméticas
    centenas;valor de las centenas en grados celsius
    decenas ;valor de las decenas en grados celsius
    unidades ;valor de las unidades en grados celsius
    cuni ;codigo unidades de grados celsius
    cdec ;codigo decenas de grados celsius
    cont ;variables para el retardo
ENDC
ORG 0x00      ;Iniciar el programa en el registro 0x00
goto INICIO
INICIO
bsf OSCCON,IRCF2,0
bsf OSCCON,IRCF1,0
bcf OSCCON,IRCF0,1 ;Oscilador interno a 500 kHz
clrf TBLPTRL,0
movlw 0x03
movwf TBLPTRH,0
clrf TBLPTRU,0 ;tblptr=0x000300
movlw 0x00
movwf TRISB ;declaración de salidas
movwf TRISD
movlw 0x0E
movwf ADCON1 ;voltajes de referencia de alimentación y configuración AN
0
movlw 0x00
movwf ADCON0 ;configuración del canal AN0
movlw 0x08
movwf ADCON2 ;configuración adc
```

```

lee
    movff unidades,TBLPTRL ;ajusta apuntador
   tblrd * ;lee la tabla sin modificar apuntador
    movff TABLAT,cuni ;cuni tiene código 7 segmentos
    movff decenas,TBLPTRL ;ajusta apuntador
   tblrd * ;lee la tabla sin modificar apuntador
    movff TABLAT,cdec ;cdec tiene código 7 segmentos
    bsf ADCON0,ADON ;ENCENDIDO DEL ADC

BUCLE
    bsf ADCON0, GO_DONE ;inicio de la conversión
    btfsc ADCON0, GO_DONE ;terminó la conversión?
    goto BUCLE ;NO, regresa al bucle
    clrf centenas ;si limpia registros de datos
    clrf decenas
    clrf unidades
    movf ADRESH,W ;muestra el registro ADRESH a W
    addwf ADRESH,W ;aproxima el valor de ADRESH al valor en grados duplicando su valor
    movwf result ;guarda el valor duplicado en result
    ;realizar divisiones por el método de las restas sucesivas

centenas1
    movlw d'100' ;muestra a w el número 100
    subwf result,W ;resta de resultado 100
    btfss STATUS,C ;resultado menor que 100?
    bra decenas1 ;si, vete a decenas
    movwf result ;no, salva resultado
    incf centenas,1; incremento las decenas
    bra centenas1 ;vuelve a realizar la resta

decenas1
    movlw d'10' ;muestra a w el número 10
    subwf result,W ;resta de resultado 10
    btfss STATUS,C; resultado menor que 10?
    bra unidades1 ;si, vete a unidades
    movwf result ;no, entonces salva resultado
    incf decenas,1 ;incrementa las decenas
    bra decenas1 ;regresa a las decenas

unidades1
    movf result,W ;muestra resultado a w
    movwf unidades ;guarda resultado en unidades

    movlw 0x02 ;enciende 2do display
    movwf PORTB
    movff cuni,PORTD ;muestro unidades en el display
    call retardo
    movlw 0x01 ;enciende 1er display

```

```

movwf PORTB
movff cdec,PORTD ;muestro decenas en el display
call retardo
movlw 0x04 ;enciendo 3er display
movwf PORTB
movlw 0x9C ;muestro simbolo de grados
movwf PORTD
call retardo
movlw 0x08 ;enciendo 4to display
movwf PORTB
movlw 0xC6 ;muestro C de grados celsius
movwf PORTD
call retardo
bra lee
;*****
***  

;rutina de retardo
retardo setf cont,0
nada nop
decfsz cont,F,0
bra nada
return
;*****
;tabla de valores del display
org 0x300 ;DB directiva que Define Byte
DB 0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x98
END

```

CONCLUSIONES

El uso del módulo ADC en conjunto con los registros de configuración son herramientas muy versátiles a la hora de hacer aplicaciones de lectura de datos de un sensor al ser este de 10 bits de precisión las mediciones realizadas serán bastante fiables para una gran cantidad de aplicaciones, también se pudo ver que el display de 7 segmentos es muy versátil a la hora de mostrar los datos adquiridos a un usuario final .

PRÁCTICA 10 TECLADO MATRICIAL Y LCD

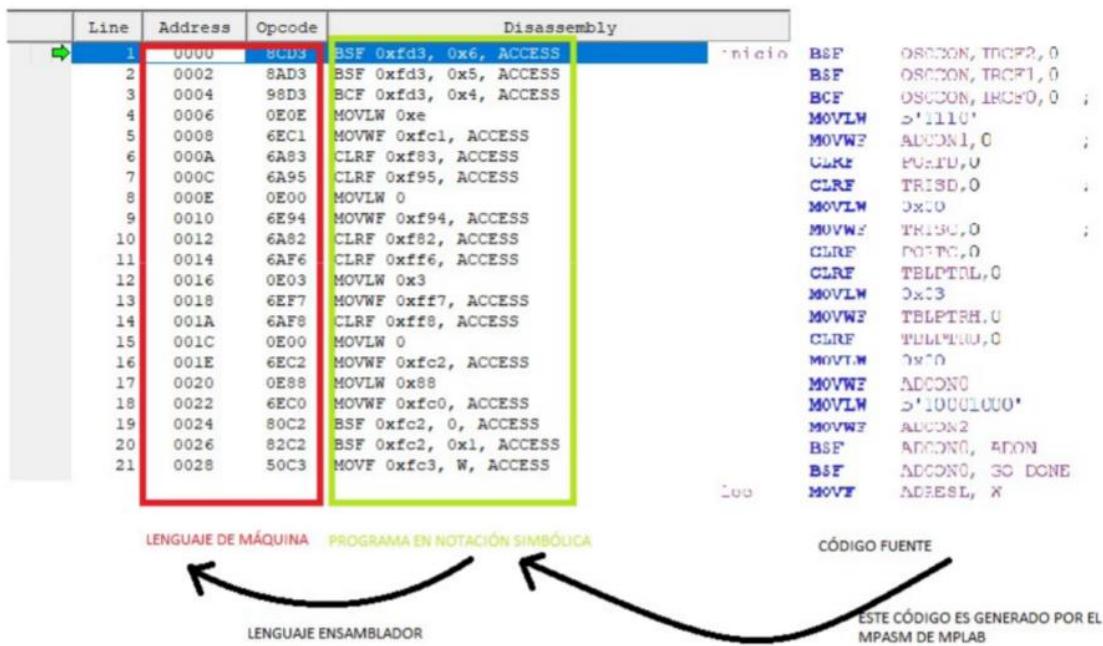
OBJETIVO

Conectar el microcontrolador a una pantalla LCD alfanumérica de 2 renglones y 16 caracteres, además un teclado matricial de 4 renglones x cuatro columnas (16 teclas) y usando el teclado, imprimir el valor asignado a la tecla en el segundo renglón del LCD mientras en el primer renglón se muestra un mensaje de bienvenida.

MARCO TEÓRICO

Ensamblador

Es un programa que toma como entrada código fuente escrito en ensamblador y lo convierte a código máquina, el cual es ejecutable directamente por el microcontrolador.



Compilador

Es un traductor, toma como entrada código fuente en un lenguaje y lo convierte a otro lenguaje.

Notas importantes

- Cada compilador funciona diferente de manera interna por lo cual, no se espera que el código fuente escrito para un compilador funcione de igual manera con otro.

- Como se depende de un compilador no se tiene control directo de los recursos del MCU, y se pueden dar casos en los cuales el uso de la memoria no es el más eficiente, por lo que se recomienda usar el lenguaje de alto nivel cuando la complejidad de la aplicación así lo requiera.

Librería **kbd_4x4.c**

Esta librería es una versión modificada de la proporcionada por PIC C compiler, esta modificación se realizó para ser usada con el teclado matricial 4x4 ya que la librería original está hecha para trabajar con un teclado matricial de 4x3 además de esto también se pueden modificar el carácter de entrada al presionar alguna tecla del teclado matricial.

DESARROLLO

Se uso el compilador PIC C compiler para la realización de esta práctica, primero se modificó la librería “kbd.c” como se muestra a continuación

```
#define COL3 (1 << 7) //Fila agregada, modificación de bits
case 3      : set_tris_kbd(ALL_PINS&~COL3);
                kbd=~COL3&ALL_PINS;
                break; //Aregar caso para columna 3
if(col==4)//Modificación para caso de columna 4
    col=0;
```

Después de modificar la librería se escribió una función para la lectura de datos del teclado matricial.

```
char tecla_time(void)
{
    char c='\0';
    unsigned int16 timeout;
    timeout=0;
    c=kbд_getc();
    while(c=='\0' && (++timeout <(key_delay*100)))
    {
        delay_us(10);
        c=kbд_getc();
    }
    return (c);
}
```

La lectura de datos se realiza mediante la función “kbд_getc()” esta pertenece a la librería “**kbd_4x4.c**” que nos permite interactuar con el teclado matricial.

```

WHILE(True)
{
    DIR=1;
    LCD_GOTOXY(1,2); //Ubica el cursor del LCD
    while (DIR<17)
    {
        if(k!='\0')
        {
            LCD_GOTOXY(DIR,2);
            lcd_putc(k);
            k='\0';
            DIR++;
        }
        k=tecla_time(); //Lee el valor del teclado pero solo espera un tiempo determinado
        if(DIR>16)
            LCD_PUTC("\f");
        lcd_gotoxy(1,1);
        lcd_putc("Bienvenido!!");
    }
}

```

También se tiene el ciclo infinito donde se ejecutan las funciones que mostrarán continuamente los datos tomados del teclado matricial en el LCD usando las funciones de la librería “**lcd.c**”

Presentamos a continuación los materiales a utilizados en la práctica:

- LCD 16x2
- Teclado Matricial
- Cables
- Protoboard
- PIC18F4550 Uva
- Ordenador para ejecutar nuestro programa

Implementación del Circuito

Se muestra el esquemático del circuito usado para la práctica

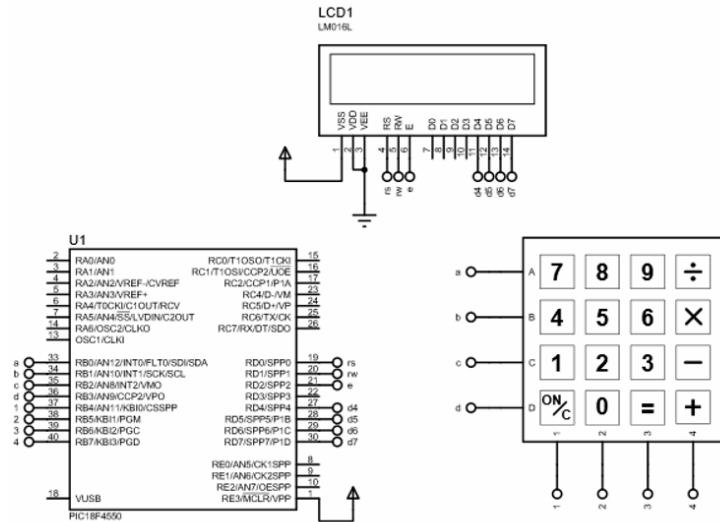


Figura 1: Esquemático Usado para la simulación del Circuito

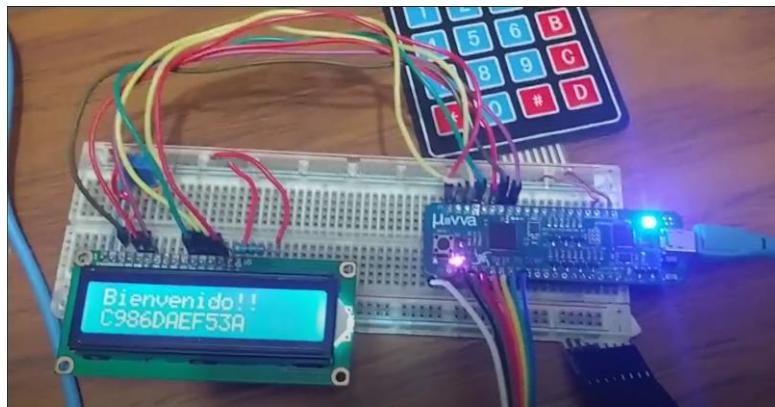


Figura 2: Implementación Física

Código Fuente

```
//Librerias del programa
#include <Práctica 10.h>
#include<lcd.c>
#include<kbd_4x4.c>
#define key_delay 200 //200 ms de espera para el teclado
int DIR;//variable para contar el numero de caracteres en el display

//Función tecla con timer
//Se encarga de esperar a que se presione una tecla
char tecla_time(void)
{
```

```

char c='\0';
unsigned int16 timeout;
timeout=0;
c=kbd_getc();
while(c=='\0' && (++timeout <(key_delay*100)))
{
    delay_us(10);
    c=kbd_getc();
}
return (c);
}

void main()
{ //La función lcd_init() usa PORTD para controlar el LCD alfanumérico
    char k;
    port_b_pullups(0xFF);
    set_tris_b(0);
    set_tris_c(0);
    lcd_init();
    lcd_putc("Bienvenido!!!");
    KBD_INIT();
    WHILE(True)
    {
        DIR=1;
        LCD_GOTOXY(1,2); //Ubica el cursor del LCD
        while (DIR<17)
        {
            if(k!='\0')
            {
                LCD_GOTOXY(DIR,2);
                lcd_putc(k);
                k='\0';
                DIR++;
            }
            k=tecla_time(); //Lee el valor del teclado pero solo espera un
tiempo determinado
            if(DIR>16)
                LCD_PUTC("\f");
            lcd_gotoxy(1,1);
            lcd_putc("Bienvenido!!!");
        }
    }
}

```

Librería Modificada

```
/////////////////////////////
```

```

////          KBD.C          /////
////          Generic keypad scan driver /////
////          /////
////  kbd_init()  Must be called before any other function. /////
////          /////
////  c = kbd_getc(c)  Will return a key value if pressed or /0 if not /////
////          This function should be called frequently so as /////
////          not to miss a key press. /////
////          /////
////////////////////////////// (C) Copyright 1996,2003 Custom Computer Services /////
////  This source code may only be used by licensed users of the CCS C /////
////  compiler.  This source code may only be distributed to other /////
////  licensed users of the CCS C compiler.  No other use, reproduction /////
////  or distribution is permitted without written permission. /////
////  Derivative programs created using this software in object code /////
////  form are not restricted in any way. /////
////////////////////////////// The following defines the keypad layout on port D

// Un-comment the following define to use port B
#define use_portb_kbd TRUE

// Make sure the port used has pull-up resistors (or the LCD) on
// the column pins

#if defined use_portb_kbd
    #byte kbd = getenv("SFR:PORTB")
#else
    #byte kbd = getenv("SFR:PORTD")
#endif

#if defined use_portb_kbd
    #define set_tris_kbd(x) set_tris_b(x)
#else
    #define set_tris_kbd(x) set_tris_d(x)
#endif

//Keypad connection:  (for example column 0 is B2)
//          Bx:

#ifndef blue_keypad ///////////////// For the blue keypad
#define COL0 (1 << 2)

```

```

#define COL1 (1 << 3)
#define COL2 (1 << 6)

#define ROW0 (1 << 4)
#define ROW1 (1 << 7)
#define ROW2 (1 << 1)
#define ROW3 (1 << 5)

#else ///////////////////////////////// For the black keypad
d
#define COL0 (1 << 4)
#define COL1 (1 << 5)
#define COL2 (1 << 6)
#define COL3 (1 << 7) //Fila agregada, modificación de bits

#define ROW0 (1 << 0)
#define ROW1 (1 << 1)
#define ROW2 (1 << 2)
#define ROW3 (1 << 3)

#endif

#define ALL_ROWS (ROW0|ROW1|ROW2|ROW3)
#define ALL_PINS (ALL_ROWS|COL0|COL1|COL2|COL3)

// Keypad layout:
char const KEYS[4][4] = {{'1','2','3','A'},
                         {'4','5','6','B'},
                         {'7','8','9','C'},
                         {'F','0','E','D'}};

#define KBD_DEBOUNCE_FACTOR 33 // Set this number to apx n/333 where
                           // n is the number of times you expect
                           // to call kbd_getc each second

void kbd_init() {
}

char kbd_getc( ) {
    static BYTE kbd_call_count;
    static int1 kbd_down;
    static char last_key;
    static BYTE col;

```

```

BYTE kchar;
BYTE row;

kchar='\0';
if(++kbd_call_count>KBD_DEBOUNCE_FACTOR) {
    switch (col) {
        case 0 : set_tris_kbd(ALL_PINS&~COL0);
                   kbd=~COL0&ALL_PINS;
                   break;
        case 1 : set_tris_kbd(ALL_PINS&~COL1);
                   kbd=~COL1&ALL_PINS;
                   break;
        case 2 : set_tris_kbd(ALL_PINS&~COL2);
                   kbd=~COL2&ALL_PINS;
                   break;
        case 3 : set_tris_kbd(ALL_PINS&~COL3);
                   kbd=~COL3&ALL_PINS;
                   break; //Aregar caso para columna 3
    }

    if(kbd_down) {
        if((kbd & (ALL_ROWS))==(ALL_ROWS)) {
            kbd_down=FALSE;
            kchar=last_key;
            last_key='\0';
        }
    } else {
        if((kbd & (ALL_ROWS))!=(ALL_ROWS)) {
            if((kbd & ROW0)==0)
                row=0;
            else if((kbd & ROW1)==0)
                row=1;
            else if((kbd & ROW2)==0)
                row=2;
            else if((kbd & ROW3)==0)
                row=3;
            last_key =KEYS[row][col];
            kbd_down = TRUE;
        } else {
            ++col;
            if(col==4)//Modificación para caso de columna 4
                col=0;
        }
    }
}
kbд_call_count=0;

```

```
    }
set_tris_kbd(ALL_PINS);
return(kchar);
}
```

CONCLUSIONES

Se pudo comprobar de primera mano una de las ventajas del uso de lenguajes de alto nivel en la programación de microcontroladores, en esta práctica fue de importancia el manejo de cadenas de caracteres y su comparación, esta es una tarea compleja de realizar en lenguaje ensamblador por lo cual se prefirió utilizar lenguaje de alto nivel, aunque el uso de memoria y recursos dependa del funcionamiento del compilador, la facilidad de lectura de código para encontrar errores de funcionamiento y la facilidad de escritura de código fuente son factores a considerar.

Práctica 11 Cerradura Electrónica

OBJETIVO

Hacer la etapa de control de una cerradura electrónica, la clave debe ser de 4 números, si la clave es correcta, se debe encender un led simulando el pulso de activación, si la clave es incorrecta al tercer intento se debe desactivar el teclado por un tiempo determinado.

MARCO TEÓRICO

Esta practica esta basada en la practica anterior, se puede realizar con los conocimientos usados para realizar la practica anterior.

Librería kbd_4x4.c

Esta librería es una versión modificada de la proporcionada por PIC C compiler, esta modificación se realizo para ser usada con el teclado matricial 4x4 ya que la librería original esta hecha para trabajar con un teclado matricial de 4x3 además de esto también se pueden modificar el carácter de entrada al presionar alguna tecla del teclado matricial.

DESARROLLO

Comenzaremos declarando variables de interés, usaremos 2 cadenas de caracteres contenido tanto la contraseña correcta y otro un valor nulo que almacenara la variable ingresada, según los requerimientos la cerradura, esta se debe bloquearse al tercer intento incorrecto por lo que se usara una variable para almacenar este dato.

```
int x=1; //Contador de caracteres digitados
int y=0;//contador de intentos
char k;// caracter obtenido del teclado
char c[]={ "05821"};//cadena de la contraseña
char d[]={ "00000"};//cadena para guardar las teclas
```

Caso la contraseña ingresada sea la correcta, se ejecutara un bucle “do-while” que imprimirá el mensaje de “Clave correcta” encendiendo el led que simula el pulso de activación y reiniciando el conteo de intentos erróneos a cero, en este estado, si se presiona la tecla asterisco el valor de la cadena que contiene la clave se limpiara así como tambien el mensaje en el LCD.

```

if(strcmp(d,c)==0)//si las cadenas son iguales
{
    do
        {//se imprime la clave es correcta
            k=kbd_getc();
            lcd_gotoxy(1,2);
            printf(lcd_putc,"Clave correcta");
            output_A(0x01);//enciende el led verde
            y=0;//se reinicia conteo de intentos
        }
    while(k!='F'); //sucede mientras no se vueva a presionar asterisco
    x=1;//se reinicia el conteo de caracteres
    d="00000";//se reinicia el valor de la cadena d
    printf(lcd_putc,"\\f");//se borra el mensaje
    delay_ms(50);
}

```

Caso la clave ingresada sea errónea por tercera vez se entrará en un bucle infinito bloqueando el teclado y encendiendo un buzzer.

```

if (y==3)
{//entra a un ciclo infinito del que no se puede salir
    while(true)
    {
        printf(lcd_putc,"\\fLlave bloqueada");
        delay_ms(15);//se imprime llave bloqueada
        output_A(0x06);//se enciende el led rojo y el buzzer
    }
}

```

Presentamos a continuación los materiales a utilizados en la práctica:

- LCD 16x2
- Teclado Matricial
- Cables
- Protoboard
- PIC18F4550 Uva
- Buzzer
- Ordenador para ejecutar nuestro programa

Implementación del Circuito

Se muestra el esquemático del circuito usado para la práctica

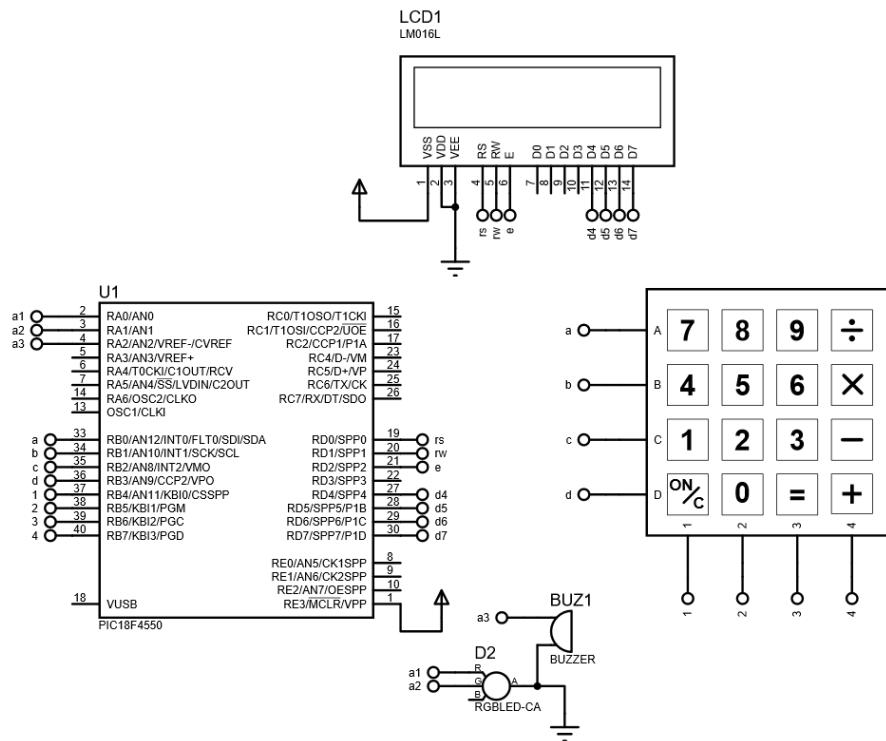


Figura 1: Esquemático Usado para la simulación del Circuito

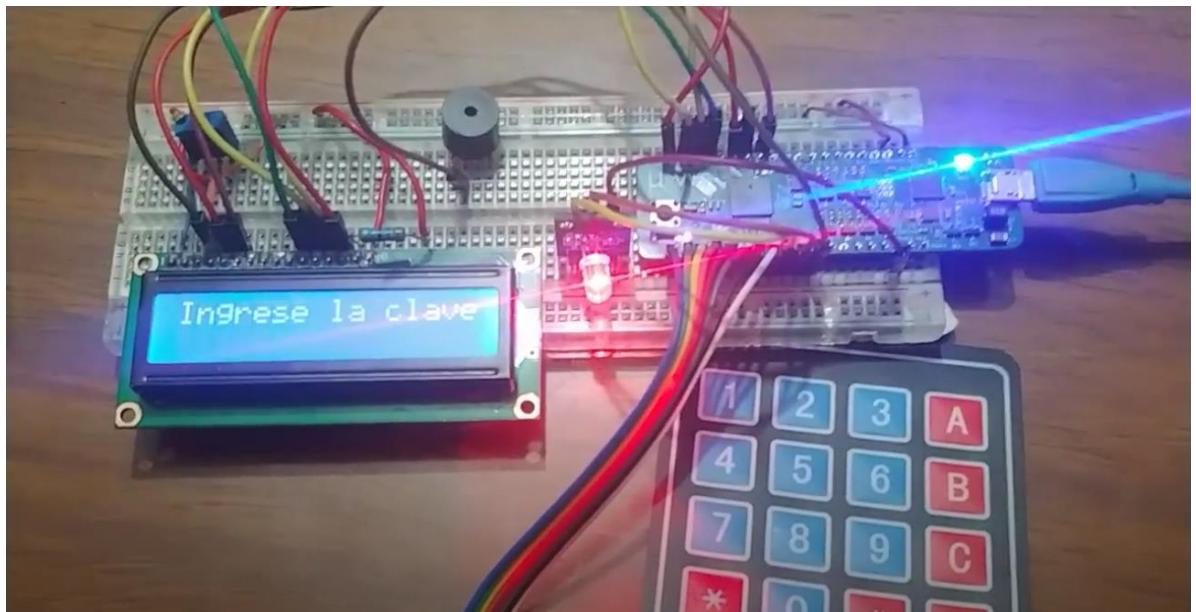


Figura 2: Implementación Física

Código Fuente

/*Este programa es para el desarrollo de una llave electrónica

la cual utiliza un teclado matricial para introducir datos y un display para la visualización, además de LEDs para ver el estado de la llave*/

```
//Encabezado, declaración de bibliotecas
#include <Práctica 11_1.h>
#include<stdio.h> /*Se declara para utilizar la función strcmp,
para la comparación de cadenas de caracteres*/
#use delay(clock=4Mhz)
#fuses hs,nowdt
#include<lcd.c>
#include <kbd_4x4.c> //libreria modificada para teclado matricial 4x4

//Declaración de variables
int x=1; //Contador de caracteres digitados
int y=0;//contador de intentos
char k;// carácter obtenido del teclado
char c[]={ "05821"}; //cadena de la contraseña
char d[]={ "00000"}; //cadena para guardar las teclas presionadas

void main()
{
    set_tris_A(0x00);
    lcd_init(); //inicialización del LCD
    kbd_init(); //inicialización del teclado matricial
    port_b_pullups(true); //Declaración pullups del puerto B
    do //ciclo infinito
    {
        k=kbd_getc(); //se obtiene la tecla digitada
        lcd_gotoxy(1,1);
        printf(lcd_putc, "Ingrese la clave");
        output_A(0x02); //Se enciende el led rojo es decir que la llave esta
bloqueada

        //condicional para escribir los digitos en la pantalla
        if((k!=0) && (k!='E') && (k!='F')) //si k es distinto de las teclas f
,e, y vacia
        {
            lcd_gotoxy(x,2);
            printf(lcd_putc, "%c",k); //imprime el carácter
            d[x]=k; //guarda el valor digitado en la cadena d
            x++; //incrementa el contador de caracteres
            //E es la en el teclado la tecla # y F es la tecla *
        }
    }
}
```

```

        }
        if((k!=0)&& (k=='F'))//si k es distinto del vacio y igual a F es decir
        {
            //la tecla de asterisco entonces se comparan las cadenas
            if(strcmp(d,c)==0)//si las cadenas son iguales
            {
                do
                {//se imprime la clave es correcta
                    k=kbд_getc();
                    lcd_gotoxy(1,2);
                    printf(lcd_putc,"Clave correcta");
                    output_A(0x01);//enciende el led verde
                    y=0;//se reinicia conteo de intentos
                }
                while(k!='F');//sucede mientras no se vueva a presionar asterisco
                x=1;//se reinicia el conteo de caracteres
                d="00000";//se reinicia el valor de la cadena d
                printf(lcd_putc,"\\f");//se borra el mensaje
                delay_ms(50);
            }
            else //en caso de que la clave sea incorrecta
            {//se imprime que la clave es incorrecta
                lcd_gotoxy(1,2);
                printf(lcd_putc,"CLAVE INCORRECTA");
                delay_ms(600);//se mantiene el mensaje 600ms
                x=1;//reinicia el conteo de caracteres
                y++;//incrementa el conteo de intentos
                d="00000";//reinicia la cadena d
                printf(lcd_putc,"\\f");//borra el mensaje
            }
        }
        //en caso de que la tecla # sea presionada
        if ((k!=0)&&(k=='E'))
        {
            X--;//disminuye el contador
            d[x]='0';//se elimina el valor digitado anteriormente
            lcd_gotoxy(x,2);
            printf(lcd_putc," ");//se borra solo el caracter digitado anteriormente
        }
        if (x==0)//en caso de que se presione muchas veces la tecla de borrar
        {
            x=1;//se asigna 1 para evitar errores en el pic
        }
    }
}

```

```

    }
    //en el caso de que se lleguen a los 3 intentos incorrectos
    if (y==3)
    {//entra a un ciclo infinito del que no se puede salir
        while(true)
        {
            printf(lcd_putc,"\\fLlave bloqueada");
            delay_ms(15); //se imprime llave bloqueada
            output_A(0x06); //se enciende el led rojo y el buzzer
        }
    }
    while(true);
}

```

CONCLUSIONES

El lenguaje de alto nivel es muy útil para aplicaciones complejas que involucren microcontroladores, tal y como se vio en esta práctica, la etapa de control de una cerradura electrónica escrita en lenguaje c implica varias llamadas a la biblioteca que controla el LCD, si se implementaría su equivalente en lenguaje ensamblador este proceso sería muy tedioso en la implementación, se concluye que la práctica se realizó con éxito.

PRÁCTICA 12 INTRODUCCION AL IOT

OBJETIVO

Controlar un led y hacer la lectura del sensor LM35 mediante una interfaz web usando la tarjeta Node MCU.

MARCO TEÓRICO

Node MCU

Esta es una tarjeta esta basada en el ESP8266, ofrece las opciones de ser usada con micropython o también con el IDE de Arduino.

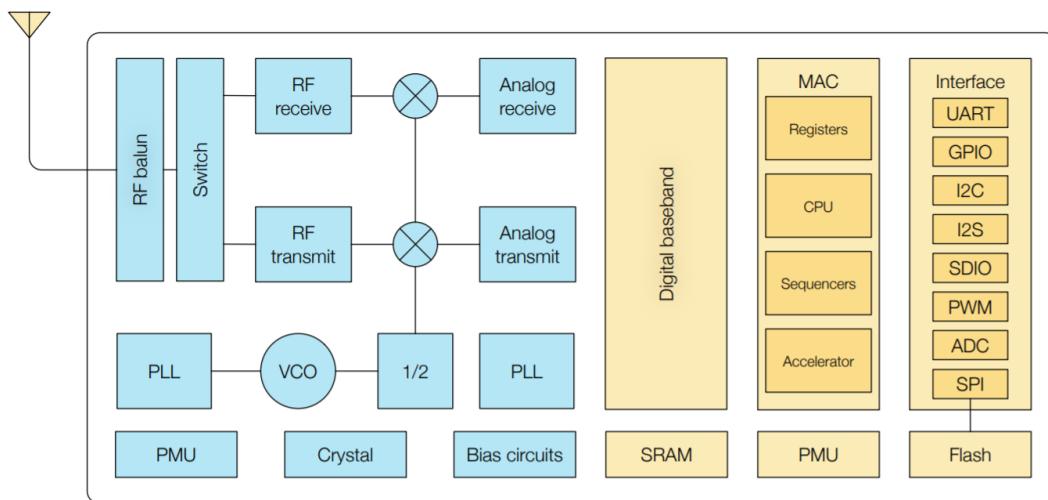


Figura 1 Diagrama funcional del ESP8266

Nos ofrece los siguientes pines

NODEMCU V2

NODE
PINOUT

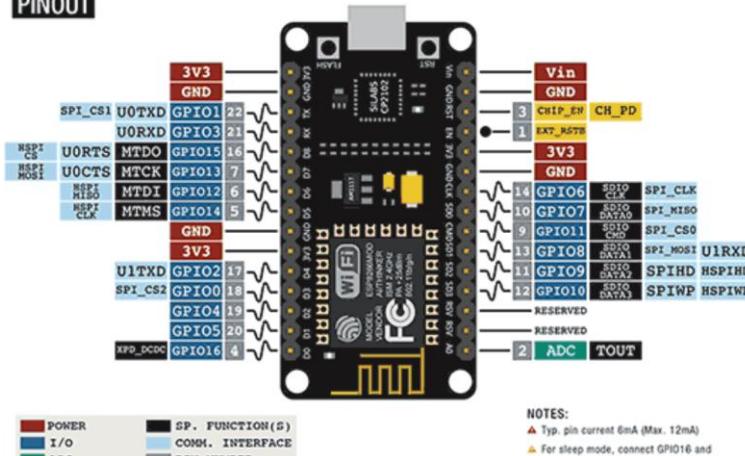


Figura 2 pines de salida de la tarjeta NodeMCU

Podemos observar que cuenta con varios pines de entrada y salida tanto digitales como analógicos lo cual hace que sea una tarjeta muy versátil.

Ubidots

Es una plataforma para el desarrollo de aplicaciones de internet de las cosas, ofrece una API que mediante una biblioteca del IDE de Arduino nos permite tomar los datos tomados con la tarjeta Node MCU y enviarlos a una interfaz web de su dominio, el flujo de nuestra información es el siguiente, la versión usada es la de Ubidots for Education que nos permite ver los tableros en vivo con la información adquirida.

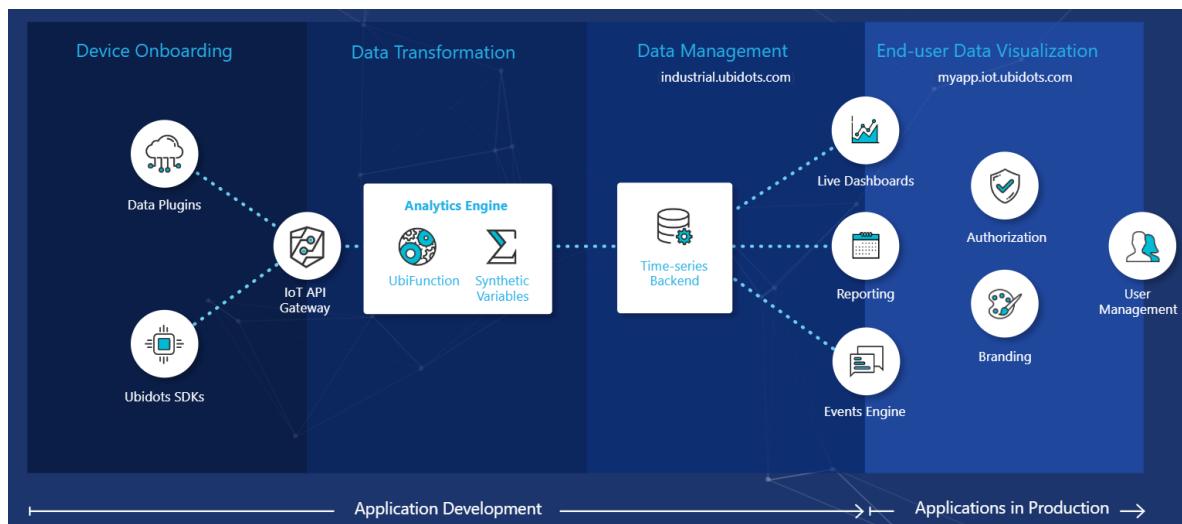


Figura 3: Flujo de Información de Ubidots

DESARROLLO

Se procederá usando la librería proporcionada "[UbidotsESPMQTT.h](#)",

```
#include "UbidotsESPMQTT.h"
#define TOKEN "BBFF-7VCypcONKsr2C2wTD5vxdNeIq7SbPu" // Your Ubidots TOKEN
#define WIFI_NAME "INFINITEUMBD67" //Your SSID
#define WIFI_PASS "unknown" // Your Wifi Pass
#define DEVICE_LABEL "tem" // Put here your Ubidots device label
#define VARIABLE_LABEL "switch" // Put here your Ubidots variable label
```

Se declararán una cadena que será el token en el servidor de Ubidots, después el nombre de la red de wifi, contraseña, nombre de los dispositivos y variables.

```

void setup() {
    // put your setup code here, to run once:
    client.ubidotsSetBroker("business.api.ubidots.com"); // Sets the broker properly for the business account
    client.setDebug(false); // Pass a true or false bool value to activate debug messages
    Serial.begin(115200);
    client.wifiConnection(WIFINAME, WIFIPASS);
    client.begin(callback);
    pinMode(16,OUTPUT);
    client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_LABEL); //Insert the dataSource and Variable's Labels
}

```

En la configuración de la tarjeta se iniciará la conexión con la API mediante las funciones de la biblioteca proporcionada.

Despues tomaremos los datos y se prodecera a su envio para su visualización en la plataforma web.

```

void loop() {
    // put your main code here, to run repeatedly:
    if(!client.connected()){
        client.reconnect();
        client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_LABEL); //Insert the dataSource and Variable's Labels
    }
    float temperatura;
    float value1 = analogRead(A0);
    temperatura=(value1*330)/1023;
    client.add("temperature", temperatura);
    client.ubidotsPublish(DEVICE_LABEL);
    client.loop();
}

```

Presentamos a continuación los materiales a utilizados en la práctica:

- Node MCU
- LM35
- Cables
- Led
- Protoboard
- Ordenador para ejecutar nuestro programa

Implementación del Circuito

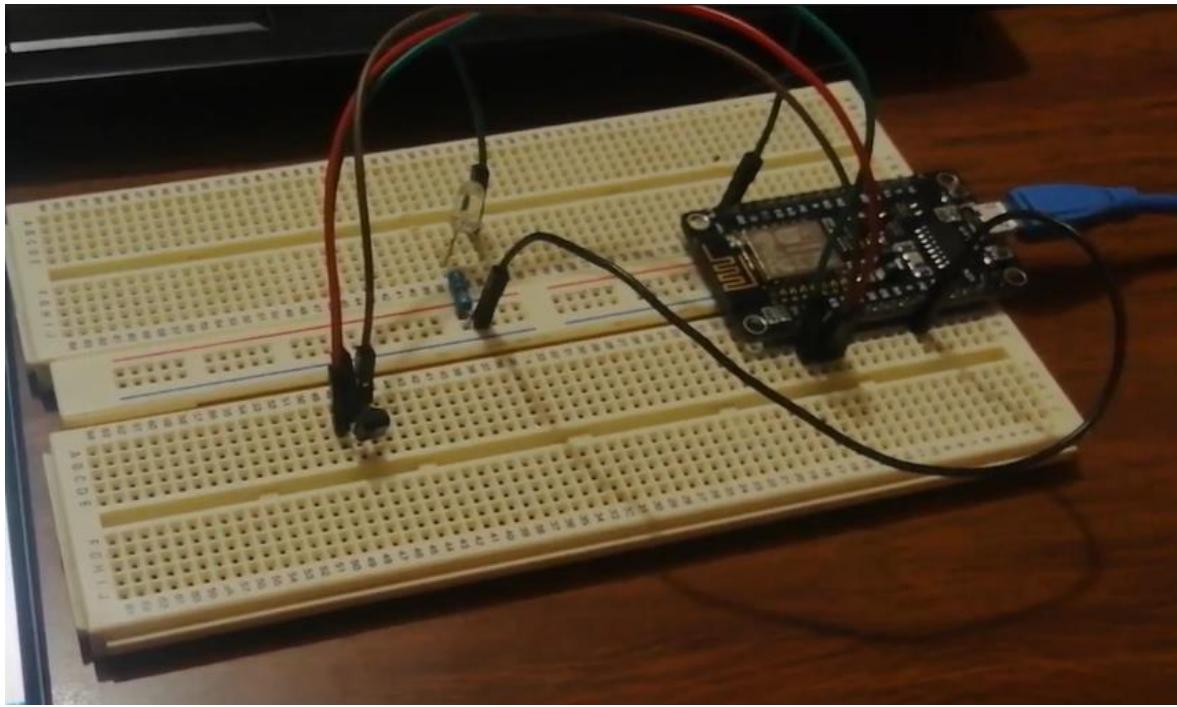
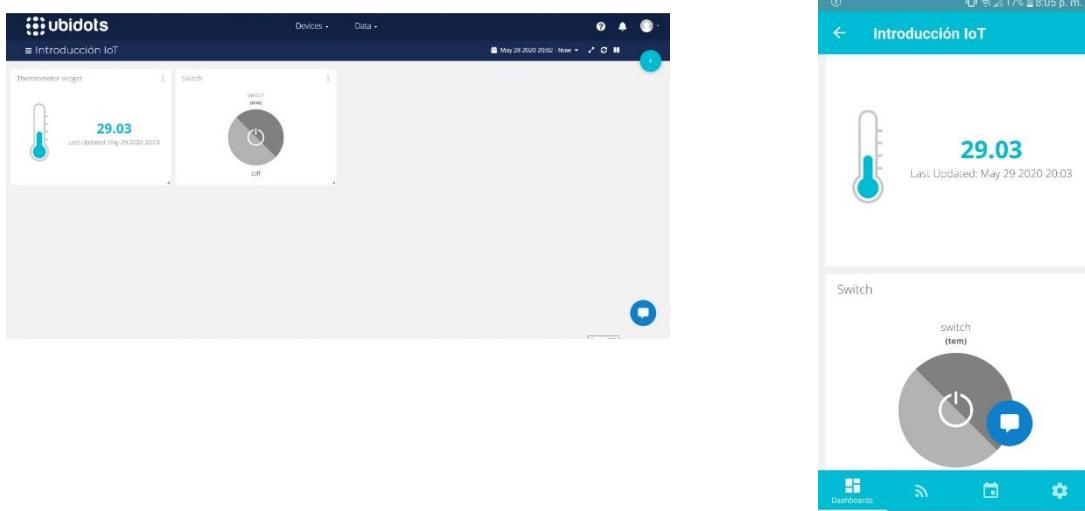


Figura 4: Implementación Física

La plataforma nos permite la visualización de los resultados y el control del led tanto en web como en el celular



Código Fuente

```
*****  
* Include Libraries  
*****  
#include "UbidotsESPMQTT.h"  
*****  
* Define Constants  
*****  
#define TOKEN "BBFF-7VCypcONKsr2C2wTD5vxdNeIq7SbPu" // Your Ubidots TOKEN  
#define WIFI_NAME "YA NO MAS INTERNET" //Your SSID  
#define WIFI_PASS "fer-ya*1105*" // Your Wifi Pass  
#define DEVICE_LABEL "tem" // Put here your Ubidots device label  
#define VARIABLE_LABEL "switch" // Put here your Ubidots variable label  
  
Ubidots client(TOKEN);  
  
*****  
*****  
  
void callback(char* topic, byte* payload, unsigned int length) {  
    Serial.print("Message arrived [");  
    Serial.print(topic);  
    Serial.print("] ");  
    for (int i=0;i<length;i++) {  
        Serial.print((char)payload[i]);  
    }  
    Serial.println();  
    if((char)payload[0]=='1')  
    {  
        digitalWrite(16,HIGH);  
    }  
    else  
    {  
        digitalWrite(16,LOW);  
    }  
}  
  
*****  
* Main Functions  
*****  
  
void setup() {  
    // put your setup code here, to run once:
```

```

client.ubidotsSetBroker("business.api.ubidots.com"); // Sets the broker pr
operly for the business account
client.setDebug(false); // Pass a true or false bool value to activate deb
ug messages
Serial.begin(115200);
client.wifiConnection(WIFINAME, WIFIPASS);
client.begin(callback);
pinMode(16,OUTPUT);
client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_LABEL); //Insert the dataSo
urce and Variable's Labels
}

void loop() {
// put your main code here, to run repeatedly:
if(!client.connected()){
    client.reconnect();
    client.ubidotsSubscribe(DEVICE_LABEL, VARIABLE_LABEL); //Insert the da
taSource and Variable's Labels
}
float temperatura;
float value1 = analogRead(A0);
temperatura=(value1*330)/1023;
client.add("temperature", temperatura);
client.ubidotsPublish(DEVICE_LABEL);
client.loop();
}

```

CONCLUSIONES

La práctica desarrollada se enfocó en desarrollar una aplicación simple usando internet de las cosas con la tarjeta de desarrollo Node MCU para su programación con el IDE de Arduino se instalaron los drivers y se usó la plataforma de Ubidots que ofrece una simplicidad de uso muy conveniente para su integración a cualquier proyecto que se esté desarrollando, además de esto la versión para educación de la misma permite su uso sin costo alguno, dado esto se concluye que la práctica se desarrolló con éxito habiendo logrado el objetivo propuesto.