Trabajo de Laboratorio 1:

Arquitectura de Sistemas de Elaboración de Datos II

Carrera: Ingeniería en Informática

Docente: Ing. Federico Gabriel D´Angiolo

Integrantes:

*Calonge Federico.

*Cabot Lucas.

*Fernández Alejandro.

Índice

1-Introducción y objetivo.

Este trabajo de laboratorio consiste en desarrollar un sistema que cuente la cantidad de vueltas que realiza un auto de “Scalextric”, mostrando la información en dos displays de 7 segmentos.

Pudimos implementar un sensor o un pulsador para simular que pasó el auto y dió una vuelta, nosotros elegimos la segunda opción. De esta manera, al presionar el pulsador, los 2 displays muestran las vueltas correspondientes. Además, colocamos otro pulsador para el control de un reset (si este se presiona la cuenta visualizada en los displays vuelve a “00”). La cantidad de vueltas máxima visualizada es “99”, si sobrepasa este límite vuelve a “00” y empieza nuevamente. Por último, agregamos un led rojo que funciona como “señalización” (se prende) si es que el contador de vueltas llega a 6.

Todo este circuito (el **hardware**) es “controlado” por un **microcontrolador**, al cual lo programamos (mediante **software**) en un lenguaje de programación de bajo nivel, **Assembler**. Para que el microcontrolador realice todas las tareas que mencionamos anteriormente, necesita tener corriendo un programa en su memoria. Ese programa lo realizamos en un entorno de desarrollo en nuestro ordenador (MPLAB) y lo descargamos en el microcontrolador desde la PC con un programador (usando para esto el PICKIT 2). De esta manera, el micro luego de tener cargado el programa con las instrucciones necesarias, se encarga de por ejemplo: prender o apagar los transistores (y así hacer funcionar o no a los displays), de regular la velocidad de muestreo de los displays y los retardos correspondientes, de prender o apagar el led de señalización, etc.).

2- Detalle de funcionamiento.

Para la realización de este TL, utilizamos el microcontrolador PIC16F630. Este microcontrolador lo configuramos para que utilice su reloj interno (y no el externo como veníamos trabajando anteriormente). Esta decisión la tomamos con la finalidad de ahorrar patitas de conexionado, ya que este PIC tiene muy pocas patitas a comparación de otros, tiene solo 6 pines en el puerto C y 6 en el puerto A; y estos pines los tuvimos que utilizar para las distintas salidas (los 2 displays de 7 segmentos, 2 transistores, 2 pulsadores y 1 led) resultándonos insuficientes para utilizar el reloj externo utilizando un cristal/oscilador. Por esta razón, utilizamos el **reloj interno** del microcontrolador. En la ***Imagen 1*** podemos observar las patitas de conexionado del PIC16F30.

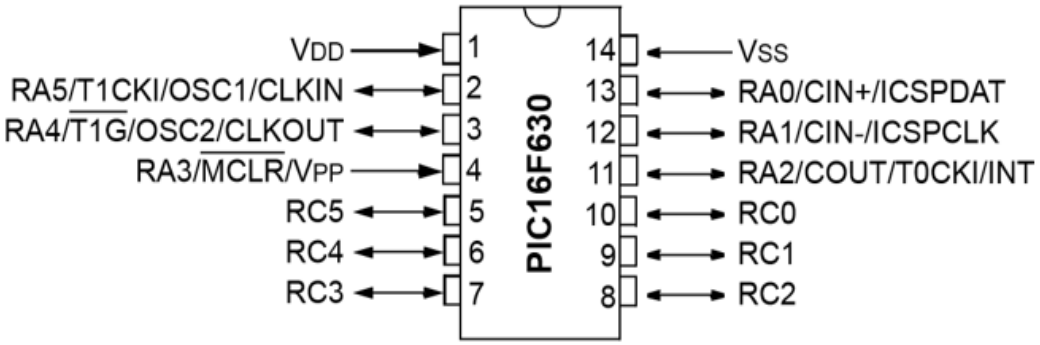


Imagen1

A continuación, se enumerara la lista de materiales para luego hacer una explicación detallada del funcionamiento de cada uno:

2.1- Lista de materiales utilizados.

- Microcontrolador PIC 16F630.
- Regulador de tensión de 9 a 5volts (L7805), pues para la alimentación se utiliza una batería de 9volts, estando establecida en 5 volts la tensión necesaria para el PIC.
- 2 displays de 7 segmentos de 1 digito cada uno (LTS547R).
- 2 pulsadores/botones, uno utilizado para incrementar el contador y otro para implementar el reset.
- 2 Transistores NPN BC548 utilizados para intercambiar el display de 7 segmentos a visualizar.
- 1 Led para visualizar que el contador llega a 6.
- Resistencias (3 de 100ohm, 2 de 1Kohm y 2 de 10Kohm).
- Protoboard y cables necesarios.

2.2- Descripción básica de funcionamiento.

Básicamente el microcontrolador PIC va incrementando la información de dos **displays de 7 segmentos** para mostrar números desde el 00 al 99. Los displays utilizados son de **cátodo común** (esto significa que tiene todos los cátodos de los diodos LED unidos, es decir están en común, y estos mismos están conectados a tierra, pero pasando por los transistores que eligen cual cátodo activar.)

. Y para activar un segmento de estos programamos al microcontrolador para que le mande un 1 lógico; 3,8V medidos, al ánodo del segmento, quedando este circuito como la ***Imagen 4***). En la ***Imagen 3*** están especificadas las "letras identificadoras" para cada led/segmento. Para mostrar la información, conectamos el primer display a 6 pines del PIC, que corresponden a todos los pines del puerto C (sí, son 6 pines en lugar de 7, ya que no nos alcanzaban los pines para conectar los 7 segmentos: a,b,c,d,e,f,g). Pero logramos mostrar correctamente la información, gracias a que el led superior del display (él A) fue puenteado con el led inferior (el D), es decir que cuando se enciende un 2, un 3, un 5, un 6, un 8, un 9 o un cero, hay dos pines que están en común y que pueden compartir el mismo pin. La imagen debajo ilustra muy bien esto, mostrando en rojo los leds del display que comparten un pin. La única desventaja fue que el 7 se muestra ligeramente diferente.

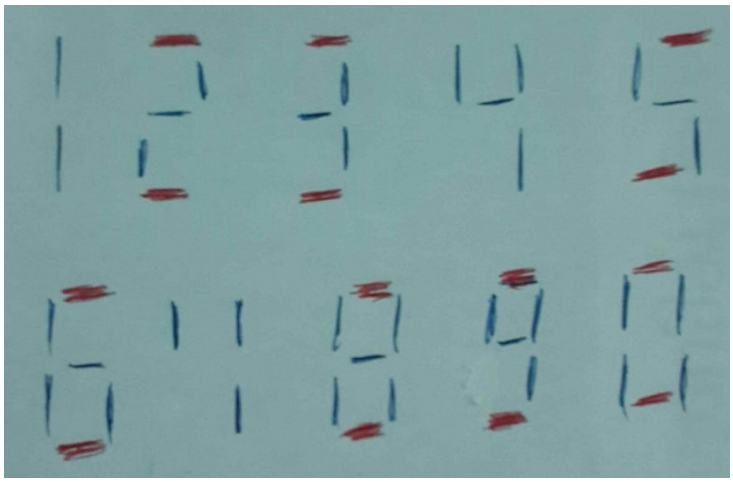


Imagen 2

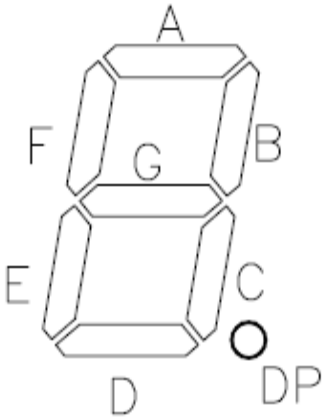


Imagen 3

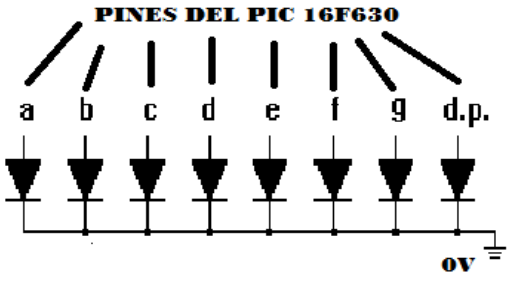


Imagen 4

Luego el segundo display fue conectado en modo puente al primero, el problema es que si comparten masa estarían mostrando el mismo número, para evitar este inconveniente, fueron conectados dos **transistores** a la entrada de masa de ambos displays, cada transistor al recibir un 1 activa un display, por lo que se logra enviar un numero de dos dígitos intercambiando rápidamente los transistores, es decir que para mandar por ejemplo el 06, se enciende el transistor del display izquierdo para mostrar el 0 y unos milisegundos después se enciende el segundo transistor apagando el primero, mostrando el 6 en el segundo display, el intercambio se hace **cada 37 milisegundos** (este es el tiempo de la rutina "Retardito" en nuestro programa en Assembler), este tiempo lo elegimos porque le da tiempo de sobra al transistor para descargar su tensión parasita que dura 5T (5 taos), ahorrándonos usar un circuito de descarga, el problema es que no podemos calcular ese tiempo exactamente porque no tenemos el dato de capacitancia parasita del transistor, pero asumimos que es extremadamente pequeño, tomamos 37ms simplemente por ser un tiempo grande y mínimamente suficiente para lograr el efecto deseado de frecuencia de refresco, podríamos haber tomado un tiempo menor, pero a los efectos prácticos no fue necesario. Con 37ms obtuvimos un refresco de 13Hz para cada display, con esto se logra la **ilusión óptica** de que se muestran en forma simultánea, por lo que el espectador observa el 06, aunque la realidad es que están intercambiando muy rápidamente. En la ***Imagen 5*** se puede apreciar el puente entre dos leds, el paralelismo de los displays y el intercambio con los transistores.

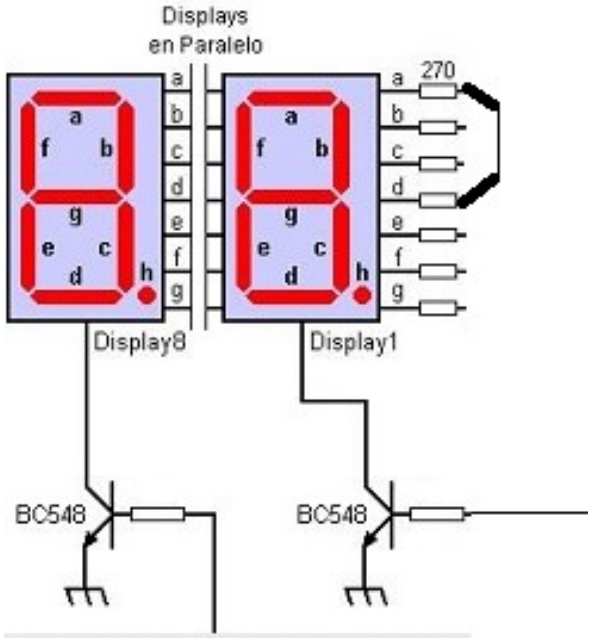


Imagen 5

Ambos transistores están conectados a dos pines del puerto A que están habilitados como salidas.

Por otra parte el **pulsador para pasar de un número a otro** está conectado a un pin del puerto A (RA2), habilitado como entrada. Cada vez que se pulsa el botón, se compara con un valor de referencia y se incrementa el número mostrado en el display. Cabe mencionar que se pulsa el botón, se hace un retardo de 780ms, para evitar que el contador cuente de más. En otras palabras, el microcontrolador con este retardo le da tiempo al usuario a soltar el botón. Lo que hay que destacar es el circuito de este pulsador; forma un circuito "PULL UP" (***Imagen 6***). De esta forma mi Vcc son los 5V y mi Vout sería el pin de entrada RA2 (el cual si se apretó el

pulsador está en 0v, 0 lógico y si no se apretó está en 5v, 1 lógico). Así, al apretarlo y mandar el 0 lógico en mi programa en Assembler le digo que pase al siguiente número con la instrucción btfsc. Esta instrucción se detalla de la siguiente manera:

btfsc registro,bit -> comprueba un determinado bit de un registro (f) y salta si el bit vale cero.

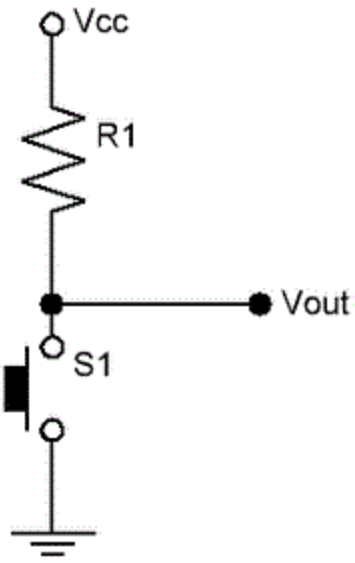
Con respecto al **pulsador para el RESET**, optamos hacerlo con el **pin MLCR**, el cual permite ser configurado mediante 3 formas:

1- Si se coloca a GND el micro entra en modo Reset, esto significa que el uC PIC no está en modo operativo.
2- Si se coloca a VDD el micro entra en modo operativo y cumple las funciones de correr el firmware grabado.
3- Si se coloca a VPP el micro entra en modo de programación y cumple funciones de edición de la memoria FLASH, donde podemos leer, grabar y borrar la memoria donde se aloja el firmware.

Nosotros queremos que cuando presionemos el pulsador, cambie de 5V a 0V, de esa forma funcionaria como RESET. El esquema que utilizamos es el de "PULL UP" (***Imagen 6***).

Siendo VCC=5V, los cuales están aplicados a la resistencia de 10k, lo cual disminuye la corriente a 0.5mA; a ese nodo (es decir a la salida de la resistencia), se conecta el pin RA3 (sería mi Vout). Este pin del puerto A tiene varias funciones, entre ellas la de MCLR nombrada anteriormente. Cuando es presionado el botón, Vout está conectado a masa, actúa como circuito cerrado, tengo 0v de salida, y el pic recibe un 0 lógico; cuando suelto el botón vuelvo al estado original, el botón en ese estado esta como circuito abierto, y el PIC recibe un 1 lógico.

Nota: Se puede optar por no utilizar la resistencia Pull Up (R1 en la ***Imagen 6*), pero es recomendable poner una resistencia para que no le llega tanta corriente a los pines del microcontrolador.



Pull Up

Imagen 6

2.3- Calculo de retardo.

El oscilador interno utilizado es de 4Mhz, esto quiere decir que hay 1 millón de ciclos de instrucciones por segundo. Para hacer el retardo utilizamos una rutina que consume 3 ciclos de instrucciones, es la siguiente expresada en modo genérico

decfsz contador

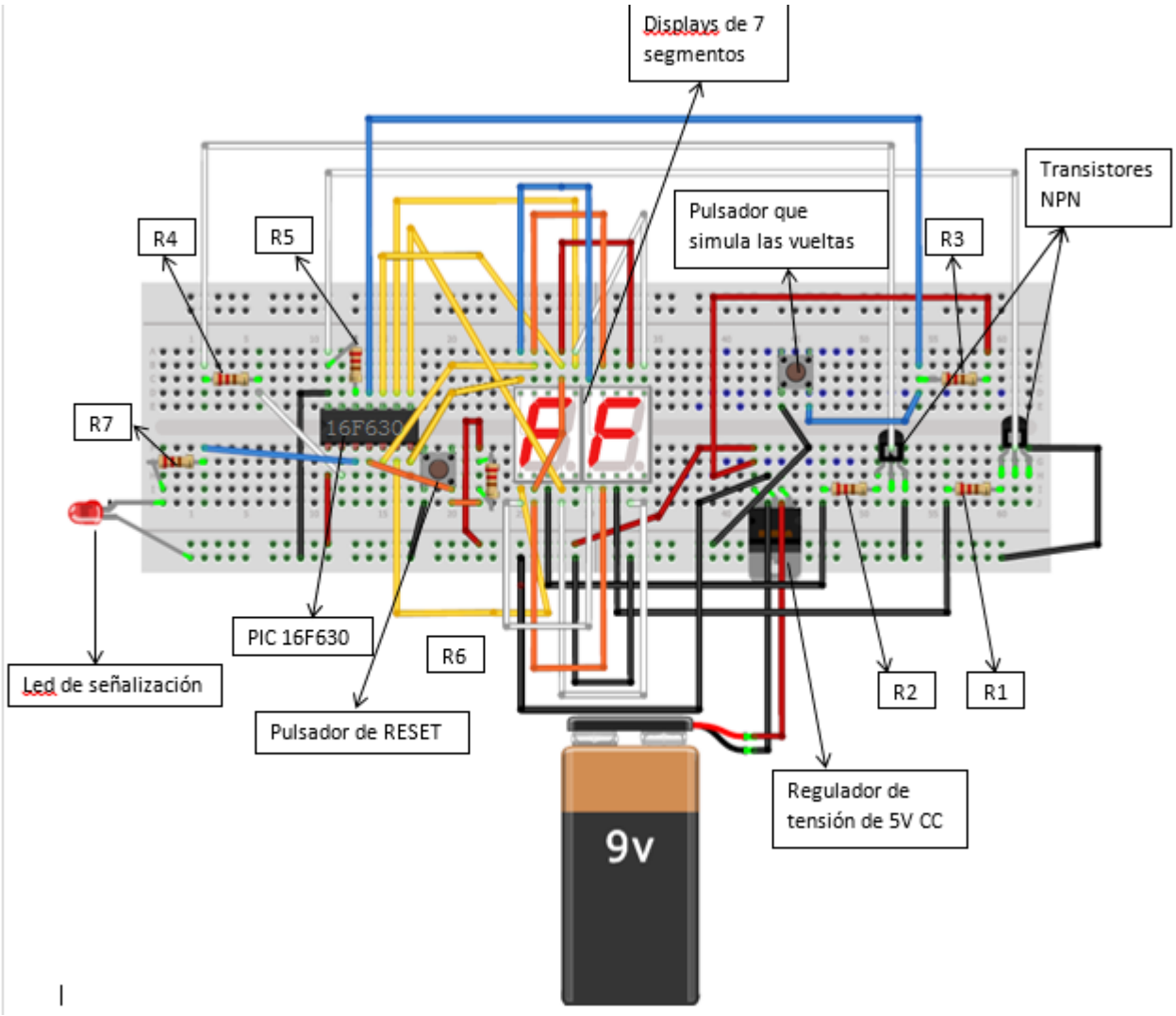
goto bandera

La instrucción decfsz consume 1 ciclo (excepto cuando llega a 0), y la instrucción goto consume dos ciclos, por cuestiones prácticas, resulta irrelevante que consuma 2 ciclos cuando llega a 0, ya que casi el 99% del tiempo está funcionando con un ciclo. Por lo tanto asumimos que ejecutar la rutina consume 3 ciclos siempre.

Por lo tanto si necesitamos establecer un retardo de 1 segundo, necesitamos ejecutar la rutina aproximadamente 333.000 veces. En nuestro caso establecimos en aproximadamente ¾ de segundo el tiempo que le damos al usuario a soltar el botón. Es suficiente con ejecutar la rutina 250.000 veces aproximadamente, para esto implementamos loops anidados, en este caso con 3 niveles es suficiente, de 64 ciclos cada nivel, lo que daría 64x64x64=262144. Entonces 262144*3=786432 ciclos, que representa aproximadamente ¾ de segundo, recordar que 1 millón es un segundo.

Para el retardo de 37ms, basta con 64x64=4096 ejecuciones de la rutina, lo que representa 4096*3=36864 ciclos de instrucciones, lo que equivale a 37ms

3.- Diagrama de conexionado.



Valores de las resistencias:

***R1, R2 y R7= 100ohm.*

***R3 y R6=1K.*

***R4 y R5= 1K.*

4-Mejoras a implementar.

-Usar un microcontrolador con más pines, como por ejemplo, el PIC16f84a; para no tener los problemas que tuvimos al ahorrar pines del micro (ya que sacamos 3, dos para el clock externo y uno del display).

-Optimizar el código para mostrar los números en los displays, ya que podíamos utilizar punteros a tablas para esto (detallado en el ejemplo 10 del apunte “PIC Programming in Assembly”).

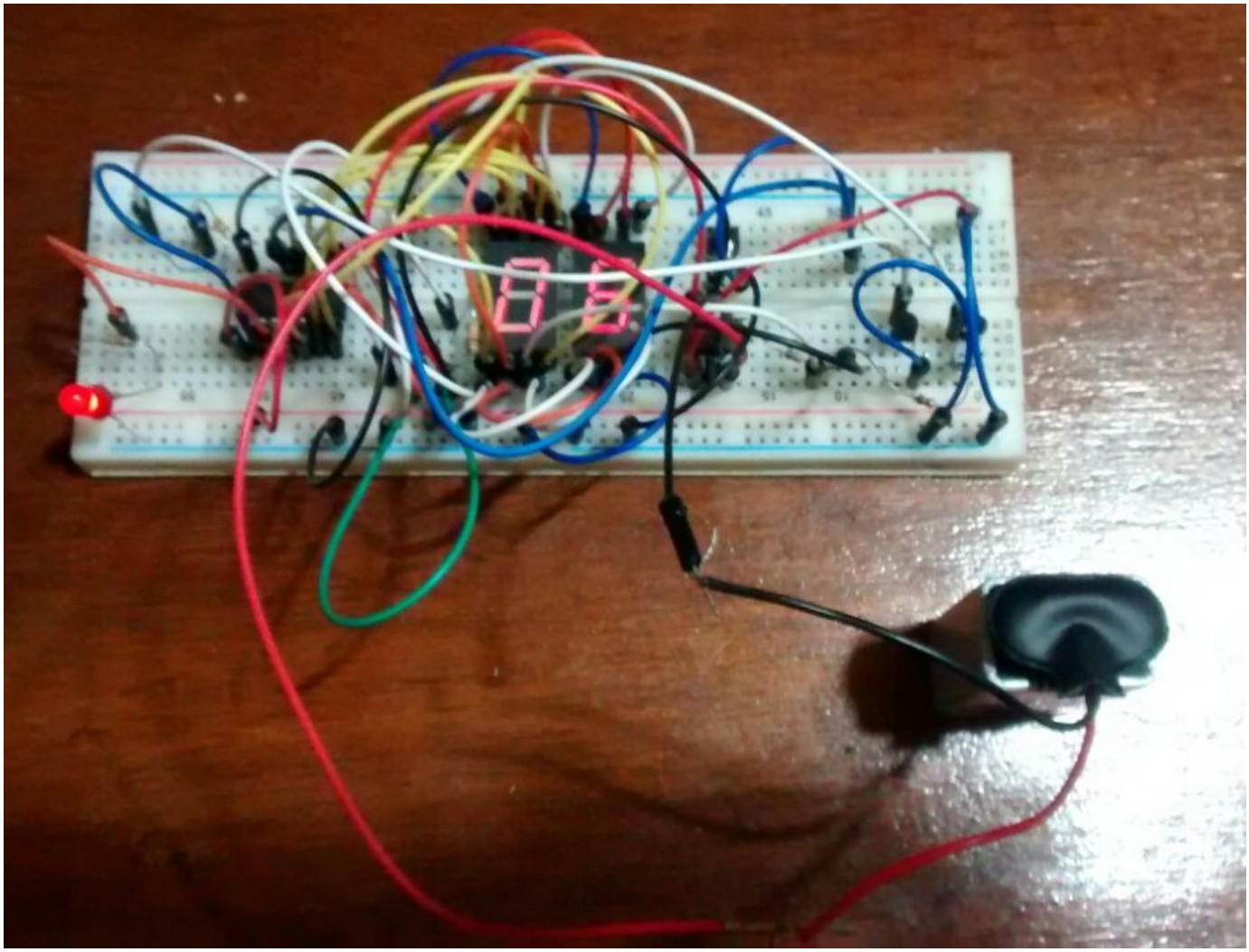
-Pasar todo el circuito a una placa de circuito impreso (PCB).

-Que al mantener apretado el pulsador para cambiar de número no lo cambie hasta que lo suelte; ya que en nuestro sistema, al mantenerlo apretado al pulsador va de 00 al 01, 02 y así hasta el final.

5-Conclusión.

Como cierre final del proyecto podemos decir que pudimos terminar en tiempo y forma con la realización del mismo, funcionando correctamente. Tuvimos algunos problemas en el transcurso del TL, pero que simplemente los solucionamos investigando detalladamente en los datasheets de los respectivos componentes. Logramos integrar los temas vistos en clases, como la programación en Assembler de un microcontrolador (que ya veníamos viendo desde Arquitecturas I) y la interacción con MPLAB y Pickit2.

Foto final del circuito:



6-Bibliografía utilizada.

Páginas web:

La siguiente web la utilizamos para visualizar los datasheets de los distintos componentes que usamos en el circuito: ***http://www.datasheetcatalog.com/

Esta web corresponde a ejemplos de programación en Assembler para PICS: ***http://www.mstracey.btinternet.co.uk/index.htm .El documento se llama "PIC Programming in Assembly".

Manuales:

*** Manual Conexión Pickit. Autor: D'Angiolo Federico Gabriel

*** Manual MPLAB X IDE1. Autor: D'Angiolo Federico Gabriel

7-Apéndice.

Código en Assembler que utilizamos para nuestro TL:

```
#include "p16F630.inc" ;se incluyen todos los parámetros del Micro
 __CONFIG _FOSC_INTRCCLK & _WDTE_OFF & _PWRTE_OFF & _MCLRE_ON & _BOREN_OFF & _CP_OFF & _CPD_OFF

org 0x0000 ;org setea el vector de inicio. PC en 0000. Debajo comienza el programa

;Defino los contadores en la zona de registro
contador0 equ 0x22
contador1 equ 0x24
contador2 equ 0x26
contador3 equ 0x28

#DEFINE led PORTA,4
#DEFINE pulsador PORTA,2
#DEFINE TIZQ PORTA,5
#DEFINE TDER PORTA,1
#DEFINE cero b'110111'
#DEFINE uno b'000110'
#DEFINE dos b'011011'
#DEFINE tres b'001111'
#DEFINE cuatro b'101110'
#DEFINE cinco b'101101'
#DEFINE seis b'111101'
#DEFINE siete b'100110'
#DEFINE ocho b'111111'
#DEFINE nueve b'101111'
```

```
    bsf STATUS,RP0 ;Pasamos al banco 1
    clrf TRISC ;Seteamos al Port C como SALIDA (todos los RC ahora son salidas).
    bsf pulsador ;Pulsador para cambiar de número como ENTRADA.
    bcf TIZQ ;Transistor del display izquierdo como SALIDA.
    bcf TDER ;Transistor del display derecho como SALIDA.
    bcf led ;Led de señalizacion como SALIDA.
    bcf STATUS,RP0 ;Pasamos al banco 0
    ;En el puerto C uso todos los pines (los 6) para el display.

INICIO
    BSF TIZQ
    BSF TDER
    BCF led  ;Apago el led
    call CERO
    goto INICIO

;Rutinas:
CERO       ;Tengo que prender todos menos la G para mostrar el 0:
;EL D ES EL QUE ESTA PUENTEADO CON A, de esta manera o se muestran los dos o ninguno.
BSF TIZQ
BSF TDER
movlw cero   ;En 1 estan prendidos, estos bits representan: F E G C B AYDjuntos (del display)
movwf PORTC
btfsc pulsador
goto CERO
call RETARDO  ;Esto es para evitar el rebote.
call UNO
```

```
UNO
BSF TIZQ
BCF TDER
movlw cero;
movwf PORTC
CALL RETARDITO
BCF TIZQ
BSF TDER
movlw uno;
movwf PORTC
CALL RETARDITO
btfsc pulsador ; ¿pulsador presionado? (pulsador == 0). Si lo presione salta a retardo.
goto UNO
call RETARDO  ;Esto es para evitar el rebote.
call DOS
```

```
DOS
BSF TIZQ
BCF TDER
movlw cero;
movwf PORTC
CALL RETARDITO
BCF TIZQ
BSF TDER
movlw dos;
movwf PORTC
CALL RETARDITO
btfsc pulsador ; ¿pulsador presionado? (pulsador == 0). Si lo presione salta a retardo.
goto DOS
call RETARDO  ;esto es psra evitar el rebote.
call TRES
```

(Desde el número 3 al 19 no lo pusimos porque el código para mostrarlos es prácticamente igual a la rutina/etiqueta UNO).

```
VEINTE
BSF TIZQ
BCF TDER
movlw dos;
movwf PORTC
CALL RETARDITO
BCF TIZQ
BSF TDER
movlw cero;
movwf PORTC
CALL RETARDITO
btfsc pulsador ; ¿pulsador presionado? (pulsador == 0). Si lo presione salta a retardo.
goto VEINTE
call RETARDO  ;Esto es para evitar el rebote.
call CERO
```

```
RETARDO
    movlw d'1'
    movwf contador0
    dec0
        movlw d'63'
        movwf contador1
        dec1
            movlw d'63'
            movwf contador2
            dec2
                movlw d'63'
                movwf contador3
                dec3
                    decfsz contador3
                goto dec3
                decfsz contador2
            goto dec2
            decfsz contador1
        goto dec1
        decfsz contador0
    goto dec0
return
```

```
RETARDITO
    movlw d'1'
    movwf contador0
    dec4
        movlw d'1'
        movwf contador1
        dec5
            movlw d'63'
            movwf contador2
            dec6
                movlw d'63'
                movwf contador3
                dec7
                    decfsz contador3
                goto dec7
                decfsz contador2
            goto dec6
            decfsz contador1
        goto dec5
        decfsz contador0
    goto dec4
return

end
```