



Universidad Nacional Autónoma de México

Facultad de Ingeniería
División de Ingeniería Eléctrica
Ingeniería en Computación



Proyecto 3 “Voltímetro Digital”

Integrantes:

- Castelan Ramos Carlos
- Castillo Montes Pamela
- Hernández Jaimes Rogelio Yael

Materia: Microcomputadoras

Grupo: 01

Semestre: 2023-2

Fecha de entrega: 12 de mayo 2023



Reporte Proyecto 3

“Voltímetro Digital”

Círculo Alambrado

El sistema mínimo está formado por:

1. Microcontrolador PIC16F877A
 2. Cristal de cuarzo 20 MHZ
 3. 2 capacitores cerámicos de 22pF
 4. 1 push uttom de 2 pines
 5. 1 resistencia 10 KΩ
 6. 2 capacitores cerámicos 0.1 µF
 7. Alambre/Jumpers
 8. Protoboard
 9. Fuente de poder 5 Volts
 10. Transmisor Serial UART/TTL con salida USB

El sistema funcional del proyecto está dado por:

1. Display LCD - LM016L
 2. Dip Switch de 4 pines (3 utilizables)
 3. 2 potenciómetros de 10 KΩ
 4. 3 resistencias 10 KΩ

Alambrado simulado:

La simulación del alambrado de nuestro circuito realizado en Proteus fue muy parecido al del proyecto anterior, quitando algunos dispositivos como el switch de 8 pines junto con sus resistencias y agregando un potenciómetro de $10\text{ K}\Omega$ logramos implementar nuevas funcionalidades haciendo uso de un convertidor analógico digital. Dentro de esta simulación logramos optimizar el funcionamiento del alambrado dadas las características de este software de simulación, tal fue el caso del oscilador de frecuencia el cual no fue necesario alambrar ya que el PIC podía ser configurado directamente. Además esta simulación nos ayudó a probar de manera inmediata el código generado para el proyecto para así identificar problemáticas de primera instancia.

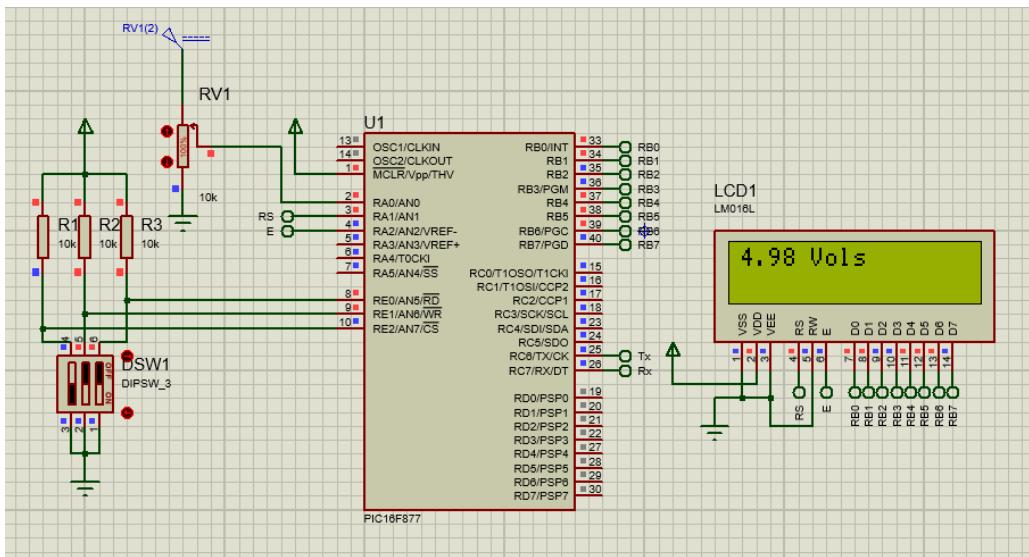


Imagen 1. Alambrado en Proteus



Alambrado físico

El alambrado físico del proyecto contempló más dispositivos al circuito final, por ejemplo un potenciómetro que regula el nivel de contraste en el display así como el oscilador necesario para el sistema mínimo. Dentro de este alambrado las fallas fueron nulas ya que gracias a el aprendizaje del proyecto pasado muy parecido a este, logramos implementar de forma eficiente el proyecto.

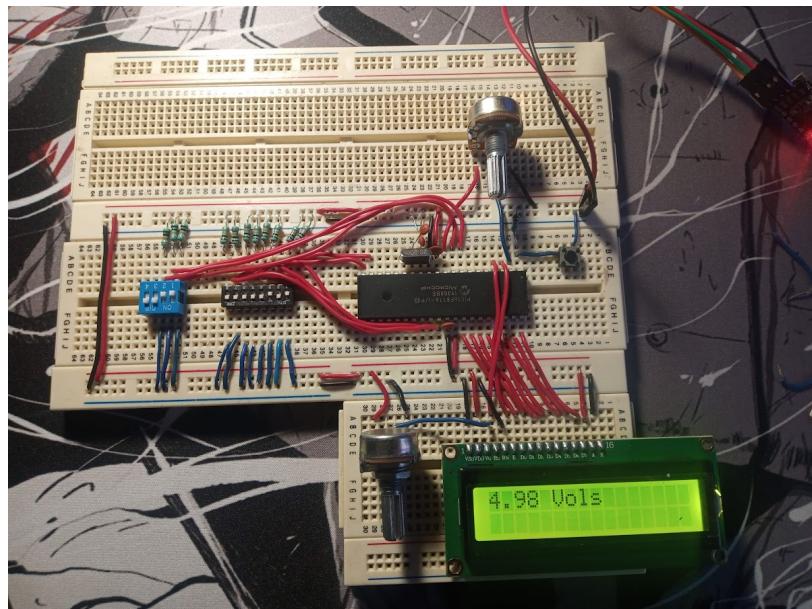


Imagen 2 Alambrado físico

Programa ASM

```
1      PROCESSOR 16F877
2      INCLUDE <P16F877.INC>
3
4      valor    equ h'20'          ;Registros auxiliares para rutina de retardo
5      valor1   equ h'21'
6      valor2   equ h'22'
7      contadorCentesima equ h'23' ;Registros auxiliares para conversión Hexa - Decimal
8      contadorDecima equ h'24'
9      contadorUnidad equ h'25'
10     aux      equ h'28'
11     regaux   equ h'26'          ;Registros auxiliares para visualizar en Hexadecimal
12     regaux2  equ h'27'
13
14     ORG 0 ;Vector de reset
15     GOTO INICIO
16     ORG 5
17
18 INICIO:    CLR PORTA ; Limpia PORTA
19           CLR PORTB ; Limpia PORTB
20           CLR PORTC ; Limpia PORTC
21           CLR PORTD ; Limpia PORTD
22           CLR PORTE ; Limpia PORTE
23           BSE STATUS,5 ; Cambia a banco 1
24           BCF STATUS,6
25           MOVWL 0x00
26           MOVWF TRISB ;TRISB<- 0x00 configuramos puerto B como salida
27           MOVWL H'0'      ; CONFIGURA PUERTO C COMO SALIDA
28           MOVWF TRISC ; (TRISC) <- 0h
29           MOVWL 0xFF      ; CONFIGURA PUERTO D COMO ENTRADA
30           MOVWF TRISD ; (TRISD) <- FFh
31           MOVWL 0xFF      ; CONFIGURA PUERTO E COMO ENTRADA
32           MOVWF TRISE ; (TRISE) <- FFh
33           MOVWL 0x07
34           MOVWF ADCON1 ;ADCON1 <- 0x07, configura puerto A y E como entradas digitales
35           MOVWL 0x00
36           MOVWF TRISA ;TRISA <- 0x00, configura puerto A como salida
37           BCF STATUS,5 ;Regresamos al banco 0
38           CALL INICIA_LCD
39           MOVWL 0x80
40           CALL COMANDO
```

Imagen 3 Configuración inicial de programa



El programa inicia declarando registros auxiliares para operar junto con el vector *reset* y la dirección de inicio del programa. Los registros auxiliares declarados tienen el siguiente uso:

- *valor, valor1 y valor2*: Los cuales son registros para operar en la subrutina de retardo y están relacionados a la conversión a segundos.
- *contadorCentesima, contadorDecima, contadorUnidad y aux*: Usados en la subrutina encargada de convertir el número obtenido a decimal.
- *regaux y regaux2*: Son registros auxiliares para almacenar valores que no deseamos perder durante la ejecución de instrucciones de la subrutina para mostrar el número ingresado en formato hexadecimal.

Prosiguiendo con la estructura del programa, encontramos la configuración de la PIC, donde primero se realiza una limpieza de los puertos paralelos con el objetivo de evitar errores de lectura o salida.

Nos cambiamos al banco 1 para poder configurar los registros TRIS así como otros necesarios y se realiza la configuración de los registros TRISA, TRISB y TRISC en 0 para posteriormente configurar PORTA, PORTB y PORTC como salidas.

Por otro lado, TRISD y TRISE le asignamos FF para hacer que los puertos D y E sean entradas para los dip switches, ADCON1 nos permite configurar el tipo de entrada que manejan los registros A y E respectivamente, con el valor 0x07 indicamos que todas sean entradas digitales. Regresamos al banco 0 y ejecutamos la inicialización de la LCD (goto INICIA_LCD). Esto para por último terminar ingresando un 0x80 a comando para posicionar el cursor en la primera posición.

<p>INICIA_LCD: MOVLW 0x30 COMANDO:</p> <p>CALL COMANDO</p> <p>CALL RET100MS</p> <p>MOVLW 0x30</p> <p>CALL COMANDO</p> <p>CALL RET100MS</p> <p>MOVLW 0x38</p> <p>CALL COMANDO DATOS:</p> <p>MOVLW 0x0C</p> <p>CALL COMANDO</p> <p>MOVLW 0x01</p> <p>CALL COMANDO</p> <p>MOVLW 0x06</p> <p>CALL COMANDO</p> <p>MOVLW 0x02</p> <p>CALL COMANDO</p> <p>RETURN</p>	<p>-----</p> <p>MOVWF PORTB</p> <p>RET200: RET200:</p> <p>CALL RET200 LOOP:</p> <p>BCF PORTA, 0 LOOP1:</p> <p>BSF PORTA, 1</p> <p>CALL RET200</p> <p>BCF PORTA, 1</p> <p>RETURN</p> <p>RET100MS: TRES:</p> <p>MOVWF PORTB</p> <p>CALL RET200</p> <p>BSF PORTA, 0 DOS:</p> <p>BSF PORTA, 1 UNO:</p> <p>CALL RET200</p> <p>BCF PORTA, 1</p> <p>CALL RET200</p> <p>CALL RET200</p> <p>RETURN</p> <p>-----</p> <p>MOVWF 0xAA</p> <p>MOVWF valor1</p> <p>MOVWF d'164'</p> <p>MOVWF valor</p> <p>DECFSZ valor, 1</p> <p>GOTO LOOP1</p> <p>DECFSZ valor1, 1</p> <p>GOTO LOOP</p> <p>RETURN</p> <p>MOVILW 0x03</p> <p>MOVWF valor</p> <p>MOVILW 0xFF</p> <p>MOVWF valor1</p> <p>MOVILW 0xFF</p> <p>MOVWF valor2</p> <p>DECFSZ valor2</p> <p>GOTO UNO</p> <p>DECFSZ valor1</p> <p>GOTO DOS</p> <p>DECFSZ valor</p> <p>GOTO TRES</p> <p>RETURN</p> <p>END</p>
---	---

Imágenes 4-6: Subrutinas control LCD

INICIA_LCD, COMANDO, RET200 y RET100MS son instrucciones puramente vistas en la teoría. Con estas configuramos la LCD para poder mostrar en pantalla y controlarla. Verdaderamente, esta secuencia está especificada por el fabricante por lo que no ahondamos demasiado en su ejecución. Se trata de un bloque de código fijo que se nos proporcionó para la elaboración del proyecto.

De forma general, INICIA_LCD realiza la configuración de la LCD, para lo cual define el formato de 8 bits, dos renglones del display y letras de 5x8; además del comando 0x06 para mantener el texto fijo (sin desplazamiento), así como el encenderlo y deshabilitar el parpadeo. Con esto se ejecuta la configuración base para operar el LCD.



Por su parte COMANDO permite enviar instrucciones de control al LCD. Esto permite simplemente cambiar la configuración del LCD. DATOS cumple con la función de imprimir caracteres en la pantalla. Ambas subrutinas generan una combinación particular entre los bits 0 y 1 de PORTA con lo que el LCD accede a modo de comando y de escritura.

Las rutinas de sub retardo son las vistas en la teoría. En esencia, dados tres valores cada uno se almacena en un registro valor. Durante la ejecución de la subrutina se tienen tres ciclos anidados, donde cada uno decrementa en uno el valor contenido en su respectivo registro valor. Con esta combinación se genera el retardo de 100 y 200 milisegundos aproximadamente.

Una vez configurado el LCD procedemos a pasar a la ejecución principal del programa. La función base del programa se encuentra definido con la etiqueta LOOP_P. En este simplemente leemos el valor de PORTE (donde se encuentran conectados los switches de selección) y aplicamos una operación AND 7. Esto hace que estrictamente los 3 bits de selección (conectados en los bits menos significativos de E) sean los únicos que se evalúen.

Para poder aplicar la operación el valor en PORTE pasa al registro W y sobre este se aplica la operación. Al haber extraído puramente los 3 bits menos significativos hacemos que haya un valor entre 0 y 7. Si esto lo sumamos al PC conseguimos que se desplace entre 0 y 7 instrucciones. De ahí que todas las instrucciones siguientes sean saltos incondicionales a los distintos casos. Si no es un caso contemplado para el proyecto se manda al caso DEFAULT que simplemente ejecuta este loop nuevamente.

Para los casos si definidos simplemente redireccionamos al caso adecuado:

```
37      LOOP_P:    CLRPF PORTD
38          MOVEF PORTE,W   ;W<-- (PORTE)
39          ANDLF 7        ;W <-- W&00000011
40          ADDWF PCL,F   ;(PCL)<-- (PCL)+W
41          GOTO  NOMBRES  ;PC+0  -> Switches: 000
42          GOTO  DECIMAL   ;PC+1  -> Switches: 001  Conversión entrada dipswitch de 8 bits a decimal
43          GOTO  HEXADECIMAL;PC+2  -> Switches: 010  Conversión entrada dipswitch de 8 bits a hexadecimal
44          GOTO  BINARIO   ;PC+3  -> Switches: 011  Conversión entrada dipswitch de 8 bits a binario
45          GOTO  CARACTER  ;PC+4  -> Switches: 100
46          GOTO  DEFAULT   ;PC+5  -> Switches: 101
47          GOTO  DEFAULT   ;PC+6  -> Switches: 110
48          GOTO  DEFAULT   ;PC+7  -> -> Switches: 111
```

Imagen 7: Case switch

Caso 1: Conversión a decimal

El siguiente caso hace referencia a la conversión del número recibido, el cual en este caso el número recibido desde el puerto varía de un rango [0 a 255] , es decir, no se considera aún una conversión a voltaje.

Para esto, se reuso el código propuesto en el proyecto 2, el cual se basa en una serie de restas como simulación de divisiones, las cuales nos permiten obtener el valor de centésimas, décimas y unidades.

Para realizar lo anterior, se tiene el uso de contadores, por lo que el primer paso es la inicialización de estos en 0, proseguimos con mostrar al display "D:" y seguido de lectura del valor en el puerto E y lo almacenamos en un registro auxiliar.



```
163 DECIMAL: CLRF contadorCentesima ;Inicializa en 0
164 CLRF contadorDecima ;Inicializa en 0
165 CLRF contadorUnidad ;Inicializa en 0
166 MOVlw a'D'
167 CALL DATOS
168 MOVlw 0x3A ;Dos puntos
169 CALL DATOS
170 MOVE PORTD,W ;Leer el valor de los switches y lo almacena en aux
171 MOVWF aux
```

Imagen 8: Inicialización de contadores

Para obtener el número de centésimas del número, se resta 100 al valor actual y en caso de que este nos arroja un valor positivo, se incrementa un contador de centésimas en uno y se repite este proceso hasta que obtengamos un valor negativo.

```
173 LOOP_Centesimas:MOVlw 0x64 ;Restar 100
174 SUBWF aux
175 BTFSC STATUS,C ;Verifica el estado de carry
176 GOTC CentesimaEncontrada ;SI hay carry, el resultado es un número positivo
177 MOVlw 0x64
178 ADDWF aux ;NO hay carry, entonces recuperar residuo

209 CentesimaEncontrada:INCF contadorCentesima
210 GOTC LOOP_Centesimas
```

Imágenes 9-10: Loop centésimas

El siguiente proceso empieza obteniendo el último valor positivo del proceso anterior, para esto se suma 100 al número actual y se procede a realizar la obtención de decimales, en este loop se realiza el mismo procedimiento que en el anterior, con la diferencia de que el número a restar es 10.

Cabe destacar que el último valor obtenido se le suma 10 y de esta manera obtenemos el número de unidades de número a convertir.

```
179 LOOP_Decimas: MOVlw 0x0A
180 SUBWF aux ;Restar 10
181 BTFSC STATUS,C ;Verifica el estado de carry
182 GOTC DecimaEncontrada ;SI hay carry, el resultado es un número positivo
183 MOVlw 0x0A
184 ADDWF aux ;NO hay carry, entonces recuperar residuo = Unidades
185 MOVE aux,w
186 MOVWF contadorUnidad ;Guardar en contador

212 DecimaEncontrada: INCF contadorDecima
213 GOTC LOOP_Decimas
```

Imágenes 11-12: Loop décimas

Por último, se muestran los valores de los contadores al display agregando h'30' para obtener su valor ascii y se llama a la subrutina HOLD_DECIMAL, para guardar el valor en display hasta que se realice un cambio.

```
188 MostrarDigitos: MOVE contadorCentesima,W
189 ADDLW 0x30 ;Obtener valor ASCII
190 CALL DATOS ;Display Centesimas
191 MOVE contadorDecima,W
192 ADDLW 0x30 ;Obtener valor ASCII
193 CALL DATOS ;Display Decimas
194 MOVE contadorUnidad,W
195 ADDLW 0x30 ;Obtener valor ASCII
196 CALL DATOS ;Display Unidades
197
198 HOLD_DECIMAL: MOVE PORTE,W
199 SUBLW 0x01 ;W<--W-0x10
200 BTFSC STATUS,Z ;¿(CONTA)=0X10?
201 GOTC HOLD_DECIMAL ;SI
202 MOVlw 0x01 ;NO, Limpia Display
203 CALL COMANDO
204 MOVlw 0x80 ;regresa a inicio linea 1
205 CALL COMANDO
206 CLRF W
207 GOTC LOOP_P
```

Imagen 13: Hold decimal



Caso 2: Conversión a Hexadecimal

El caso de conversión a HEXADECIMAL opera siguiendo la misma lógica del switch-case del LOOP_P. El valor resultante del conversor analógico-digital (almacenado en este punto en el registro conver) lo almacenamos en regaux2 para evitar perder el valor a convertir. En el registro W simplemente aplicamos una operación AND. Primero con 0xF0 para extraer la parte alta y 0x0F para la parte baja. Con esto extraemos 4 bits de la parte alta y 4 bits de la baja.

En el caso de la alta aplicamos una rotación a la derecha para convertirla en parte baja. Con esto, operamos con ambas partes por separado. HOLD_HEX simplemente retiene la ejecución del programa hasta que no se cambie de opción con los switches de selección, para evitar así que se esté recalculando todo el número constantemente. Para esto simplemente almacenamos el valor de PORTE en W y restamos 0x02, si son iguales (Z=0) entonces retenemos en el ciclo, si no, limpiamos la pantalla y regresamos al LOOP_P.

```
HEXADECIMAL:MOVIA a' H'
CALL DATOS
MOVIA 0x3A ;Dos puntos
CALL DATOS
CLRF W
MOVE conver,W
MOVWF regaux2
MOVE regaux2,W ;restauramos el valor guardado por si acaso sufriese algun cambio
ANDLW 0xF0 ;extraemos la parte alta
MOVWF regaux ;regaux <- W
RRF regaux,f
RRF regaux,f
RRF regaux,f
RRF regaux,f ;convertida en parte baja
CALL CONVERHEXA
MOVE regaux2,W ;restauramos el valor guardado por si acaso sufriese algun cambio
ANDLW 0x0F ;extraemos la parte BAJA
MOVWF regaux ;regaux <- W
CALL CONVERHEXA
HOLD_HEX: MOVF PORTE,W ;FUNCION PARA RETENER EL RESULTADO EN LCD
SUBIW 0x01 ; W<--W-0x30
BTFS STATUS,Z ;¿(CONTA)=0X20?
GOTO HOLD_HEX ;SI
MOVIA 0x01 ;NO, Limpia Display
CALL COMANDO
MOVIA 0x80
CALL COMANDO
CLRF W
GOTO LOOP_P
```

Imagen 14: Lectura y hold de Hexa

CONVERHEXA es una subrutina que aplica un switch-case al valor contenido en regaux (4 bits parte baja ocupados). Este valor se puede sumar al PC, siendo que tendrá un rango de 0 a 15. Cada uno de los casos corresponde a un número en el sistema hexadecimal, por lo que al sumar al PC el valor de regaux hacemos que pase al caso directo de su respectiva letra.

```
CONVERHEXA: MOVE regaux,W ; W<- (regaux)
;ANDLW 15 ;W<-- W&00001111, el cuarto bit siempre está activo para las letras
ADDWF PCL,F ;(PCL)<-- (PCL)+W
GOTO CASO0 ;PC+0 Caso 0000: Numero 0
GOTO CASO1 ;PC+1 Caso 0001: Numero 1
GOTO CASO2 ;PC+2 Caso 0010: Numero 2
GOTO CASO3 ;PC+3 Caso 0011: Numero 3
GOTO CASO4 ;PC+4 Caso 0100: Numero 4
GOTO CASO5 ;PC+5 Caso 0101: Numero 5
GOTO CASO6 ;PC+6 Caso 0110: Numero 6
GOTO CASO7 ;PC+7 Caso 0111: Numero 7
GOTO CASO8 ;PC+8 Caso 1000: Numero 8
GOTO CASO9 ;PC+9 Caso 1001: Numero 9
GOTO CASOA ;PC+10 Caso 1010: Letra A
GOTO CASOB ;PC+11 Caso 1011: Letra B
GOTO CASOC ;PC+12 Caso 1100: Letra C
GOTO CASOD ;PC+13 Caso 1101: Letra D
GOTO CASOE ;PC+14 Caso 1110: Letra E
GOTO CASOF ;PC+15 Caso 1111: Letra F
```

Imagen 15: Case dígito hexadecimal



Estos CASOS simplemente pasan a W el valor correspondiente del número hexadecimal, así en el caso 0 pasan el valor de 0 y en el CASOF el de F. CONVEND simplemente escribe el valor de W en el canal de datos para el LCD.

```
CASO0:    MOVLW a'0'
           GOTO CONVEND   CASO9:    MOVLW a'9'
           GOTO CONVEND   GOTO CONVEND
CASO1:    MOVLW a'1'
           GOTO CONVEND   CASOA:    MOVLW a'A'
           GOTO CONVEND   GOTO CONVEND
CASO2:    MOVLW a'2'
           GOTO CONVEND   CASOB:    MOVLW a'B'
           GOTO CONVEND   GOTO CONVEND
CASO3:    MOVLW a'3'
           GOTO CONVEND   CASOC:    MOVLW a'C'
           GOTO CONVEND   GOTO CONVEND
CASO4:    MOVLW a'4'
           GOTO CONVEND   CASOD:    MOVLW a'D'
           GOTO CONVEND   GOTO CONVEND
CASO5:    MOVLW a'5'
           GOTO CONVEND   CASOE:    MOVLW a'E'
           GOTO CONVEND   GOTO CONVEND
CASO6:    MOVLW a'6'
           GOTO CONVEND   CASOF:    MOVLW a'F'
           GOTO CONVEND   GOTO CONVEND
CASO7:    MOVLW a'7'
           GOTO CONVEND   CONVEND:  CALL DATOS      ; Imprimimos el simbolo en el LCD
           GOTO CONVEND   CLRF W
CASO8:    MOVLW a'8'
           GOTO CONVEND   RETURN
```

Imagen 16: Case dígito hexadecimal

Caso 3: Conversión a Binario

La conversión binaria es un caso bastante directo. Simplemente evaluamos de bit en bit si este se encuentra encendido o apagado, si lo está (respectivamente) omite escribir el contra símbolo. Por ejemplo, BTFSC conver, 7 evalúa si el bit 7 está apagado, si lo está no escribe uno (CALL ES_UNO), y por el contrario, con BTFSS evalúa si está encendido, y si lo está, no escribe 0.

Esto se repite con todos los bits del registro conver. HOLD_BIN hace lo mismo que HOLD_HEXA, comparando el valor de PORTE con 0x03 (restándoles y viendo si Z=0), si no son iguales (Z=0) limpia la pantalla y retorna al LOOP_P. Con el orden que se evalúa cada bit, en la pantalla se muestra el bit más significativo a la izquierda y el menos a la derecha.

Las subrutinas ES_UNO y ES_CERO simplemente mueven a W el valor de 1 o 0 respectivamente, para posteriormente imprimirla en el LCD con la subrutina DATOS.

```
BINARIO:  MOVF conver,W
           MOVLW a'B'
           CALL DATOS
           MOVLW 0x3A      ;Dos puntos
           CALL DATOS
           :CLRF conver    ;PRUEBA DE FUNCIÓN
           BTFSC conver,7  ;Evaluamos si el bit 7 es 0
           CALL ES_UNO     ; No es cero, imprime 1
           BTFSS conver,7  ; Evaluamos si el bit es 1
           CALL ES_CERO    ; No es uno, imprime 0
           BTFSC conver,6  ;Se repite la misma lógica con cada bit
           CALL ES_UNO
           BTFSS conver,6
           CALL ES_CERO
           BTFSC conver,5
           CALL ES_UNO
           BTFSS conver,5
           CALL ES_CERO
           BTFSC conver,4
           CALL ES_UNO
           BTFSS conver,4
           CALL ES_CERO
           BTFSC conver,3
           CALL ES_UNO
           BTFSS conver,3
           CALL ES_CERO
           BTFSC conver,2
           CALL ES_UNO
           BTFSS conver,2
           CALL ES_CERO
           BTFSC conver,1
           CALL ES_UNO
           BTFSS conver,1
           CALL ES_CERO
           BTFSC conver,0
           CALL ES_UNO
           BTFSS conver,0
           CALL ES_CERO
           MOVF PORTE,W
           SUBLN 0x02 ; W<--W-0x30
           BTFSC STATUS,Z ; ;(CONTA)=0X30?
           GOTO HOLD_BIN ;SI
           MOVLW 0x01      ;Limpia Display
           CALL COMANDO
           MOVLW 0x80      ;regresa a inicio linea
           CALL COMANDO
           CLRF W
           GOTO LOOP_P
           ES_UNO:        MOVLW a'1'
                           CALL DATOS
                           RETURN
           ES_CERO:       MOVLW a'0'
                           CALL DATOS
                           RETURN
```

Imágenes 17-18: Conversión Binaria

Caso 4: Voltaje

Este caso consiste en mostrar el resultado de la conversión a niveles físicos de voltaje. Esto es, según el nivel leído en la LCD se mostrará el voltaje equivalente de dicho nivel. Para ello, planteamos la siguiente división: se tienen 255 niveles posibles (la conversión tiene resolución de 8 bits), con lo que por cada 51 niveles equivale a 1 V.



Por otra parte, de esos 51 niveles tendríamos 10 casos (10 decimales, 0 a 9), con lo que cada 5 niveles (omitiendo el nivel 0 que tendrá asignado el decimal 0) tendríamos un decimal más. De esta forma, el nivel leído de 56 equivaldría al voltaje de 1.1 V.

Por último, cada 5 niveles tendríamos nuevamente 10 casos (10 centésimas, 0 a 9), sin embargo, en esta ocasión el mínimo valor que podemos cubrir es de 0.02 por nivel. La idea del programa es relativamente sencilla, en el caso VOLTAJE llamamos a la subrutina UNIVOLT, que obtendrá el valor entero del voltaje.

El valor resultante de la conversión evaluamos su valor respecto a los niveles de referencias. La comparación se hace con el valor límite superior para cada nivel. De esta forma, para el 0 evaluamos que el valor convertido sea menor a 0x33 (nivel 51, donde empieza el nivel 1). Este patrón se repite para cada nivel, siendo el nivel 5 donde se llega tras haber determinado que ningún otro rango lo contiene.

```
UNIVOLT:    MOVF conver,W ; Caso para obtener el nivel   CASO2B:      CALL V2          ;2.00-2.99V: 2  la condic
              SUBLW 0x33    ; 0x33-W      51 niveles
              BTFS STATUS,C ;¿STATUS.C==1? -> 0x33<W
              GOTO CASO1A    ;NO, ES MAYOR EL NUMERO
              GOTO CASO0B    ;SI, ES MENOR EL NUMERO
              CALL V0        ;0.00-0.99V: 0  la condicion de perter
              MOVlw 0x00
              MOVWF regaux2
              GOTO RETOMAV
              MOVF conver,W
              SUBLW 0x66      ;0110 0110 NIVEL 102
              BTFS STATUS,C ;¿STATUS.C==1? -> 0x66<W
              GOTO CASO2A    ;NO, ES MAYOR EL NUMERO
              GOTO CASO1B    ;SI, ES MENOR EL NUMERO
              CALL V1        ;1.00-1.99V: 1  la condicion de perter
              MOVlw 0x33
              MOVWF regaux2
              GOTO RETOMAV
              MOVF conver,W
              SUBLW 0x99      ;1001 1001 NIVEL 153
              BTFS STATUS,C ;¿STATUS.C==1? -> 0x99<W
              GOTO CASO3A    ;NO, ES MAYOR EL NUMERO
              GOTO CASO2B    ;SI, ES MENOR EL NUMERO
              RETOMAV:       CALL V2          ;2.00-2.99V: 2  la condic
              MOVlw 0x66
              MOVWF regaux2
              GOTO RETOMAV
              MOVF conver,W
              SUBLW 0xCC      ;1001 1001 NIVEL 204
              BTFS STATUS,C ;¿STATUS.C==1? -> 0xCC<W
              GOTO CASO4A    ;NO, ES MAYOR EL NUMERO
              GOTO CASO3B    ;SI, ES MENOR EL NUMERO
              CALL V3        ;3.00-3.99V: 1  la condicion de perter
              MOVlw 0x99
              MOVWF regaux2
              GOTO RETOMAV
              MOVF conver,W
              SUBLW 0xFF      ;1111 1010 NIVEL 255
              BTFS STATUS,C ;¿STATUS.C==1? -> 0xFF<W
              GOTO CASO5A    ;NO, ES MAYOR EL NUMERO
              GOTO CASO4B    ;SI, ES MENOR EL NUMERO
              CALL V4        ;4.00-4.99V: 4  la condicion de perter
              MOVlw 0xCC
              MOVWF regaux2
              GOTO RETOMAV
              CALL V5        ;5V: 5  la condicion de perter
              MOVlw 0xFF
              MOVWF regaux2
              GOTO RETOMAV
              RETURN
```

Imágenes 19-20: Función UniVolt

Ahora bien, para los valores decimales y centesimales simplemente restamos al valor convertido el nivel entero. Por ejemplo, el valor leído de 1.1 (nivel 52) resta su entero 1 para resultar en 0.1. Esto lo hacemos con el fin de sintetizar los procesos de evaluación.

Ahora bien, dado a que este valor depende según la lectura, cuando determinamos la unidad la almacenamos en la variable resNivel, de donde recuperamos el valor en la rutina DECVOLT. El resto de la lógica es la misma, simplemente añadiendo casos de 6 a 9. Nuevamente comparamos el valor con el límite superior de cada nivel.



```
DECVOLT:    MOVE regaux2,W CASO2AD:    MOVE regaux,W
             SUBWF conver,W      SUBLW 0x10
             ;MOVF conver,W      BTFSS STATUS,C  CASO4BD:    CALL V4
             MOVWF regaux      GOTO CASO3AD      MOVlw 0x15
             SUBLW 0x06      GOTO CASO2BD      MOVWF regaux2
             BTFSS STATUS,C  CASO2BD:    CALL V2      GOTO RETOMADV
             GOTO CASO1AD      MOVlw 0x0B      MOVLW regaux,W
             GOTO CASO0BD      MOVWF regaux2      CASO5AD:    CALL V4
             CALL V0      GOTO RETOMADV      MOVlw 0x1F
             MOVlw 0x00      CASO3AD:    MOVLW regaux,W      BTFSS STATUS,C
             MOVWF regaux2      SUBLW 0x15      GOTO CASO6AD
             GOTO RETOMADV      BTFSS STATUS,C      GOTO CASO5BD
             MOVF regaux,W      GOTO CASO4AD      CALL V5
             SUBLW 0x0B      GOTO CASO3BD      MOVLW 0x1A
             BTFSS STATUS,C  CASO3BD:    CALL V3      MOVWF regaux2
             GOTO CASO2AD      MOVlw 0x10      GOTO RETOMADV
             GOTO CASO1BD      MOVWF regaux2      MOVLW regaux,W
             CASO1BD:    CALL V1 ;1.00-1      GOTO RETOMADV      SUBLW 0x24
             MOVlw 0x06      CASO4AD:    MOVLW regaux,W      BTFSS STATUS,C
             MOVWF regaux2      SUBLW 0x1A      GOTO CASO7AD
             GOTO RETOMADV      BTFSS STATUS,C      GOTO CASO6BD
             CASO6BD:    CALL V6      GOTO CASO6BD
             MOVLW 0x1F      MOVWF regaux2
             MOVWF regaux2      GOTO RETOMADV
             CASO7AD:    MOVLW regaux,W      CALL V6
             SUBLW 0x29      SUBLW 0x1F
             BTFSS STATUS,C      MOVWF regaux2
             GOTO CASO8AD      GOTO RETOMADV
             GOTO CASO7BD      CASO7BD:    CALL V7
             CALL V7      MOVLW 0x24
             MOVLW 0x1F      MOVWF regaux2
             MOVWF regaux2      GOTO RETOMADV
             CASO8AD:    MOVLW regaux,W      CASO8AD:    CALL V7
             SUBLW 0x2E      SUBLW 0x2E
             BTFSS STATUS,C      MOVWF regaux2
             GOTO CASO9AD      GOTO RETOMADV
             GOTO CASO8BD      CASO8BD:    CALL V8
             CALL V8      MOVLW 0x29
             MOVLW 0x29
             MOVWF regaux2      MOVWF regaux2
             GOTO RETOMADV      GOTO RETOMADV
             CASO9AD:    CALL V9      CASO9AD:    CALL V9
             MOVLW 0x2E      MOVLW 0x2E
             MOVWF regaux2      MOVWF regaux2
             GOTO RETOMADV      GOTO RETOMADV
             RETOMADV:   RETURN
```

Imágenes 21-24: Función DecVolt

De forma similar al calcular la décima, para la centésima restamos la décima para poder evaluar puramente los valores límite. Si no hiciéramos estas restas tendríamos un gran número de niveles. La lógica que sigue es la misma que los casos anteriores, solo cambiando los valores de referencia. En este caso se plantean niveles que equivalen a 0.02 V dada la resolución de la conversión.



```
;EVALUAMOS LAS CENTESIMAS, SE RESTA LAS DECENAS
CENVOLT:    MOVF regaux2,W ;CASO DE CENTESIMAS.
            SUBWF regaux,W ; Restamos a la conversión el nivel base
            MOVEF conver,W ; Caso para obtener el nivel entero de v0
            MOVWF regaux ; Guardamos el valor para restaurarlo en
            SUBLW 0x01 ; 0x01-W      1 niveles (0.0)
            BTFS STATUS,C ;STATUS.C==1? -> 0x01<W
            GOTO CASO1AC ;NO, ES MAYOR EL NUMERO
            GOTO CASO0BC ;SI, ES MENOR EL NUMERO
CASO0BC:    CALL V0 ;0.00-0.99V: 0 la condicion de pertenencia
            GOTO RETOMACV
CASO1AC:    MOVE regaux,W
            SUBLW 0x02 ;0x02-W 1 niveles (0.02)
            BTFS STATUS,C ;STATUS.C==1? -> 0x66<W
            GOTO CASO2AC ;NO, ES MAYOR EL NUMERO
            GOTO CASO1BC ;SI, ES MENOR EL NUMERO
CASO1BC:    CALL V2 ;Se imprime 2, la resolución minima para
            GOTO RETOMACV
CASO2AC:    MOVE regaux,W
            SUBLW 0x03 ;0x03-W 1 niveles (0.04)
            BTFS STATUS,C ;STATUS.C==1? -> 0x03<W
            GOTO CASO3AC ;NO, ES MAYOR EL NUMERO
            GOTO CASO2BC ;SI, ES MENOR EL NUMERO
CASO2BC:    CALL V4
            GOTO RETOMACV
Caso3AC:    MOVF regaux,W
            SUBLW 0x04 ;0x04-W 1 niveles (0.06)
            BTFS STATUS,C ;STATUS.C==1? -> 0x04<W
            GOTO CASO4AC ;NO, ES MAYOR EL NUMERO
            GOTO CASO3BC ;SI, ES MENOR EL NUMERO
CASO3BC:    CALL V6 ;3.00-3.99V: 1 la condicion
            GOTO RETOMACV
CASO4AC:    CALL V8
            GOTO RETOMACV
RETOMACV:   RETURN
```

Imágenes 25-26: Función CenVolt

Ahora bien, en los tres casos se emplean subrutinas V0, V1, etc. Esto es simplemente una rutina que imprime el valor indicado en la LCD. Estas subrutinas son:

Imágenes 27-28: Casos por voltaje

El caso principal se estructuraría de la siguiente forma:

```
VOLTAJE:      CALL UNIVOLT
                MOVLW a'.''
                CALL DATOS
                CALL DECVOLT
                CALL CENVOLT
                MOVLW a' '
                CALL DATOS
                MOVLW a'V'
                CALL DATOS
                MOVLW a'o'
                CALL DATOS
                MOVLW a'l'
                CALL DATOS
                MOVLW a's'
                CALL DATOS
```

Imagen 29: Caso principal voltaje

La idea es hacer la estructura X.XX Volts. Esto permite que el LCD muestre resultados como los ejemplos mostrados por el profesor.



Caso 5: Muestra de carga en pila

Por último, haciendo uso de CGRAM, generamos 6 diferentes caracteres, cada uno se relaciona con el nivel de llenado de la pila según el voltaje.

Para cada carácter se consideró un cuadrante de 5 columnas x 8 filas de bits, siendo encendido representado con 1 y apagado con 0, en donde la figura base de la pila es la siguiente

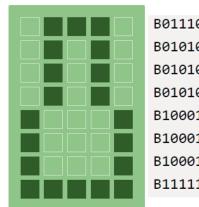


Imagen 30: Carácter Pila base

En los diversos casos de voltaje, se fue rellenando una fila dentro de los bits de columna delimitados por la pila para dar la ilusión de llenado. Cada uno de estos caracteres personalizados se guardó en un cuadrante diferente.

Imagen 3-: Carácter por caso de voltaje

Por lo que, para realizar la selección de alguno de estos se comparó el valor obtenido del nivel de voltaje, o bien, la unidad obtenida durante el proceso de conversión con la función *uniVolt*, dicho nivel se guardó en la variable *resNivel*.

Imagen 3-: Comparación voltaje

Donde cada caso está separado por restas de 1 unidad para ir a los casos de BATERIA/X según sea el caso, los cuales se encargan de posicionar el cursor en la primera línea del primer renglón del display. Y posteriormente, mandar a llamar al cuadrante almacenado adecuado para cada caso

Imagen 3-: Impresión de carácter

Finalizando cada caso, se manda a llamar a la función hold, similar a la usada en las otras casos, cuya existencia radica únicamente para que los caracteres no tuvieran la cualidad de parpadear.

```
HOLD_CAR:    MOVF  PORTE,W
              SUBLW 0x04 ;  W<--W-0x40
              BTFSC STATUS,Z ;¿ (CONTA)=0X40?
              GOTO HOLD_CAR ;SI
              MOVFW 0x01      ;Limpia Display
              CALL COMANDO
              MOVFW 0x80      ;regresa a inicio linea 1
              CALL COMANDO
              CLRF W
              GOTO LOOP_P
```

Imagen 3-: Función Hold carácter

Finalmente como parte del código se agregó un caso default que contempla todas las combinaciones no aprovechadas en el proyecto.



DEFAULT: GOTC LOOP_P

Imagen 34: Caso Default

Conclusiones:

Castelan Ramos Carlos

Para el desarrollo de este proyecto logramos implementar el uso de señales continuas y discretizadas a través de un convertidor analógico digital permitiendo establecer una funcionalidad de medición de entrada de voltaje, así también implementamos todo el conocimiento anteriormente aprendido aplicando la lógica de puertos paralelos, la definición de entradas y salidas con los puertos TRIs y finalmente la configuración del convertidor A/D que nuestro PIC implementa. No existieron complicaciones con el armado del circuito físico, en cambio con la parte lógica programática hubieron diversas situaciones en las cuales se tuvo que revisar y probar el código varias veces, finalmente considero que el proyecto se llevó a cabo con éxito en todos los rubros establecidos

Castillo Montes Pamela

En el desarrollo de este proyecto, pudimos implementar la lógica aprendida en clase para convertir el valor percibido por el PIC en uno con significado para nosotros, como lo es el voltaje obtenido después de una modulación de resistencia en un circuito. De igual forma, utilizando lo aprendido en el proyecto pasado, fue de fácil resolución, la integración de las funciones convertidoras de diversas bases numéricas para posteriormente su impresión en un display, en este caso LCD.

Por último, se logró profundizar más en el uso de CGRAM, como almacenamiento de caracteres personalizados, como lo es el caso de la pila. Como resultado, se obtuvo un proyecto modular adaptable a otros proyectos de mayor magnitud.

Hernández Jaimes Rogelio Yael

El uso del convertidor analógico-digital se ha vuelto una necesidad indispensable. Sabemos que en la realidad las señales a procesar son continuas, que si bien pueden ser convertidas en analógicas de formas relativamente sencillas, ni siquiera estas resultan útiles en la era digital. La digitalización de señales analógicas resulta indispensable para permitir que microcontroladores y microcomputadoras puedan operar con señales analógicas. Así pues, el PIC16F877A incorpora el conversor analógico-digital. La configuración de este CAD es sencilla, requiriendo definir si la entrada es digital o analógica en los puertos A y E, así como definir el canal de entrada al conversor. Esto es bastante parecido a la configuración de puertos paralelos (los cuales también interceden en la conversión). Esto permite que las señales analógicas como lo serían sensores de temperatura, humedad, presión y entre otros, se comuniquen con el microcontrolador, permitiendo así programas más específicos y de mayor utilidad para la automatización de procesos.