# ELECTRÓNICA DIGITAL II

Trabajo práctico final integrador

**UNC – FCEFyN**

*Ingeniería en Computación*

*Junio, 2020*


- **Integrantes:**
  - Merino, Mateo – 41232347
  - Bonino, Francisco Ignacio – 41279796
- **Profesor:**
  - Ing. Del Barco, Martín Ignacio.

# ÍNDICE

# Objetivos

El objetivo principal de este trabajo práctico final integrador de la materia es el de desarrollar un proyecto de forma teórica y práctica, aplicando los conceptos vistos en el semestre para lograr una profunda comprensión sobre la configuración y funcionamiento de un microcontrolador (en particular, el *PIC16F887* de *Microchip*), el manejo de registros y la interacción con periféricos.

Los requisitos mínimos para este trabajo práctico final integrador son:

- Implementar al menos tres fuentes de interrupción por periféricos.
- Implementar transmisión y recepción por *puerto serie*.

Deberá desarrollarse la totalidad del proyecto de forma teórica y práctica, con diagramas, cálculos y toda información explicativa que ayude a la comprensión del mismo.

# Introducción

## Trabajo a desarrollar

El tema elegido fue el desarrollo de un *teclado musical*. Este proyecto consta de un teclado simple de siete teclas, abarcando las notas *do, re, mi, fa, sol, la* y *si*. No tendremos en cuenta variaciones como notas *sostenidas, bemoles, disminuidas, aumentadas*, etcétera.

Además de los botones para las notas, el teclado tendrá un botón que activará la función de *metrónomo*, que constará de un LED que marcará un ritmo *4/4* con un pulso lumínico cada 1[s] (*tempo* = 60[BPM]).

Tendrá también la funcionalidad de *ingreso de partitura*. Esta función se podrá activar con un pulsador aparte. Esto consta de ingresar por puerto serie (recepción) seis notas codificadas en *cifrado americano* para mostrarlas por seis displays de 7 segmentos de cátodo común multiplexados. El objetivo de esta funcionalidad es la de que el usuario ingrese las notas de alguna melodía básica y pueda tenerlas a la vista para practicar, sin necesidad de recordar de memoria las notas y su orden.

Finalmente, tendrá la funcionalidad de *grabado de melodía*. Esta funcionalidad podrá accederse mediante otro pulsador. Cuando el teclado entre en estado de *grabado de melodía*, automáticamente se habilitará el puerto serie para transmisión de todas las notas musicales que el usuario ingrese. Estas notas pulsadas serán almacenadas en un archivo *.txt* y estarán codificadas en *cifrado americano*.

Cabe resaltar que tanto para la recepción como para la transmisión por puerto serie, se desarrolló una aplicación en lenguaje *Java* que puede comunicarse con el microcontrolador haciendo uso de librerías externas y de un simulador de puertos serie virtuales.

### Cifrado americano

Conocido más formalmente como *sistema de notación musical anglosajón*, el *cifrado americano* es un tipo de notación musical con base alfabética. Consta de la asignación de letras del alfabeto anglosajón a notas musicales. La tabla de equivalencia es la siguiente:

| **Nota** | **Cifrado** |
|---|---|
| *Do* | *C* |
| *Re* | *D* |
| *Mi* | *E* |
| *Fa* | *F* |
| *Sol* | *G* |
| *La* | *A* |
| *Si* | *B* |

*Tabla 1: Cifrado americano*

Si bien el *cifrado americano* también tiene su notación para notas *bemoles, sostenidas, menores, disminuidas, aumentadas* y demás, nosotros nos limitaremos a la Tabla 1 (notas mayores).

# Funcionamiento básico

Para el uso de las notas musicales, el usuario simplemente tendrá que encender el teclado accionando un *switch* que alimentará al microcontrolador con 5[V] continuos. Una vez alimentado el microcontrolador, se encenderá un LED verde indicando que el teclado está encendido. A partir de este momento, el usuario podrá accionar cualquiera de las teclas cuyos *labels* indican notas musicales (*DO, RE, MI, FA, SOL, LA, SI*), y se mostrará la letra del cifrado americano correspondiente a la nota presionada por un display de 7 segmentos conectado al puerto PORTD. Estas teclas abarcan los pines RB<7:1>.

Para accionar el metrónomo, el usuario deberá pulsar una tecla particular cuyo label es *"METRÓNOMO"* que está conectada al pin RB0. Cuando el usuario la pulse, un LED de color rojo comenzará a titilar marcando un ritmo cuyo tempo es de 60[BPM]. Mientras el metrónomo está funcionando, el usuario podrá seguir tocando una melodía, tomando como guía el metrónomo lumínico. Para salir del modo *metrónomo*, el usuario simplemente debe volver a tocar la tecla que lo activó.

Si se desea activar el modo de *ingreso de partitura*, el usuario deberá presionar el pulsador cuyo label es *"INGRESAR PARTITURA"* (se encenderá un LED amarillo) y, mediante la aplicación desarrollada en Java mencionada anteriormente, deberá acceder a la opción *"Enviar partitura"* del menú, e ingresar seis notas musicales en cifrado americano. La validación de las mismas se hace dentro de la aplicación. Cuando el usuario las ingrese, automáticamente se mostrará la secuencia de notas en 6 displays de 7 segmentos multiplexados para usarlo como partitura de guía.

Finalmente, si se desea usar el modo *grabado de melodía*, se deberá pulsar el botón cuyo label es *"GRABAR"*. Esto encenderá un LED azul que indica que se está grabando todo lo que el usuario pulse en RB<7:1>. Como complemento para un correcto funcionamiento de grabado, el usuario deberá acceder a la opción *"Grabar melodía"* del menú de la aplicación. Las notas que el usuario toque serán procesadas por esta aplicación que decodificará lo que lea por el puerto serie y lo transformará en cifrado americano, almacenando el resultado final en un archivo de texto en una ruta y con un nombre especificados por el usuario. Cuando se quiera dejar de grabar, basta con pulsar nuevamente el botón de *"GRABAR"* y automáticamente volveremos al estado inicial del teclado.

# Restricciones

El proyecto cuenta con ciertas restricciones. Las mismas se listan a continuación:

- No se podrán emitir dos o más notas musicales a la vez.
- No se podrá cambiar el *tempo* del metrónomo.
- El usuario sólo podrá ingresar por puerto serie *seis notas mayores en cifrado americano*.
- Lo grabado sólo podrá almacenarse en formato *.txt*.
- Si el usuario está grabando, no podrá hacer uso de la función *ingresar partitura*. Estas funcionalidades son mutuamente excluyentes para este proyecto.
- Si el usuario está ingresando una partitura, no podrá hacer uso del metrónomo ni de la función *grabar*.

# Herramientas de desarrollo

Se listan aquí las herramientas de desarrollo utilizadas para el trabajo:

- ***MPLAB X IDE v5.35***: Para el desarrollo y testeo del código en lenguaje *Assembly* que usará el microcontrolador.
- ***Proteus Professional v8.6***: Para la simulación por computadora del proyecto.
- ***Virtual Serial Port Driver by Eltima Software v9.0***: Para la simulación de puertos serie virtuales.
- ***GiovynetDriver***: Librería externa para manejar puertos series en Java.
- ***Visual Studio Code***: Para programar la aplicación en Java.

# Desarrollo teórico

## Diagramas de flujo

Comenzamos con el desarrollo teórico planteando los diagramas de flujo de alto nivel para el funcionamiento del programa:

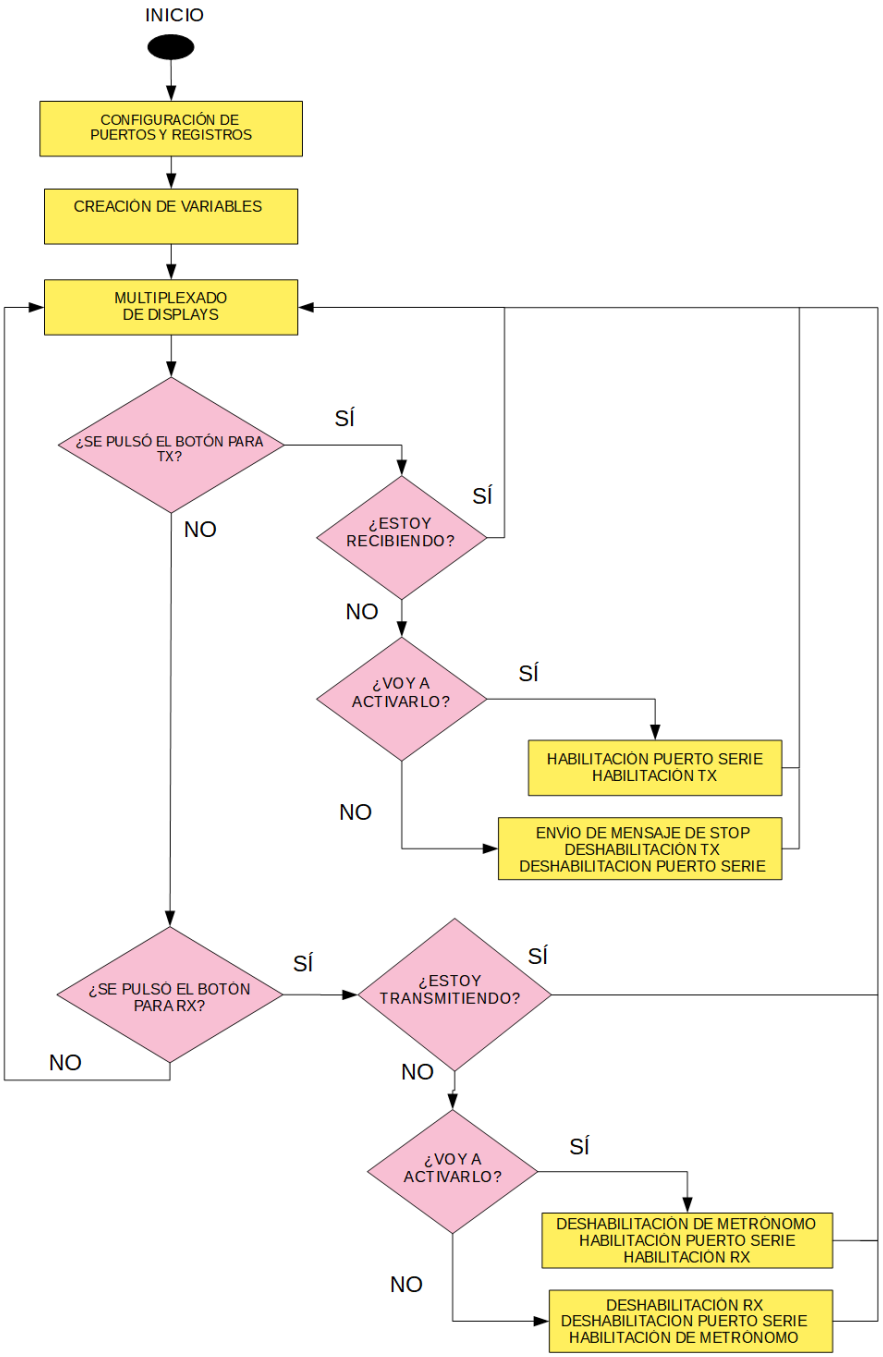


*Figura 1: Inicio, multiplexado y polling*

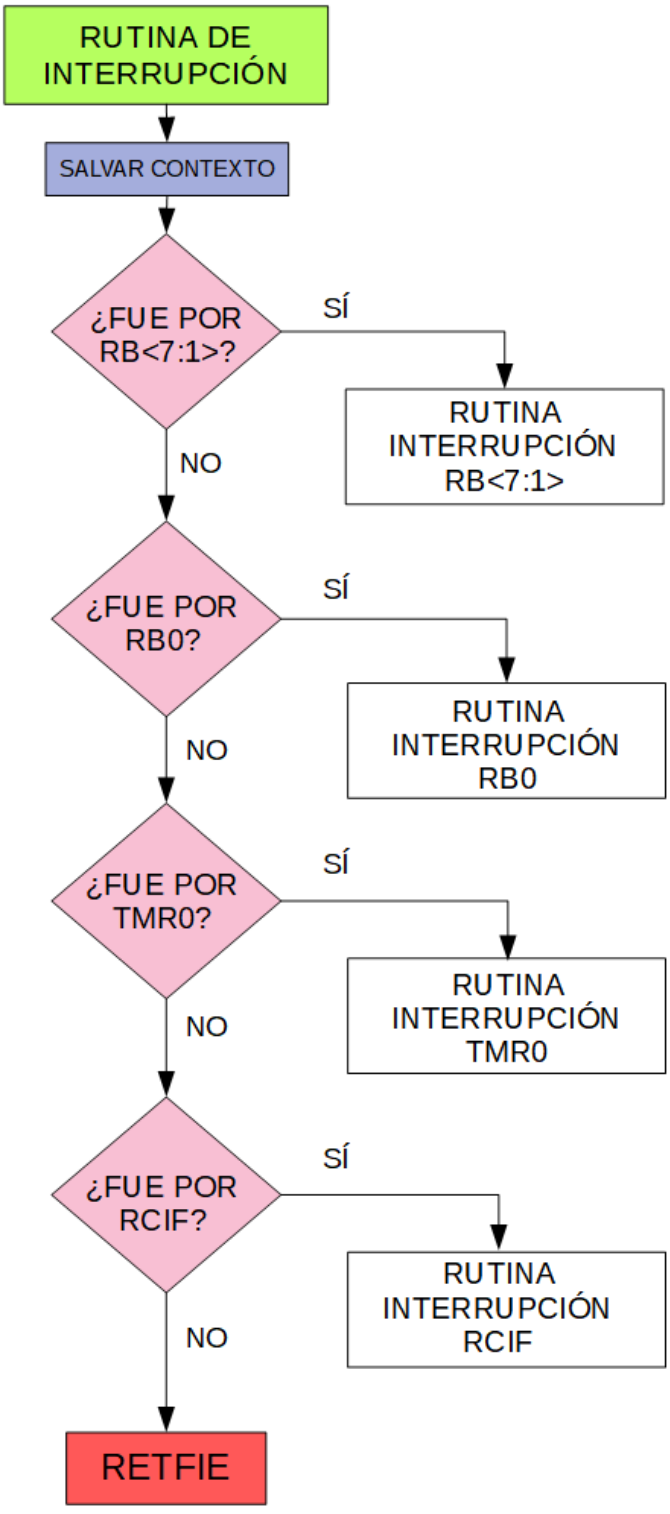


*Figura 2: Detección de origen de interrupción*

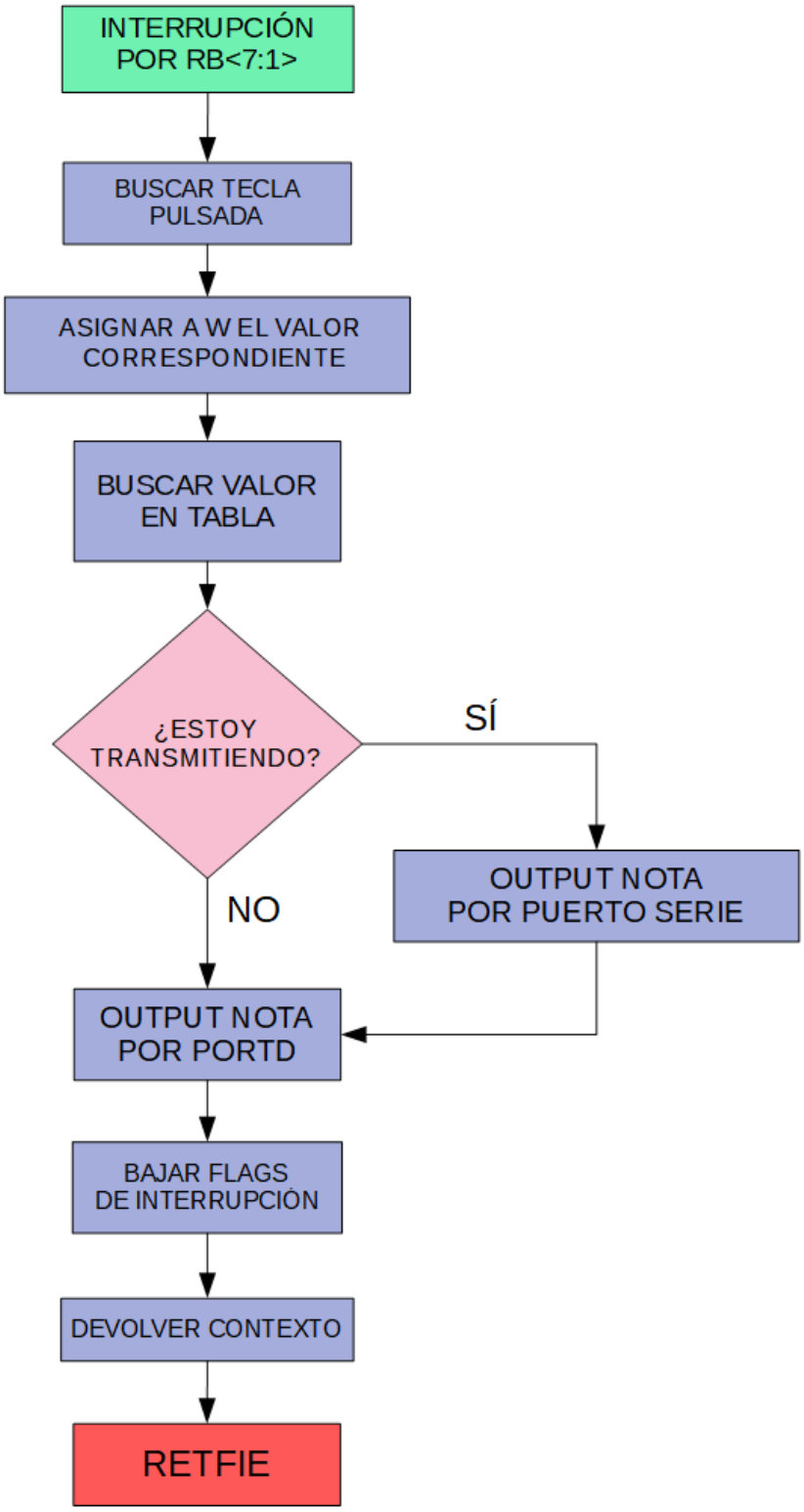


*Figura 3: Rutina de interrupción por RB<7:1>*

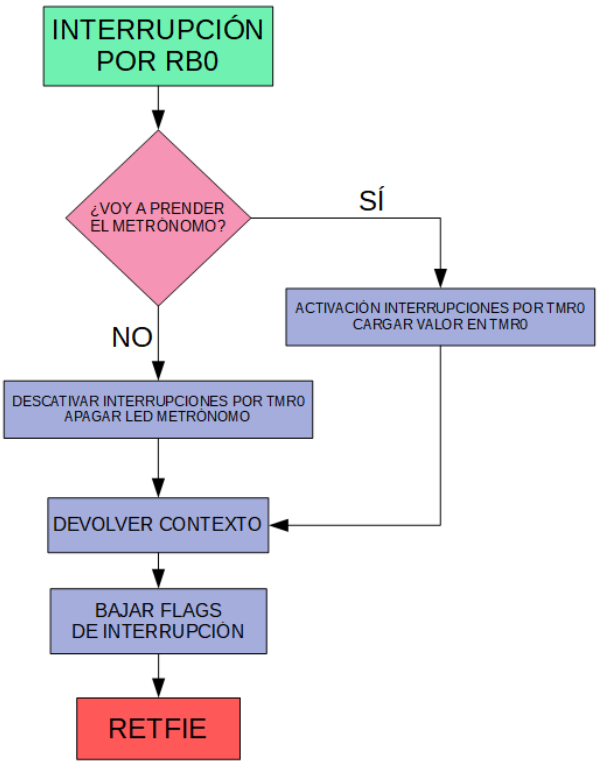


*Figura 4: Rutina de interrupción por RB0*

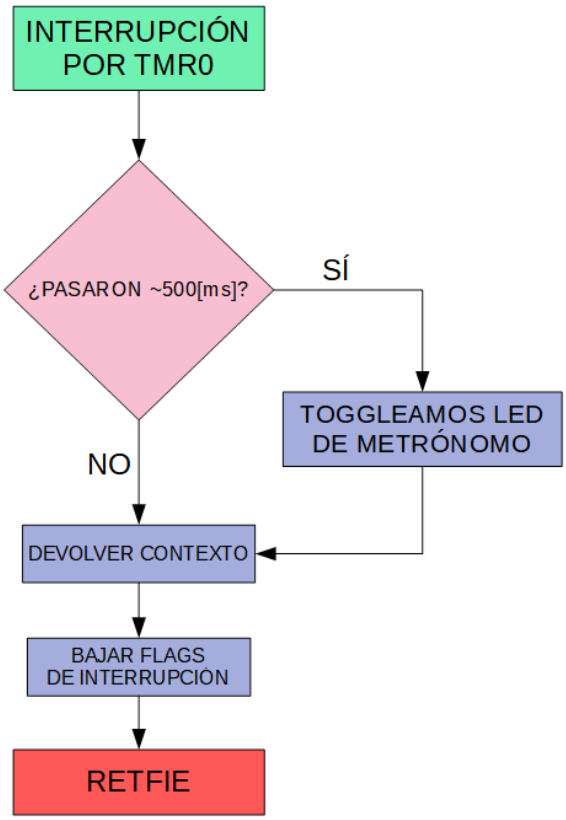


*Figura 5: Rutina de interrupción por TMR0*

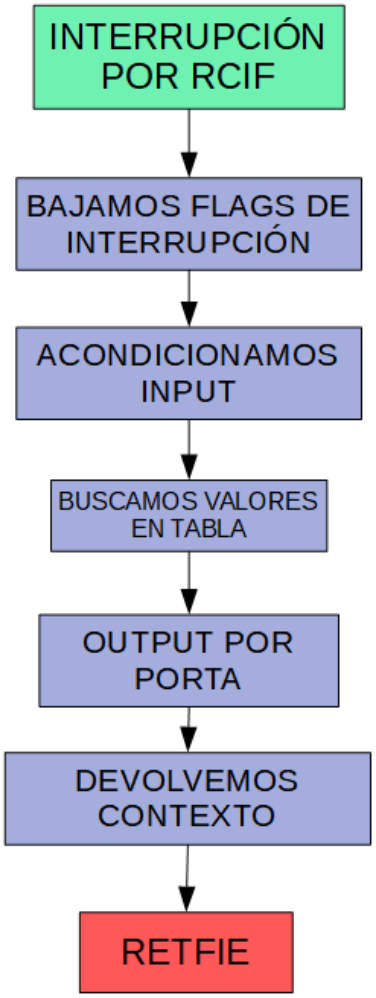


*Figura 6: Rutina de interrupción por RCIF*

Los diagramas mostrados fueron de mucha ayuda a la hora de programar. Gracias a la implementación de estos diagramas de flujo, el grupo se percató de algunas restricciones que debían ser implementadas y de otras que al implementarlas facilitaban la programación.

# Desarrollo práctico

En cuanto al desarrollo práctico del trabajo, el mismo se divide en 3 etapas correspondientes a las implementaciones que tiene el proyecto. Las mismas se detallan a continuación:

## Interrupción por RB<7:1>

En cuanto a la implementación de las interrupciones por los pines RB<7:1> del puerto PORTB, el grupo atravesó por varias etapas en el proceso de codificar el programa en *Assembly.* Comenzamos identificando el pin correspondiente a la tecla pulsada desarrollando un algoritmo que intentaba (una vez pulsada alguna tecla e ingresando un '0' lógico por algún pin de RB<7:1>), leer el puerto PORTB, almacenar dicha lectura e ir rotando el valor del puerto constantemente hasta encontrar en el bit de *"carry"* el '0' correspondiente a la tecla pulsada, incrementando una variable contadora en cada rotación; para luego poder buscar en una tabla (utilizando el valor de dicha variable) la nota musical correspondiente al botón pulsado. En un principio, el programa realizaba bien las instrucciones dadas haciendo uso del simulador de *MPLAB X IDE* junto con su herramienta *Stimulus*, pero no se obtuvo el mismo comportamiento en la simulación en *Proteus*. La detección fallaba y lo simulado no era lo deseado. Dado este inconveniente, se optó por evaluar los botones (pines del puerto PORTB) *"uno por uno"* hasta encontrar el que se correspondía con lo que el usuario pulsó. Dicha modificación implicó un código un poco más extenso, pero funcional al fin y al cabo.

# Interrupciones por RB0 y TMR0

Respecto a la interrupción por TMR0, al mismo se lo utiliza para realizar la función de *metrónomo.* Esta función se puede activar o desactivar mediante el accionar del pulsador en RB0, lo cual genera una interrupción (RBIF) y dentro de la misma se habilita o deshabilita esta funcionalidad.

Luego de probar varias codificaciones, el grupo se dio cuenta de que, utilizando la rutina de interrupción tanto para atender las interrupciones por RB<7:1> como las del TMR0, el programa funcionaba correctamente.

Cabe resaltar que el grupo desarrolló la rutina de interrupción por TMR0 de la manera más compacta y simple posible para evitar que el PIC quede demasiado tiempo en la misma. Esto se hizo con el fin de que si el usuario está haciendo uso del metrónomo y está tocando a la vez, entonces el PIC tarde lo menos posible en marcar el pulso y pueda volver a atender las interrupciones del usuario por RB<7:1>. Se corroboró mediante la herramienta *Stopwatch* de *MPLAB X IDE* que la rutina de interrupción por TMR0 tarda unos 34[µs], siendo esto básicamente imperceptible.

# Interrupción por recepción en puerto serie

El puerto serie fue implementado tanto para transmitir como para recibir datos (para las funcionalidades *"GRABAR"* e *"INGRESAR PARTITURA*" respectivamente). Empezando por la recepción, la misma se implementa en una rutina de interrupción; en el hardware del proyecto, se tiene un pulsador el cual corresponde a la función "*INGRESAR PARTITURA"*. Éste pulsador lo que hace internamente en el PIC es habilitar las interrupciones por recepción del puerto serie, y en la rutina de interrupción lo que se hace simplemente es leer el valor del registro RCREG y almacenarlo en una variable para luego ser mostrado en los displays correspondientes.

Siguiendo con la transmisión, la misma se realiza de manera similar a la recepción. Se tiene un pulsador que indica la funcionalidad *"GRABAR"*, el cual al pulsarse habilita la transmisión por puerto serie y, en base a eso, en el momento en el cual se pulsa una tecla de una nota en RB<7:1> (la que quiere ser grabada) se entra en la rutina de interrupción de RB<7:1> en donde se tiene en cuenta si está en *"on"* la función *"GRABAR"*. Si es así, el dato, además de mostrarse por el display, se coloca en el registro TXREG implementando de esa manera la transmisión de dicho dato. Luego, el programa desarrollado en Java recibe este dato y lo almacena en una cadena de caracteres que luego será guardada en un archivo *.txt*.

# Cálculos

## Cálculo de resistencias para LEDs

En este proyecto utilizamos LEDs de 4 colores distintos: rojo, amarillo, verde y azul. Como cada LED tiene una caída de tensión distinta, debemos hacer los cálculos pertinentes para asignarles una resistencia de un valor razonable.

La caída de tensión para LEDs de distintos colores se detalla en la tabla que sigue:

| **Color del LED** | **Caída de tensión [V]** |
|---|---|
| *Rojo* | *1.2* |
| *Amarillo* | *1.8* |
| *Verde* | *2.4* |
| *Azul* | *3.4* |

*Tabla 2: Rango de caída de tensión para LEDs de distintos colores*

Se procedió a desarrollar un cálculo sencillo mediante las leyes de Krichhoff para hallar el valor de las resistencias necesarias para cada caso contemplando el siguiente circuito:

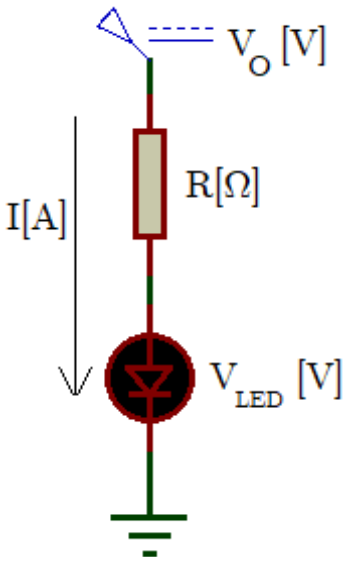


*Figura 7: Circuito para cálculo de resistencias*

Sabiendo que la tensión entregada por un pin de salida del PIC16F887 ($V_O$) es de entre 4.5[V] y 4.7[V], tomamos la media del rango, trabajando entonces con $V_O = 4.6$[V].

Además, la corriente I entregada a cada LED será tomada como 15[mA] para evitar quemarlos.

Para LED rojo

Se plantean y resuelven las siguiente ecuaciones aplicando las leyes de Kirchhoff y de Ohm:

$$(V_O - V_R - V_{LED})[V] = 0[V]$$

$$4.6[V] - IR_R[A\Omega] - 1.2[V] = 0[V]$$

$$4.6[V] - 15(10^{-3})R_R[A\Omega] - 1.2[V] = 0[V]$$

$$(4.6 - 1.2)[V] = 15(10^{-3})R_R[A\Omega]$$

$$3.4[V] = 15(10^{-3})R_R[A\Omega]$$

$$\frac{3.4[V]}{15(10^{-3})[A]} = R_R[\Omega]$$

$$\Rightarrow R_R \cong 227[\Omega]$$

Trabajando con valores comerciales, diremos entonces que: R = 270[Ω].

Para LED amarillo

$$(V_O - V_R - V_{LED})[V] = 0[V]$$

$$4.6[V] - IR_{Am}[A\Omega] - 1.8[V] = 0[V]$$

$$4.6[V] - 15(10^{-3})R_{Am}[A\Omega] - 1.8[V] = 0[V]$$

$$(4.6 - 1.8)[V] = 15(10^{-3})R[A\Omega]$$

$$2.8[V] = 15(10^{-3})R_{Am}[A\Omega]$$

$$\frac{2.8[V]}{15(10^{-3})[A]} = R_{Am}[\Omega]$$

$$\Rightarrow R_{Am} \cong 187[\Omega]$$

Trabajando con valores comerciales, diremos entonces que: $R_{Am} = 220[\Omega]$.

Para LED verde

$$(V_O - V_R - V_{LED})[V] = 0[V]$$

$$4.6[V] - IR_V[A\Omega] - 2.4[V] = 0[V]$$

$$4.6[V] - 15(10^{-3})R_V[A\Omega] - 2.4[V] = 0[V]$$

$$(4.6 - 2.4)[V] = 15(10^{-3})R_V[A\Omega]$$

$$2.2[V] = 15(10^{-3})R_V[A\Omega]$$

$$\frac{1.65[V]}{15(10^{-3})[A]} = R_V[\Omega]$$

$$\Rightarrow R_V = 110[\Omega]$$

Trabajando con valores comerciales, decimos entonces que: $R_V = 150[\Omega]$.

Para LED azul

$$(V_O - V_R - V_{LED})[V] = 0[V]$$

$$4.6[V] - IR_{Az}[A\Omega] - 3.4[V] = 0[V]$$

$$4.6[V] - 15(10^{-3})R_{Az}[A\Omega] - 3.4[V] = 0[V]$$

$$(4.6 - 3.4)[V] = 15(10^{-3})R_{Az}[A\Omega]$$

$$1.2[V] = 15(10^{-3})R_{Az}[A\Omega]$$

$$\frac{1.2[V]}{15(10^{-3})[A]} = R_{Az}[\Omega]$$

$$\Rightarrow R_{Az} = 80[\Omega]$$

Trabajando con valores comerciales, decimos entonces que: $R_{Az} = 100[\Omega]$.

# Cálculo de retardo para antirrebote

Como los pulsadores que activan/desactivan las funciones de *"GRABAR"* e *"INGRESAR PARTITURA"* no están conectados a pines que puedan ser manejados por interrupciones, optamos por implementar un retardo por software que cumple la función de *antirrebote*. Luego de varias pruebas, el valor que encontramos más adecuado para esta implementación fue de 300[ms].

Para calcularlo, acudimos al método de *loops* anidados. En particular, usaremos tres.

El tiempo que tardará el loop más interno, se calcula de la forma:

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

Donde $p$ es el valor asignado al loop, y $t_i$ es el tiempo de instrucción del PIC. Para nuestro caso, tenemos: $t_i = 1[\mu s]$. Entonces, si iniciamos con $p = 255$ tenemos:

$$T_1 = (254)10^{-6} + (2)10^{-6} + (254)(2)10^{-6}\ [s]$$

$$T_1 = (764)10^{-6}\ [s]$$

$$\therefore T_1 = 764[\mu s]$$

Si ahora anidamos otro loop por fuera, el nuevo tiempo de retardo se calcula de la forma:

$$T_2 = 2t_i m + mT_1 + t_i(m\text{-}1) + 2t_i + 2t_i(m\text{-}1)$$

Donde $m$ es el valor asignado al loop. Para este caso, podemos usar $m = 131$ y obtenemos:

$$T_2 = (262)10^{-6} + (131)(764)10^{-6} + (131)10^{-6} + (2)10^{-6} + (260)10^{-6}\ [s]$$

$$T_2 = (100739)10^{-6}\ [s]$$

$$\Rightarrow T_2 = 100739[\mu s]$$

$$\therefore T_2 = 100.739[ms]$$

De esta manera, si repetimos tres veces este anidamiento de aproximadamente 100[ms], logramos un retardo de unos 300[ms].

Podemos entonces confirmar que los valores que usaremos para estos loops anidados son 255, 131 y 3.

# Consideraciones

Con el propósito de reducir la dificultad de la implementación de hardware y de software para lograr mayor simplicidad, se decidió que se usarían displays de 7 segmentos de cátodo común, y que se trabajará con lógica positiva (los segmentos se encienden con un '1' y para la multiplexación elegimos el display con un '0'). De esta manera, evitamos el cálculo y uso de transistores.

# Esquema circuital

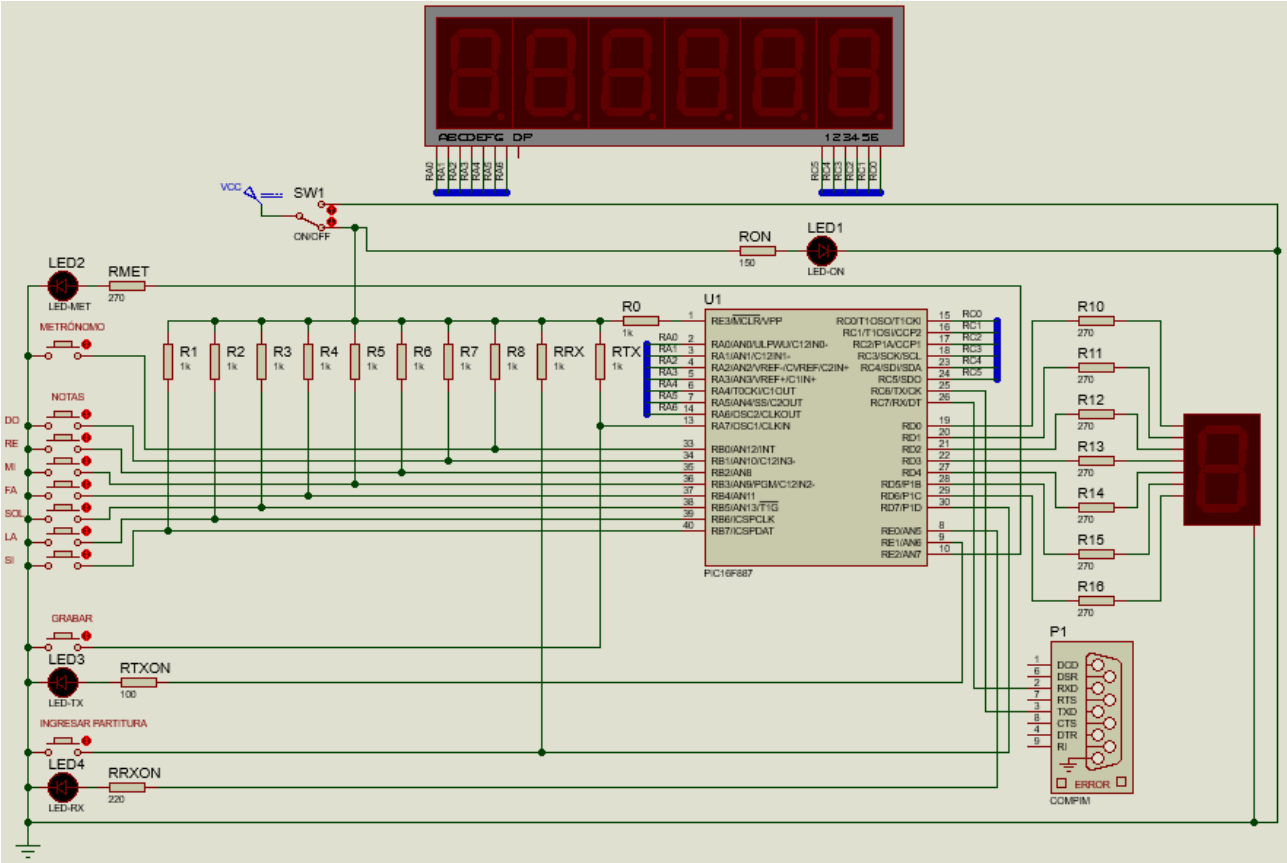


*Figura 8: Esquema circuital del proyecto*

## Descripción del hardware

Iniciando por el pin VPP del PIC, se observa que está conectado a un switch para alimentarlo con 5[V] a través de una resistencia de 1[KΩ], lo que permite encender y apagar el microcontrolador. El encendido del circuito se indica por medio de un LED verde con una resistencia de 150[Ω]. Siguiendo con la parte izquierda del circuito, se encuentran varios pulsadores los cuales (como sus *labels* indican) son los encargados de enviar ceros al PIC cuando son pulsados, para que el mismo realice determinadas funciones. Comenzando por el PORTB, se encuentran los botones correspondientes a las notas musicales del teclado *(DO, RE, MI, FA, SOL, LA, SI)* y, además, en RB0 está conectado el pulsador correspondiente a la función *"METRÓNOMO"*. Se puede observar que arriba del pulsador "*METRÓNOMO"* se encuentra un LED conectado desde RE2 a través de una resistencia de 270[Ω], el cual comienza a titilar cuando el pulsador del metrónomo se pulsa y se acciona esta funcionalidad. Todos estos pulsadores *"input"* se conectan a través de una resistencia de 1[KΩ] a $V_{CC}$ para lograr que cuando los botones no estén pulsados, constantemente le llegue un '1' lógico a cada pin del PORTB y, cuando alguno se pulsa, la corriente desde $V_{CC}$ va a GND por lo que el PIC recibe un '0' desde el pulsador correspondiente, indicando que se quiere realizar alguna función (ya sea tocar alguna nota o utilizar el metrónomo).

Continuando con los pulsadores que tienen como label *"GRABAR"* e *"INGRESAR PARTITURA"*, los mismos se conectan a RA7 y RD7 respectivamente a través de una resistencia de 100[Ω] y 220[Ω] (dependiendo del color del LED indicativo de cada funcionalidad), siendo también inputs del PIC. Los mismos son los encargados de enviar un '0' al microcontrolador cuando se quiera utilizar alguna de estas funciones y, cuando alguno se pulse, se encenderá el LED que tiene asociado como se ve en el esquema (LED3 y LED4). Estos pulsadores también están conectados a $V_{CC}$ por medio de una resistencia de 1[KΩ] con el mismo propósito de enviar un '1' cuando no se esté pulsando el botón.

Observamos también en el puerto PORTA (el cual está configurado como *"output"* en los pines RA<6:0>) que se conectan sus pines (salvo RA7) a un bus azul; estando éste conectado a los pines de datos de los 6 displays que están arriba, justamente para enviar los datos a los displays que se quieran mostrar cuando se active la funcionalidad de *"INGRESAR PARTITURA"*.

Pasando al puerto PORTC, se ve que en los pines RC<5:0> hay un bus azul que se conecta a los 6 pines para multiplexación de displays que se encuentran en la parte superior del circuito. El output de estos pines será lo que se encargue de multiplexar los datos que sean recibidos para mostrar en la *"partitura"*. Los pines restantes del puerto PORTC (RX y TX) se conectan al conector del puerto serie *COMPIM* para establecer la comunicación con la aplicación desarrollada en Java, de la cual hablaremos más adelante.

Si se observa el puerto PORTD, se ve que los pines RD<6:0> (pines de output) se conectan a un display de 7 segmentos de cátodo común. Cada uno de ellos, a través de una resistencia de 180[Ω]. Este display se enciende cuando se presiona algún pulsador del puerto PORTB correspondiente a una nota musical, mostrando en el display (en cifrado americano) qué nota se presionó.

Por último, con lo que respecta al puerto PORTE, se utilizan RE<2:0> como pines de output que se conectan a los LEDs correspondientes a *"INGRESAR PARTITURA"*, *"GRABAR"*, y *"METRÓNOMO"*, siendo éstos los encargados de encender dichos LEDs cuando alguna de estas funcionalidades esté activada.

A modo de detalle, se puede observar que todos los pines del PIC fueron utilizados para alguna función.

# Desarrollo de aplicación en Java

El desarrollo de la aplicación en Java presentó dificultades a la hora de encontrar una librería para trabajar con puertos serie que tenga la suficiente documentación y material en internet para su correcta implementación. Luego de prueba y error, descubrimos que la librería ***GiovynetDriver*** funciona muy bien y no es complicada de implementar a pesar de los relativamente pocos tutoriales de uso que encontramos en internet.

Si bien no se va a profundizar en el desarrollo del código en Java, presentaremos a continuación qué se puede hacer con dicho programa.

## Funcionalidades

Cuando se inicie la aplicación, el usuario deberá ingresar por consola (el programa no cuenta con interfaz gráfica) *el número* del puerto *COM* con el que se trabajará. Sólo se permitirán números del 1 al 9.

Una vez ingresado correctamente el número del puerto *COM*, el usuario tendrá frente a sí un menú con 3 opciones:

- Enviar partitura.
- Grabar melodía.
- Salir.

### Enviar partitura

Para el correcto uso de la primera opción, el usuario debe activar en la simulación en *Proteus* la funcionalidad de *"INGRESAR PARTITURA"*, para poder habilitar así la recepción por puerto serie en el PIC. En caso de no activar esta funcionalidad, los datos serán enviados por puerto serie pero no serán correctamente recibidos en la simulación.

Una vez se haya cumplido este requisito, el usuario debe ingresar una secuencia de seis letras que representen notas musicales *mayores* en cifrado americano (cualquier letra desde la A hasta la G). En caso de ingresar más (o menos) de 6, ingresar alguna letra que no corresponda a este rango, o ingresar algún carácter no permitido, el programa invalidará el input y pedirá un reingreso. Esto se repetirá hasta que el input sea válido.

Finalmente, cuando el input es válido, el programa se encargará de enviar la secuencia ingresada por puerto serie para que el receptor (el PIC) pueda hacer uso de ella.

Una vez la secuencia haya sido enviada y recibida correctamente, el programa preguntará al usuario si desea enviar otra partitura, o si desea volver al menú principal.

### Grabar melodía

Antes de poder ejecutar esta funcionalidad, es requisito que el usuario cree una carpeta en el disco *C* donde almacenará la grabación (o grabaciones) que haga con el programa.

Cuando seleccione la opción de *Grabar melodía*, el usuario deberá ingresar el nombre de la carpeta creada donde almacenará la grabación y el nombre con el que desea guardar la melodía grabada. Sólo se permitirá grabar melodías en formato *.txt*.

En caso de que el usuario haya cometido un error de tipeo en el ingreso del nombre de la carpeta o del nombre del archivo, podrá corregirlo antes de grabar.

Cuando todo esté listo, el programa notificará al usuario que está leyendo todo lo que sea enviado desde el PIC al puerto serie indicado.

Cuando se quiera dejar de grabar, se deberá pulsar el botón de *"GRABAR"* desde la simulación para desactivar esta función y notificarle al programa en Java que se debe almacenar todo lo recibido en un archivo de texto en la ruta especificada. Después de esto, el usuario será redireccionado automáticamente al menú principal.

### Salir

Cuando el usuario seleccione esta opción, toda comunicación serial entre el programa en Java y el PIC será deshabilitada y el programa se cerrará por completo.

## Restricciones

El programa cuenta con las siguiente restricciones:

- Una vez que se selecciona el número del puerto *COM*, el usuario no podrá cambiarlo hasta reiniciar el programa.
- Lo grabado sólo se podrá almacenar en formato *.txt*.
- No se podrá enviar al PIC una cantidad de notas distinta a 6.

# Notas complementarias

Resaltamos que, si bien el funcionamiento del proyecto es el deseado, cuando el usuario desea activar/desactivar la funcionalidad de *"INGRESAR MELODÍA"* o "*GRABAR"*, puede percatarse de que los displays de la *partitura* se apagan por un tiempo de aproximadamente 300[ms]. Este es el antirrebote explicado anteriormente. Si este retardo no se implementa, la lectura del botón es errónea y el PIC no se comporta como queremos.

Decidimos no trabajar este antirrebote con interrupciones por TMR0 porque si se activa/desactiva la función de *Grabar*, el usuario puede estar usando el metrónomo (el cual ya está haciendo uso del TMR0), y el grupo encontró complicado asignarle un valor distinto a TMR0 en medio de la ejecución del metrónomo, porque se mezclan las instrucciones y no obtuvimos un buen resultado en las pruebas. Por otro lado, si el usuario activa la función de *Ingresar partirura*, entonces se deshabilitan las interrupciones por TMR0 para desactivar el metrónomo. Si el usuario desea desactivar esta función, entonces las interrupciones por TMR0 se activan luego de mostrar los valores por los displays y de desactivar las interrupciones por recepción en puerto serie.

Todos estos casos hicieron llegar al grupo a la conclusión de que, si bien es algo que podría evitarse, este problema entre multiplexación y retardo por software no era de vital importancia para este trabajo donde lo más crítico es el correcto funcionamiento del PIC frente a interrupciones de distintas fuentes, por lo que se decidió no modificar el código y continuar con el desarrollo de las demás funcionalidades.

En cuanto al valor de este retardo (∼300[ms]), se hicieron varias pruebas cuyos resultados no fueron los deseados con valores menores al establecido.

Por otro lado, cuando en *Proteus* agregamos todas las funcionalidades del proyecto, resulta que la primera pulsación de una tecla correspondiente a una nota musical no es reconocida, sino que debe ser pulsada nuevamente para que el simulador comience a trabajar. No sucede esto con los pulsadores de las demás funcionalidades. Con la herramienta *Stimulus* de *MPLAB X IDE* corroboramos rigurosamente –y más de una vez– que si toggleamos el estado de un pin de RB<7:1> (los pines de las notas musicales), la interrupción es captada y procesada correctamente, por lo que descartamos un *bug* en el código implementado.

# Conclusión

Este trabajo práctico final fue de gran ayuda para la comprensión de la estructura interna de un microcontrolador, sus registros y sus respectivas funciones, el manejo de interrupciones por periféricos, la comunicación por puerto serie con otros dispositivos y la programación rigurosa en *Assembly* siendo consciente y cuidadoso a cada momento sobre qué se está haciendo con cada registro (es decir, saltos de banco, seteo/testeo de bits, operaciones entre registros, etcétera).

Si bien no fue implementado el conversor *ADC*, durante el semestre fue requisito estudiarlo y también nos fue evaluado. La razón por la que no lo implementamos aquí, fue porque todos los pines del PIC utilizado resultaron ocupados con alguna funcionalidad y ésto nos condicionó bastante.

A pesar de no haber podido desarrollar la funcionalidad musical (en vez de mostrar las notas en cifrado americano, hacer un output sonoro) por cuestiones de tiempo, creemos firmemente que lo estudiado fue comprendido y pudo ser aplicado de manera exitosa para nuestro proyecto, con la salvedad del conflicto entre la multiplexación y el antirrebote, y el problema de no reconocer en *Proteus* la primera pulsación de una nota. Cabe resaltar que algunas restricciones fueron necesarias y otras fueron impuestas por el grupo para simplificar el proyecto.

Podemos decir que el trabajo resultó ser un incremento significativo en nuestros conocimientos sobre la electrónica digital y, tras muchas pruebas y errores, logramos un buen entendimiento sobre las capacidades del microcontrolador con el que trabajamos.