# Universidad Nacional de Córdoba

# Facultad de Ciencias Exactas, Físicas y Naturales

# ELECTRÓNICA DIGITAL II

## Trabajo práctico I

*Informe*


**Profesores:**

- Del Barco, Martín
- Sánchez, Julio

**Alumno**: Bonino, Francisco Ignacio
**Matrícula:** 41279796
**Carrera:** Ingeniería en computación

# ÍNDICE

# Objetivos

El objetivo de este primer trabajo práctico es realizar un programa que permita sumar dos valores de 4 bits cada uno, ingresados desde puertos configurados como entradas al microcontrolador *PIC16F887*. El resultado debe mostrarse mediante cuatro LEDs conectados a puertos configurados como salidas. En caso de producirse un acarreo del tipo "digit carry", un quinto LED empezará a parpadear indefinidamente con un período de parpadeo de aproximadamente un segundo prendido y un segundo apagado. A partir de ese instante ya no podrán realizarse más sumas salvo que se realice un reset del microcontrolador. El programa será implementado con el software *MPLAB X IDE v5.35* y será simulado con el software *Proteus v8.6*.

# Desarrollo

## Configuraciones

Para comenzar se estableció que se trabajará con el microcontrolador PIC16F887 y se decidió trabajar con los puertos PORTC y PORTD configurados como salida y entrada respectivamente.

Sabiendo que los puertos con los que se trabaja son de 8 bits y las dos entradas deben ser de 4 bits cada una, se optó por ahorrar puertos y trabajar con uno solo para las entradas en vez de un puerto para cada una.

Por otro lado, se configuraron los "*configuration bits*" de forma que el "*watchdog timer*" esté "*OFF*" y el "*oscillator selection*" esté como "*EXTRC CLKOUT*" (desactivamos el timer watchdog para trabajar tranquilos y establecemos que trabajaremos con un clock externo).

A continuación, se crearon las variables necesarias para el desarrollo del programa:

- num1: Variable donde se almacenará el primer sumando.
- num2: Variable donde se almacenará el segundo sumando.
- iter1, iter2, iter3: Variables que se utilizarán para iterar los bucles.

Se estableció, además, el "*Program counter*" (PC) en 0x00 para comenzar el programa.

Finalmente, acorde a la información proporcionada en la hoja de datos del PIC16F887, accedemos al banco de memoria número 1 para setear a TRISD con unos (logrando así que PORTD quede configurado como entrada) y para setear a TRISC con ceros (logrando así que PORTC quede configurado como salida). Luego, volvemos al banco de memoria número 0 para trabajar con los puertos correctamente.

En el puerto PORTC se mostrará el resultado por los primeros cuatro bits (bits 0, 1, 2 y 3), y el bit que quedará intermitente será el bit 5. No acudimos al bit 4 porque ese bit formará parte del resultado si se llega a lograr un acarreo (considerando que la mayor suma podrá ser: 1111 + 1111 = 11110, donde el bit 4 estará en '1').

# Lógica del programa

En cuanto al resto del programa, podemos dividirlo en tres partes: suma, retardo y parpadeo. La lógica de trabajo de cada parte será explicada a continuación:

- ***Suma:*** Consta de leer la entrada en PORTD y separar los nibbles en dos variables distintas utilizando el operador *andlw* con los literales 0x0F y 0xF0. Luego, se verifica el estado del bit DC (digit carry):
  - Si está en '0', significa que no hubo carry de nibble, por lo que el programa puede seguir recibiendo entradas y operando.
  - Si está en '1', significa que hubo carry de nibble, por lo que el programa debe detenerse y dejar parpadeando un LED de forma indeterminada hasta que ocurra un reset por hardware del PIC.

    Sea cual sea el estado del bit DC, el resultado de la operación deberá mostrarse por el puerto PORTC.
- ***Retardo:*** Consta de tres bucles anidados que en total lograrán un retardo lo más aproximado posible a un segundo.
- ***Parpadeo:*** Consta de la operación "*prender LED, esperar un segundo, apagar LED, esperar un segundo*" repetida indefinidamente.

# Cálculo de los bucles

Para el cálculo de los bucles a utilizar con el fin de obtener un retardo de 1 (un) segundo implementado por software con *MPLAB* en el lenguaje de programación *Assembly*, acudimos a la estructura siguiente:

```
iterador = valor

iterador = iterador - 1

si iterador != 0, vuelvo a la línea anterior para decrementar nuevamente

si iterador = 0, salgo del bucle
```

En lenguaje *Assembly*, esto puede hacerse de la siguiente forma:

```
        iterador = valor
Loop    decfsz iterador
        goto Loop
```

El tiempo que tarda este bloque en ser ejecutado puede calcularse como:

$$T_1 = t_i(\text{valor} - 1) + 2t_i + 2t_i(\text{valor} - 1)$$

donde:

- $T_1$: Tiempo que tarde el bucle en ejecutarse completamente.
- $t_i$: Tiempo que tarda cada instrucción en ejecutarse (recordamos que el tiempo de instrucción es equivalente a cuatro ciclos de reloj).
- valor: Valor inicial asignado al iterador (cuántas veces se ejecuta el bucle).

Como estamos trabajando con un reloj de 4[MHz] de frecuencia, podemos ver que:

$$F_{clock} = 4[MHz] \Rightarrow t_i = 4\left(\frac{1}{4(10^6)}\right)[s] = \frac{1}{10^6}[s] = 1[\mu s]$$

Cada bucle puede tener un iterador con un valor tal que:

$$\text{valor} \in [0,256),\ \text{valor} \in \mathbb{Z}$$

Dado que cada instrucción tarda 1[µs] podemos calcular el tiempo que tardaría un bucle con su valor de iterador máximo:

$$T_1 = (1)(255 - 1) + 2(1) + 2(1)(255 - 1)\ [\mu s]$$
$$T_1 = 254 + 2 + (2)(254)\ [\mu s]$$
$$T_1 = 764\ [\mu s]$$

Vemos que aún estamos muy lejos de alcanzar el retardo de 1[s]. Para lograrlo, anidamos bucles de la forma:

```
        iterador1 = valor1

Bucle1  iterador2 = valor2

        Bucle2 decremento iterador2
               si interador2 != 0, vuelvo a Bucle2
               si iterador = 0, paso a la siguiente línea

        decremento iterador1
        si iterador1 != 0 vuelvo a Bucle1
        si iterador1 = 0, salgo del bucle
```

El cálculo del tiempo que tardarán en ejecutarse estos dos bucles es de la forma:

$$T_2 = 2valor_2 t_i + valor_2 T_1 + (valor_2 - 1)t_i + 2t_i + 2t_i(valor_2 - 1)$$

En el caso extremo de que

$$T_1 = 764\ [\mu s]\ (valor_1 = 255)$$

y

$$valor_2 = 255$$

tenemos:

$$T_2 = 2(255)(1) + (255)(764) + (255 - 1)(1) + 2(1) + 2(1)(255 - 1)\ [\mu s]$$
$$T_2 = 510 + 194820 + 254 + 2 + 508\ [\mu s]$$
$$T_2 = 196094\ [\mu s]$$
$$T_2 = 196.094\ [ms]$$

Estando aún lejos de lograr el retardo de 1[s], anidamos un tercer bucle de la misma forma que los anteriores, cuyo cálculo de tiempo de ejecución del bucle completo es:

$$T_3 = 2valor_3 t_i + valor_3 T_2 + (valor_3 - 1)t_i + 2t_i + 2t_i(valor_3 - 1)$$

Si hacemos:

$$T_1 = 764\ [\mu s]\ (valor_1 = 255),$$
$$T_2 = 196094\ [\mu s]\ (valor_2 = 255)$$

y:

$$valor_3 = 255$$

tenemos:

$$T_3 = 2valor_3 t_i + valor_3 T_2 + (valor_3 - 1)t_i + 2t_i + 2t_i(valor_3 - 1)\ [\mu s]$$
$$T_3 = 2(255)(1) + (255)(196094) + (255 - 1)(1) + 2(1) + 2(1)(255 - 1)\ [\mu s]$$
$$T_3 = 510 + 50003970 + 254 + 2 + 508\ [\mu s]$$
$$T_3 = 50005244\ [\mu s]$$
$$T_3 = 50.005244\ [s]$$

De esta forma vemos que nos hemos sobrepasado (y por mucho) el retardo de 1[s].

Luego de prueba y error, llegué a que los valores óptimos de $\text{valor}_1$, $\text{valor}_2$ y $\text{valor}_3$ son:

$$\text{valor}_1 = 255$$
$$\text{valor}_2 = 131$$
$$\text{valor}_3 = 10$$

De esta forma, obtenemos:

$$T_1 = 764 \text{ [µs]}$$
$$T_2 = 100476 \text{ [µs]} \ (T_2 = 100.476 \text{ [ms]})$$
$$T_3 = 1004809 \text{ [µs]} \ (T_3 = 1.004809 \text{ [s]})$$

Teniendo en cuenta que los retardos por software no suelen ser precisos, considero que esta aproximación es lo suficientemente buena para trabajar.

# Cálculo de las resistencias

Para este trabajo se utilizaron LEDs de distintos colores (amarillo y rojo), cuya caída de tensión se muestra a continuación:

| Color de LED | Caída de tensión |
|---|---|
| **Amarillo** | 2.1[V] a 2.18[V] |
| **Rojo** | 1.63[V] a 2.03[V] |

Y la situación a la que fueron expuestos es la siguiente:

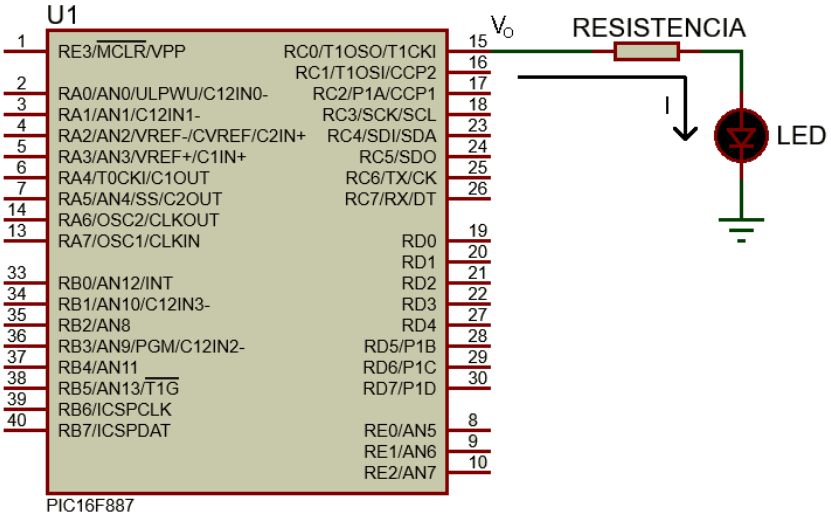


**Figura 1:** Conexión de un LED arbitrario en una salida de un puerto arbitrario del PIC16F887.

Siguiendo las ya conocidas leyes de Kirchhoff, planteamos la ecuación de tensión de nodos para la rama que se muestra en la Figura 1, obteniendo:

$$V_O - V_R - V_{LED} = 0$$

Donde:

- $V_O$ : Tensión de salida del PIC por sus puertos.
- $V_R$ : Tensión que cae en la resistencia.
  - Por ley de Ohm, esta es: $V_R = IR$.
- $V_{LED}$ : Tensión que cae en el LED.

Reescribiendo la ecuación, tenemos:

$$V_O - IR - V_{LED} = 0 \tag{E1}$$

## Para un LED amarillo

En este caso, tomamos una tensión de caída promedio de 2.14[V] para un LED amarillo. Es decir:

$$V_{LED} = 2.14\ [V]$$

Por otro lado, sabemos que la tensión de salida ofrecida por el PIC16F887 en sus puertos es de aproximadamente 4.7[V]. Esto es:

$$V_O = 4.7\ [V]$$

Finalmente, sabemos que la corriente óptima para manejar LEDs es de aproximadamente 10[mA], por lo que:

$$I = 10\ [mA]$$

Ahora reescribimos la ecuación (E1), quedando:

$$\begin{gathered} V_O - V_R - V_{LED} = 0 \\ V_O - IR_1 - V_{LED} = 0 \\ 4.7[V] - 10R_1[mA][\Omega] - 2.14[V] = 0[V] \\ 2.56[V] - 10R_1[mA][\Omega] = 0[V] \\ -(10R_1)[mA][\Omega] = -(2.56)[V] \\ 10R_1[mA][\Omega] = 2.56[V] \\ R_1 = \frac{2.56[V]}{(10)10^{-3}[A]} \\ R_1 = 256[\Omega] \end{gathered}$$

Llevado a valores comerciales, podemos decir que el valor de $R_1$ (resistencia para un LED amarillo) debe ser:

$$R_1 = 270[\Omega]$$

## Para un LED rojo

En este caso, tomamos una tensión de caída promedio de 1.7[V] para un LED rojo. Es decir:

$$V_{LED} = 1.7\ [V]$$

Tomamos los mismos datos de los cálculos para $R_1$, es decir:

$$V_O = 4.7\ [V]$$
$$I = 10\ [mA]$$

Ahora reescribimos la ecuación (E1), quedando:

$$V_O - V_R - V_{LED} = 0$$
$$V_O - IR_2 - V_{LED} = 0$$
$$4.7[V] - 10R_2[mA][\Omega] - 1.7[V] = 0[V]$$
$$3[V] - 10R_2[mA][\Omega] = 0[V]$$
$$-(10R_2)[mA][\Omega] = -3[V]$$
$$10R_2[mA][\Omega] = 3V]$$
$$R_2 = \frac{3[V]}{(10)10^{-3}[A]}$$
$$R_2 = 300[\Omega]$$

Llevado a valores comerciales, podemos decir que el valor de $R_2$ (resistencia para un LED rojo) debe ser:

$$R_2 = 330[\Omega]$$

# Diagrama de flujo

Se muestra a continuación el diagrama de flujo de este programa:

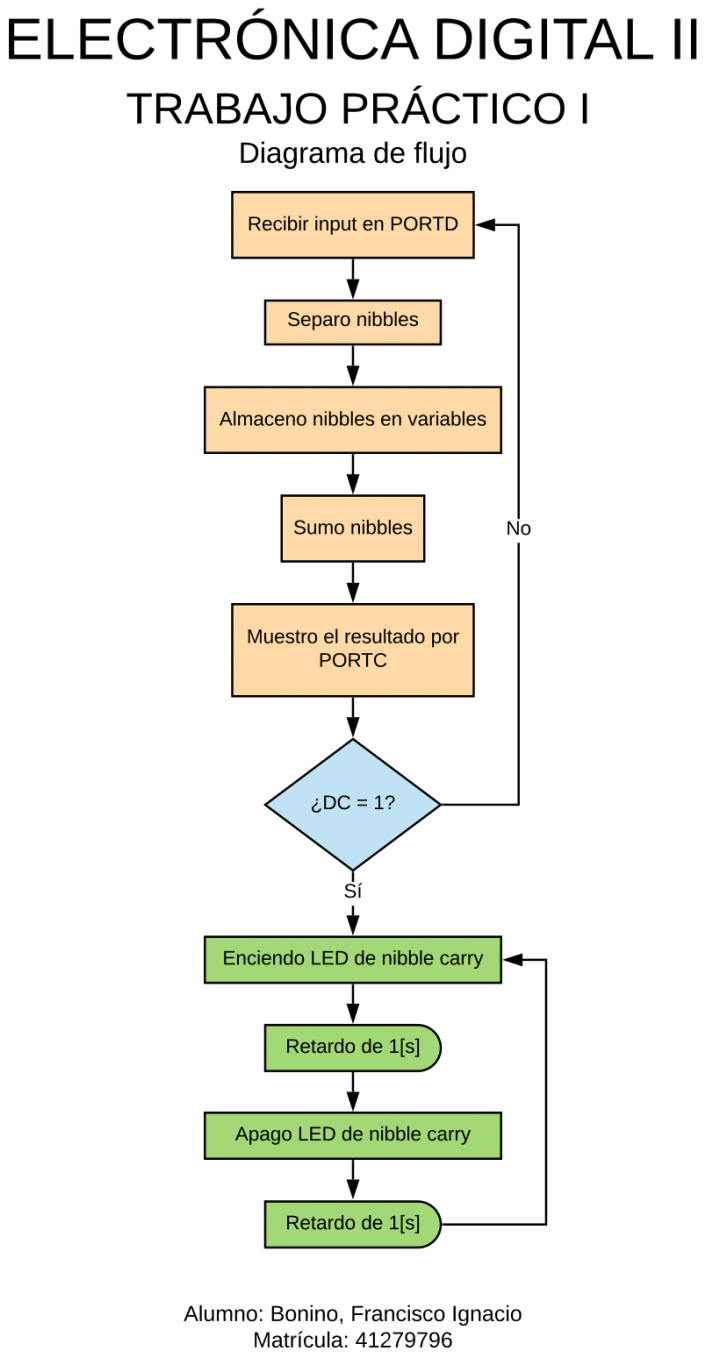


**Figura 2:** Diagrama de flujo del programa

# Circuito implementado

Se muestra a continuación una captura de pantalla del programa *Proteus*, con el circuito implementado para este trabajo práctico:

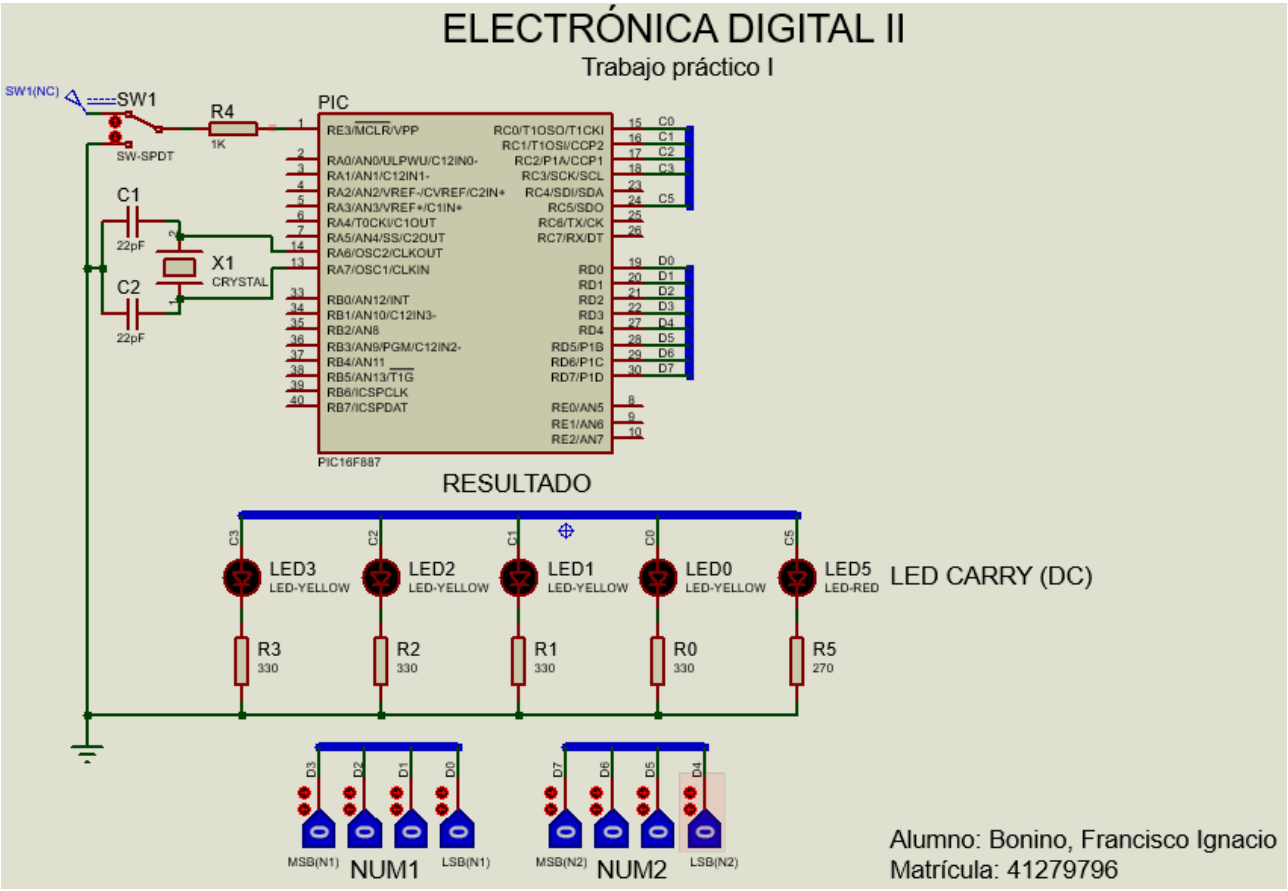


**Figura 3:** Implementación circuital en *Proteus*.

# Conclusiones

Con este primer trabajo práctico se logró familiarizarse con el software *MPLAB X IDE v5.35* para el desarrollo de programas simples en el lenguaje de programación *Assembly* y se logró un fructífero primer acercamiento a la forma de trabajar con un microcontrolador como es el PIC16F887.

Se logró comprender la estructura interna del microcontrolador, las instrucciones básicas para el desarrollo de programas para el mismo, y la correcta simulación en el software *Proteus*.