Universidad Autónoma Metropolitana

Unidad Azcapotzalco

Proyecto De Integración en Ingeniería electrónica
Modalidad de proyecto tecnológico

Diseño y construcción de una caja de música digital

Presenta:

Yordi Edgardo Delgado Ortiz

Matricula: 210300200

Asesor: Ing. Nicolás Reyes Ayala

Trimestre: 18-I

Marzo de 2018

Declaratoria

Yo, Nicolás Reyes Ayala, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Nicolás Reyes Ayala

Yo, Yordi Edgardo Delgado Ortiz, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Yordi Edgardo Delgado Ortiz

Resumen

El presente trabajo describe el diseño y la implementación de un sistema embebido, cuyo propósito es el de sintetizar sonidos con un timbre agradable. Valiéndose para esto de la técnica de síntesis digital de sonidos conocida como síntesis wavetable o síntesis de tablas de onda. El dispositivo creado es capaz de generar una polifonía de hasta 9 notas y de reproducir melodías en formato MIDI almacenadas en una memoria Micro SD, además de esto dispone de una pequeña pantalla de cristal líquido para el despliegue de información, sobre la melodía que se quiere seleccionar , el tempo con el cual se está reproduciendo, etc. Además de esto dispone de botones touch para la manipulación del sistema.

Tabla de contenido

Tabla de Figuras

Introducción:

El desarrollo de la tecnología a lo largo de la evolución del hombre ha avanzado exponencialmente, sobre todo en la electrónica, de la cual el primer tipo en aparecer fue la electrónica analógica, desde que apareció esta rama de la electrónica se ha enfocado a facilitar la vida del hombre creando desde aparatos para uso cotidiano hasta complejos dispositivos militares, industriales, médicos, etc. Bajo esta rama de la electrónica nacieron los primeros intentos de reproducir sonidos de instrumentos musicales o de generar nuevos sonidos con componentes electrónicos. Fue entonces que aparecieron los primeros teclados y órganos electrónicos, así como una variedad de instrumentos electrónicos.

Actualmente la tecnología electrónica ha evolucionado más, siendo así que se dio el nacimiento de nuevos dispositivos electrónicos, dispositivos digitales y con ellos una nueva era llamada la era digital. Esta nueva rama de la electrónica trajo consigo la creación de poderosos procesadores de cómputo capaces de realizar millones de operaciones por segundo, además de esto existen diversos dispositivos de hardware altamente paralelizable como lo son los FPGA o dispositivos encargados de hacer tareas específicas como lo son los DSP en el caso del procesamiento digital de señales. Todo esto ha permitido la evolución de los sintetizadores de sonido, ya que el gran poder de cómputo de los dispositivos mencionados anteriormente permite la aplicación en tiempo real de complejos algoritmos para la síntesis digital de sonido.

Bajo esta premisa se realizó el diseño e implementación de un dispositivo digital el cual emula el sonido característico de una caja de música, aplicando para ello la síntesis de sonido por tablas de onda, el dispositivo es capaz de cargar distintas melodías desde una memoria SD.

La Caja de música:

La caja de música es un instrumento mecánico, en el que unos dientes de acero, afinados, son puestos en vibración por contacto con partes en movimiento impulsados por un mecanismo de relojería. En 1796, Antoine Faure creó la primera música conocida producida por dientes de acero percutidos por puntas sujetas a un disco o rodillo.

Originalmente un complemento de relojes evolucionaron a cilindros de latón con pernos de acero que percutían una línea de dientes afinados, los cuales dieron lugar al peine afinado, de acero y en una sola pieza, provisto de la mejora esencial de apagadores de acero.

Hacia 1825, la caja musical estaba bien establecida, con peines sonoros de hasta 250 dientes que cubrían un ámbito de 6 octavas. La producción surgió, sobre todo, de Suiza bajo los constructores Henriot y Paillard. Además de música sacra y popular, se reprodujeron con este instrumento arias y oberturas de las óperas más famosas [1].

Justificación:

La síntesis de sonido digital es un tema bastante interesante y complejo. Muy estudiado en el ámbito académico, por distintas disciplinas de la ingeniería. Como la ingeniería Electrónica, La Ingeniería Física y La ingeniería en Computación, ya que este tema permite la aplicación de diversos conceptos abordados en estas.

Actualmente los proyectos que se han encontrado, similares a la presente propuesta, tienen una polifonía, y memoria de almacenamiento para las melodías bastantes limitados. Por lo cual la presente propuesta, pretende extender estas limitaciones, permitiendo mayor almacenamiento y/o mayor número de polifonía. Así mismo se pretende que el dispositivo, sintetice un sonido de mejor calidad al de los trabajos ya existentes.

Objetivos:

- Objetivo general:

  Diseñar y construir un sistema embebido capaz de Sintetizar el sonido de una caja de música.

- Objetivos particulares:

  - Evaluar y Seleccionar el dispositivo a utilizar para la síntesis de acuerdo a los dispositivos disponibles, su capacidad de procesamiento, y su complejidad de desarrollo.

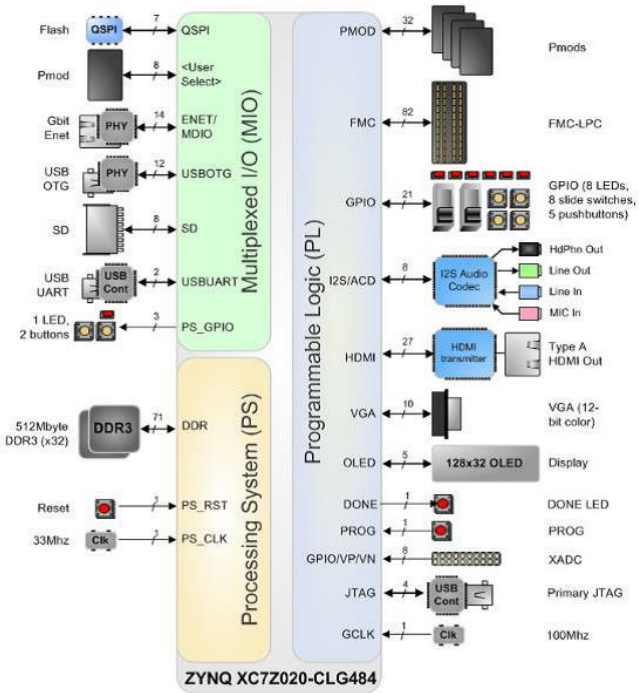
  - Implementar el esquema de síntesis wavetable (Tablas de onda) en el dispositivo elegido.

Evaluación del dispositivo a utilizar para la síntesis de sonido:

En esta parte del trabajo se hace un análisis de los dispositivos disponibles para realizar la síntesis digital de sonido, a continuación se enumera una serie de dispositivos para realizar esta síntesis:

1. Tarjeta de desarrollo zedboard
2. Tarjeta de desarrollo lauchpad tiva c
3. Tarjeta de desarrollo PSOC 5 Prototyping Kit

1.- [2] La tarjeta de desarrollo zedboard es una tarjeta que cuenta con un dispositivo ApSoc de la marca xillinx en específico el zynq7020 este dispositivo cuenta con un microprocesador ARM córtex A-9 de doble núcleo corriendo a la velocidad de 700mhz además de esto cuenta con un FPGA equivalente a la familia Artix 7 además de esto la tarjeta dispone de otros elementos como diversos periféricos y un códec de audio ADAU1761 de Analog Devices, una diagrama de la tarjeta en cuestión se muestra a continuación.



Cabe destacar que para trabajar con esta tarjeta se tiene que tener un dominio de algún lenguaje de descripción de hardware para trabajar con la parte de la lógica programable y de lenguaje c para trabajar con el microprocesador. La dificultad de desarrollo con esta tarjeta es relativamente alta sin embargo provee un gran poder de cómputo y de flexibilidad dado a que contiene lógica programable.

2.- [3]La tarjeta de desarrollo launch pad tiva c es una tarjeta de la compañía Texas Instruments la cual posee un microcontrolador TM4C123GH6PM el cual entre otras cosas posee un núcleo ARM córtex M4 a 80mhz, las características completas de esta tarjeta se muestran a continuación

- 256KB Flash, 32KB SRAM, 2KB EEPROM
- Dual 12-bit 2MSPS ADCs
- 8 UART, 6 I2C, 4 SPI
- 80MHz 32-bit ARM Cortex-M4-based microcontrollers CPU

El ambiente de trabajo de este microcontrolador es el code composer estudio IDE que provee el fabricante o algún otro IDE que permita la programación de este dispositivo como lo puede ser el KEIL o el IAR el lenguaje para su programación es el lenguaje c.

3.- [4]La tarjeta de desarrollo PSOC 5 Prototyping Kit contiene un dispositivo del fabricante de semiconductores Cypress el cual es llamado PSOC(sistema programable en un chip) el cual combina periféricos analógicos y lógica digital programable basada en CPLD con un microcontrolador ARM Córtex M3, algunas especificaciones de este dispositivo se muestran a continuación.

- 32-bit ARM Cortex-M3 CPU
- 32 Entradas de interrupción
- 24 canales de acceso directo a memoria
- Procesador para filtro digital de 24 bits de punto fijo.
- 24 universal digital blocks (UDB)
- 4 convertidores digital a analógico de 8 bits
- 4 amplificadores operacionales
- 4 bloques analógicos programables
- Etc.

Como se puede observar de los dispositivos anteriormente mencionados el que ofrece más versatilidad como poder de procesamiento es la tarjeta zedboard, sin embargo está bastante sobrada para nuestro propósito además de que el desarrollar en esta tarjeta es por demás complejo y requiriere un alto conocimiento del entorno de desarrollo y de diseño de hardware ya sea en vhdl o verilog por lo cual esta queda descartada.

De nuestras 2 tarjetas restantes ambas poseen un CPU de 32 bits que puede funcionar a la velocidad de 80mhz, sin embargo el núcleo de la tarjeta de Texas Instruments es un poco más robusto ya que es un Córtex M4 y ofrece un poco más de instrucciones por ejemplo instrucciones para DSP que bien podríamos utilizar en nuestro proyecto, sin embargo la tarjeta de cypress semiconductor ofrece la flexibilidad de contar con bloques programables digitales y analógicos además de contar con un hardware especializado para DSP por si lo requerimos. Entonces por lo anteriormente dicho se optó por seleccionar la tarjeta PSOC 5 Prototyping Kit como la plataforma de desarrollo para nuestro proyecto.

Desarrollo del sintetizador de sonido

La síntesis por tablas de onda o wavetable es un tipo de síntesis muy utilizada para generar sonidos de instrumentos ya existentes. Para este tipo de síntesis se toma una muestra del instrumento previamente grabada y se aísla un solo ciclo de la señal, estas muestras se reproducen para recrear la forma de onda que contiene, la frecuencia de la señal dependerá tanto del número de muestras de la tabla que contiene la señal como de la frecuencia de muestreo con la que estas muestras son reproducidas, la fórmula para obtener el periodo quedaría entonces de la siguiente forma.

$$P = \frac{N}{Fs}$$

Donde N es el número de muestras que contiene la tabla con nuestra forma de onda y Fs la frecuencia con la que se reproducen estas muestras. Tomemos de ejemplo una tabla con 1024 muestras y una frecuencia de muestreo de 44100hz.

$$P = \frac{1024}{44100} = 0.02321$$

$$f = \frac{1}{P} = 43.0848 \text{ Hz}$$

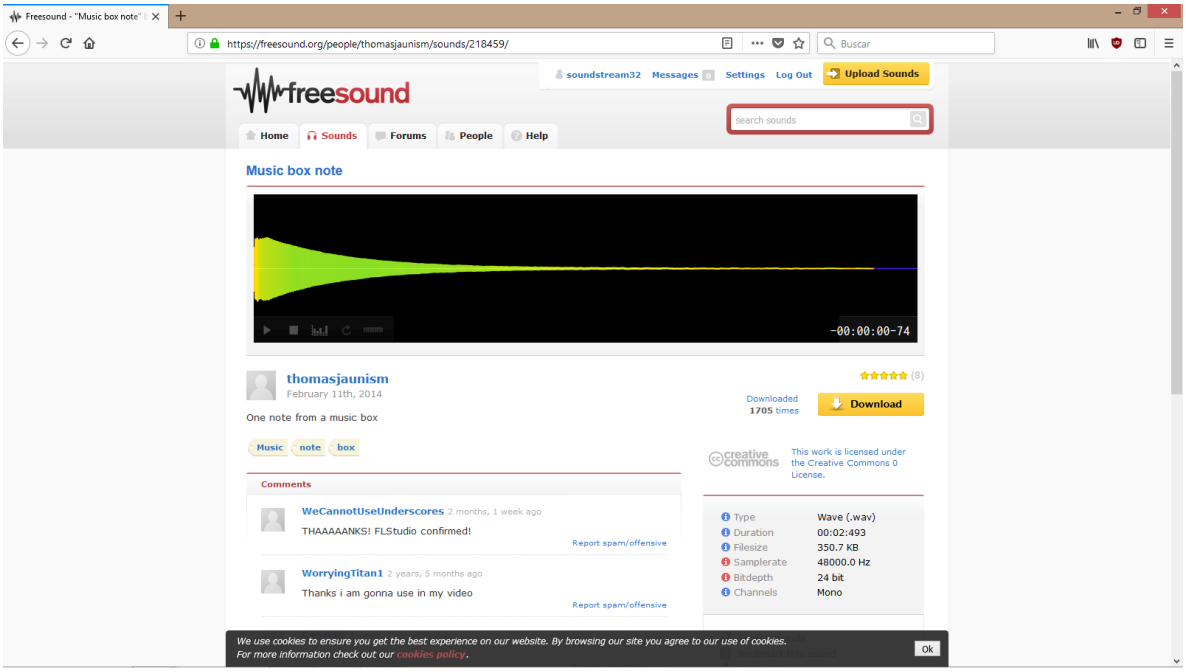
Como se puede observar obtendríamos una frecuencia de 43.0848 Hz con estos parámetros, si se desea obtener otra frecuencia habría que cambiar la frecuencia de muestreo o el tamaño de la tabla. Un primer enfoque podría ser cambiar la frecuencia de muestreo sin embargo en los sistemas digitales este parámetro es generalmente fijo, por lo cual solo resta cambiar el tamaño de la tabla. En primera instancia se podría tener una tabla para cada frecuencia deseada sin embargo esto sería tremendamente ineficiente y consumiría demasiada memoria, por lo que usualmente lo que se suele hacer es saltarse muestras. La fórmula para obtener el número de muestras que hay que saltarse para la frecuencia deseada es la siguiente.

$$S = N\frac{f}{Fs}$$

Donde S hace referencia a las muestras que hay que saltarse (por step en inglés) N a el número de muestras y Fs a la frecuencia de muestreo. Cabe destacar que al hacer esto algunos de los índices de la tabla no resultan en números enteros por lo que hay que redondear o aplicar algún tipo de interpolación, aunque esto último agrega más carga al procesador pero aun así es menos costoso que calcular la forma de onda[5].

Además de esto cabe mencionar que el sonido de un instrumento es dinámico es decir que la forma de onda cambia con el tiempo, por lo generar suele tener una sección de ataque donde el sonido es rico en armónicos y después una sección donde se estabiliza y se repite la misma forma de onda continuamente, por ende para generar un sonido más real los sintetizadores suelen tener varias tablas de onda que se reproducen en una sucesión para generar el sonido deseado.

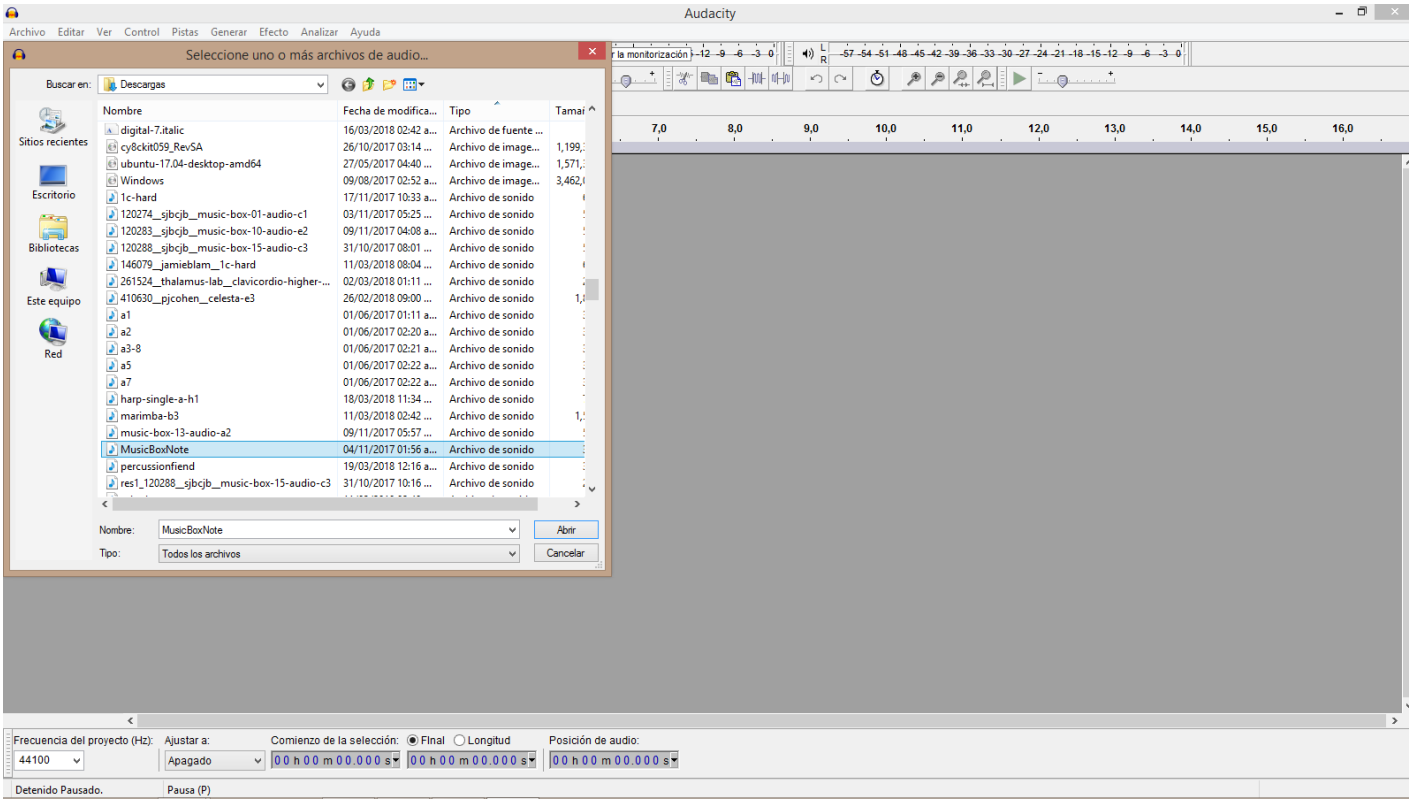
Una vez visto lo anterior procedemos entonces al desarrollo de nuestro sintetizador. Lo primero es obtener un ciclo de la forma de onda que deseamos generar, para esto descargamos una grabación de una de una caja de música, de una página de sonidos libres de autor



2.- descarga de la grabación de una nota de una caja de música

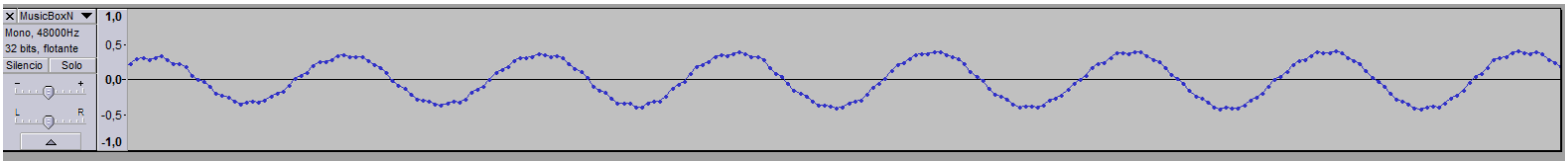
Una vez descargada la grabacion procedemos a su manipulacion para obtener un solo ciclo de nuestra forma de onda.Para esta tarea nos apoyamos en el programa de uso libre Audacity el cual nos permitira manipular nuestro audio.

Primero importamos nuestro audio.



3.- Importación de un archivo de audio en Audacity

A continuación podemos observar la forma de onda de la señal de nuestro archivo importado



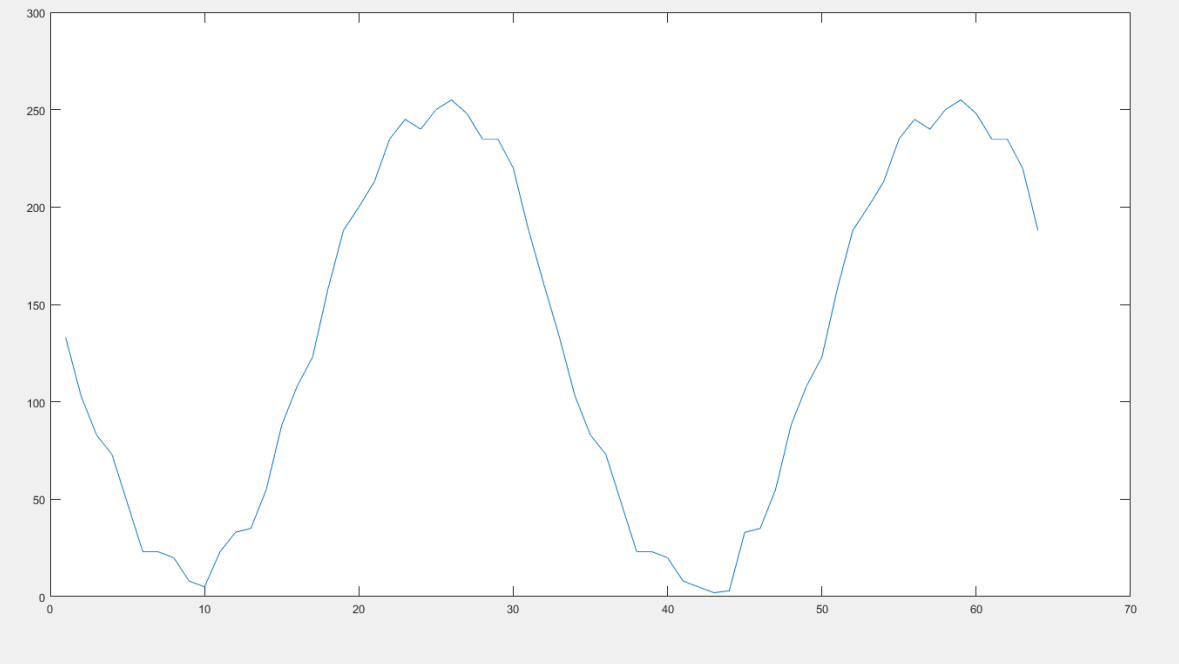
4.- Forma de onda del sonido de una caja de música

Ahora recortamos la señal para quedarnos con solamente con un ciclo

5.- Un ciclo de la forma de onda de una caja de música

Una vez hecho esto procedemos a exportar nuestro archivo de audio como PCM unsigned 8 bits esto debido a que la codificación PCM es una codificación que fácilmente puede ser interpretada por un DAC y 8 bits puesto que nuestro DAC es de dicha resolución.

Una vez exportado el archivo se importa en Matlab para recuperar estos datos, debido a que las muestras recuperadas eran muy pocas se decidió tomar 2 ciclos de nuestra señal para nuestra tabla de onda. A continuación se muestra la gráfica en Matlab



6.- Forma de onda de la señal graficada en Matlab

Como se puede ver tenemos una forma de onda con 64 muestras.

Ahora el próximo paso es obtener esta forma de onda a la frecuencia que deseamos en el dispositivo de desarrollo para esto inicialmente hay que crear nuestra tabla en memoria

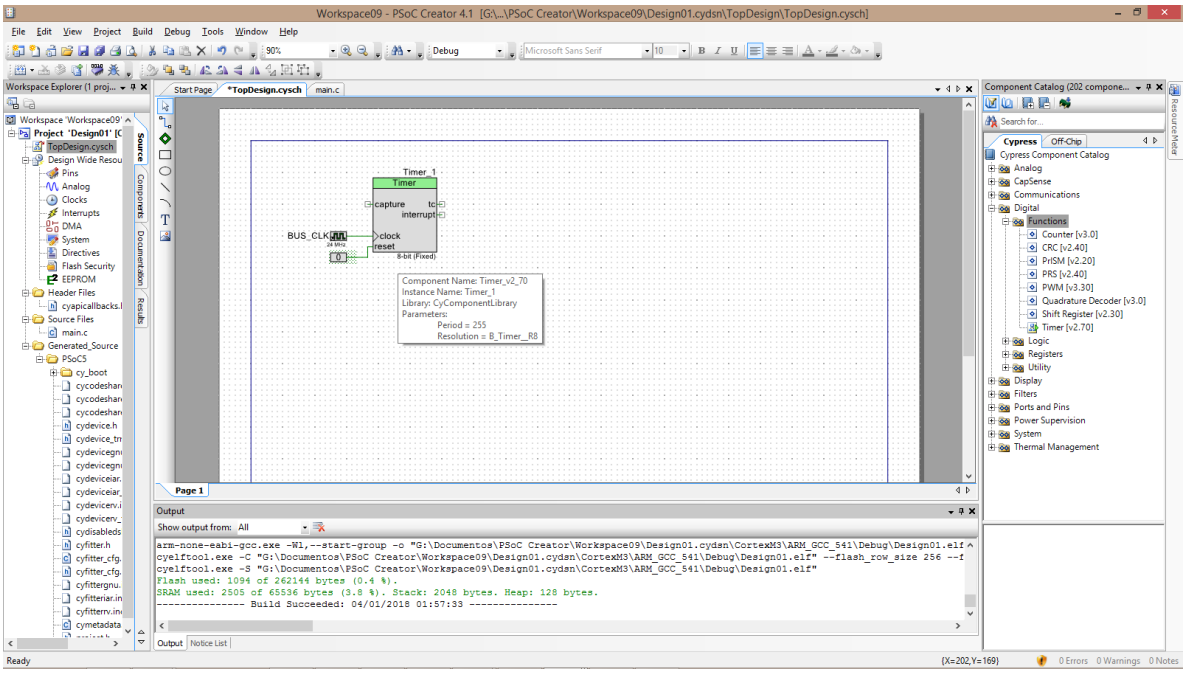
Para esto en nuestro entorno de desarrollo hacemos uso de la directiva CYCODE la cual le indica al compilador que guarde nuestros datos dentro de la memoria flash del dispositivo. Nuestra tabla entonces se declara de la siguiente forma:

```
CYCODE const uint8 sineTable[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,
2,3,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,2
20,188
};
```

Una vez hecho esto solamente queda de alguna manera leer nuestra tabla en el tiempo correcto para generar la forma de onda a la frecuencia deseada.

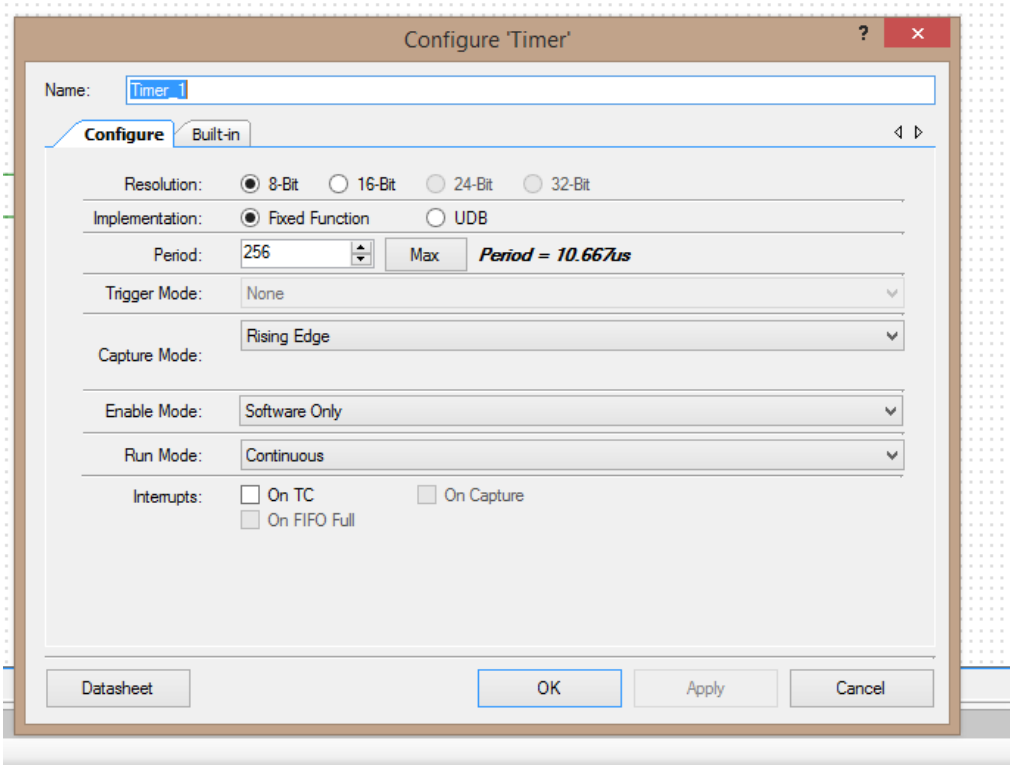
Como un primer enfoque probamos hacer esto mediante un timer que genere una interrupción la cual actualizara el índice de la tabla y enviara el dato al DAC para generar nuestra forma de onda. Procedemos entonces primero a la configuración de nuestro hardware:

Primero en el entorno de desarrollo de nuestra tarjeta el cual como se mencionó anteriormente es un entorno grafico bastante amigable, ubicamos un bloque timer y lo agregamos a nuestro diseño.



7.- Agregando un bloque timer a nuestro diseño en hardware

Como podemos observar en la imagen ya tenemos agregado un bloque de hardware en nuestro diseño el cual es un timer este bloque se sintetizara dentro del dispositivo haciendo uso de la lógica programable que contiene. Ahora solo resta configurar nuestro timer para esto hacemos doble clic en el bloque.



8.-Ventana de Configuración de nuestro bloque timer

Como se puede observar en la imagen anterior hay diversos parámetros que se pueden configurar en este bloque uno de ellos es la resolución de nuestro timer. Para obtener la resolución máxima de nuestro timer necesitamos hacer unos pequeños cálculos, esta resolución estará determinada por la frecuencia del reloj a la que esté trabajando nuestro timer así como el número de muestras de nuestra tabla y la frecuencia Mínima que queramos obtener en nuestra señal. Ahora bien como nuestro dispositivo trabajara con archivos MIDI la nota mínima que pueden contener dichos archivos es la nota DO en la octava -1, cuya frecuencia corresponde a 8.176 Hz como podemos observar en la siguiente tabla.

Octava MIDI	NOTA MIDI	Nota del Piano	Nota musical	Frecuencia (Hz)
-1	0	-	C	8.176
-1	1	-	C#/Db	8.662
-1	2	-	D	9.177
-1	3	-	D#/Eb	9.723
-1	4	-	E	10.301
-1	5	-	F	10.913
-1	6	-	F#/Gb	11.562
-1	7	-	G	12.250
-1	8	-	G#/Ab	12.978
-1	9	-	A	13.75
-1	10	-	A#/Bb	14.567
-1	11	-	B	15.434
0	12	-	C	16.352
0	13	-	C#/Db	17.323
0	14	-	D	18.354
0	15	-	D#/Eb	19.445
0	16	-	E	20.602
0	17	-	F	21.827
0	18	-	F#/Gb	23.125
0	19	-	G	24.5
0	20	-	G#/Ab	25.957
0	21	1	A	27.5
0	22	2	A#/Bb	29.135
0	23	3	B	30.868
1	24	4	C	32.703
1	25	5	C#/Db	34.647
1	26	6	D	36.708
1	27	7	D#/Eb	38.890
1	28	8	E	41.203
1	29	9	F	43.653
1	30	10	F#/Gb	46.249
1	31	11	G	48.999
1	32	12	G#/Ab	51.913

Figura 9.- Tabla con algunas frecuencias de las notas y su correspondiente numero en MIDI

Entonces con lo anterior mente dicho si se requiere una frecuencia de 8.176 Hz para nuestra señal el periodo que habría que poner a nuestro timer estaría determinado de la siguiente forma:

$$P_t = \frac{f_c}{N * f}$$

Dónde:

P_t Es el periodo de nuestro timer

f_c La frecuencia de reloj de nuestro timer

N el número de muestras de nuestra señal

f La frecuencia de nuestra señal

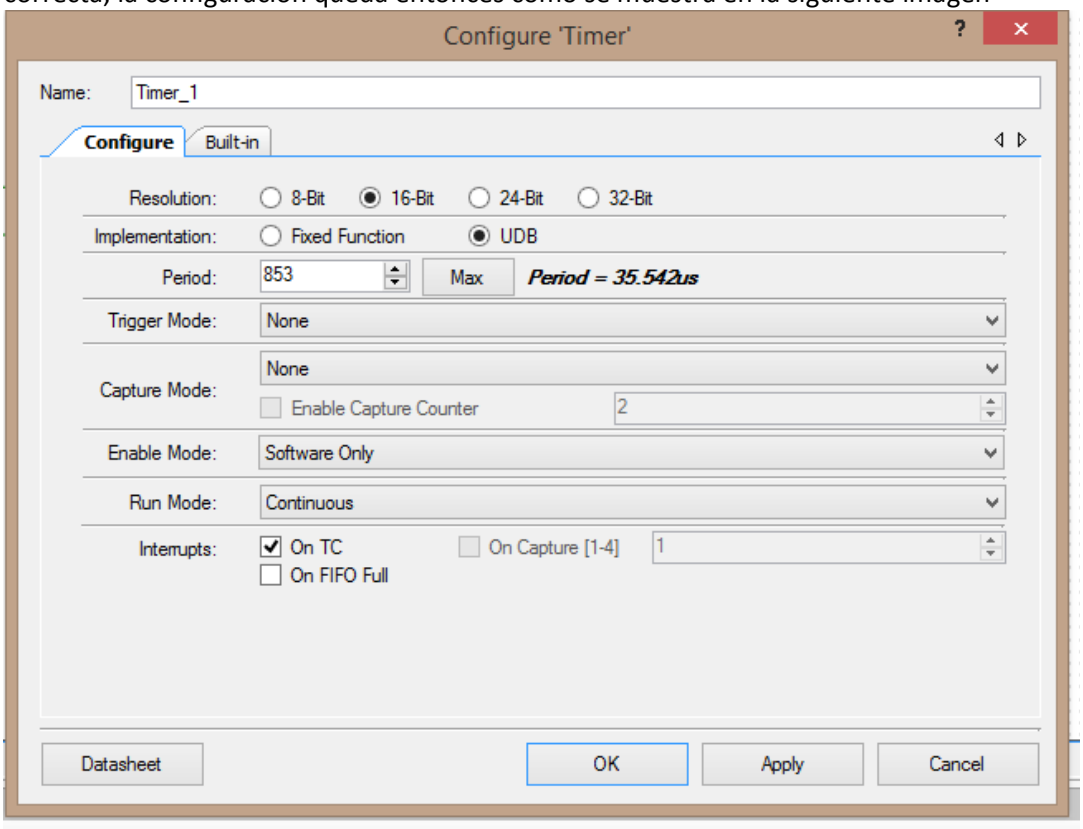
Entonces tomando en cuenta la formula anterior y sustituyendo los datos correspondientes tenemos que:

$$P_t = \frac{24000000}{64 * 8.176}$$

$$P_t = 45865.94$$

Entonces como podemos observar nuestra resolución tendrá que ser de 16 bits puesto que un timer de 8 bits no nos alcanza ya que el valor máximo que podemos escribir en este es 255.

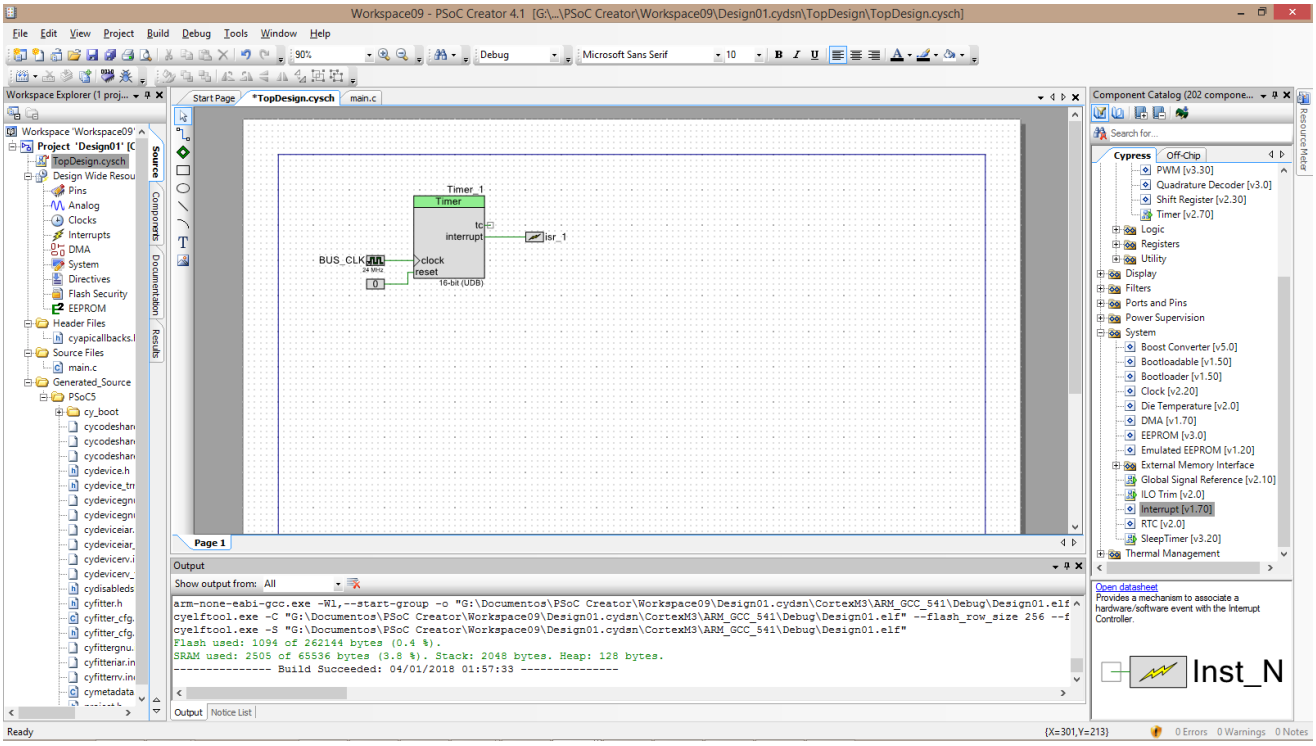
Ahora que conocemos la resolución de nuestro timer procedemos a configurarlo de manera correcta, la configuración queda entonces como se muestra en la siguiente imagen



10.- Ventana con la configuración inicial del timer

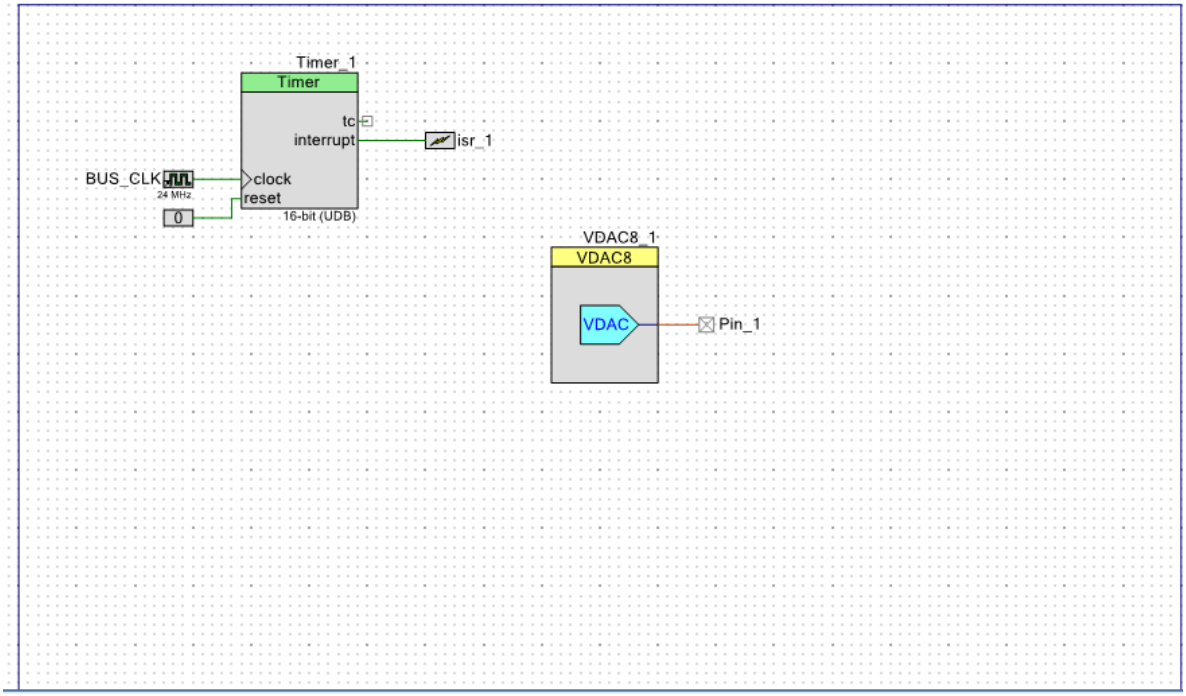
Como se puede observar la resolución es de 16 bits por lo ya anteriormente dicho, en el cuadro de implementación seleccionamos UDB ya que lo queremos implementar en los bloques digitales programables posteriormente hará uso también de los bloques fijos o dedicados a esta función. El siguiente parámetro que tenemos que tener en cuenta es el de interrupción ya que haremos uso de esta la seleccionamos en TC (terminal count) que provocara que la interrupción ocurra cuando el timer la cuenta que hemos fijado en el parámetro del periodo. Ahora bien ya que hemos agregado una interrupción hay que agregar esta misma en nuestro diagrama de hardware como se muestra a continuación.

Ubicamos el bloque Interrupt dentro de la pestaña System en nuestro entorno de desarrollo y lo arrastramos hacia nuestro diseño, posteriormente hacemos la conexión con la terminal Interrupt en nuestro timer, todo esto queda como en la siguiente imagen:



11.- bloque timer con su interrupción conectada

Una vez configurado hay que agregar un bloque DAC a nuestro diseño que es el que permitirá hacer la conversión de la representación digital que tenemos en nuestra tabla a una forma de onda analógica. Para esto ubicamos la pestaña Analog después la sub-pestaña DAC y podemos observar que tenemos 4 tipos de DAC de entre todos ellos seleccionamos el que dice VDAC y lo arrastramos a nuestro diseño, también hay que agregar un pin analógico al hardware, nuestro diseño queda entonces de la siguiente forma:



12.- Timer y VDAC ya configurados

Una vez realizado esto compilamos el diseño para que nuestro entorno de desarrollo genere las APIS de las que haremos uso para la manipulación de nuestro hardware. Una vez que se generaron las APIS y que el entorno de desarrollo conoce que hardware es el que estamos utilizando procedemos entonces a la programación necesaria para hacer uso de este.

Primero que nada hay que inicializar los bloques de hardware que añadimos a nuestro diseño. Para esto buscamos en el data sheet las APIS disponibles para esto. A continuación podemos ver una tabla con algunas de las APIS disponibles para la manipulación del timer

Functions

Function	Description
Timer_Start()	Sets the initVar variable, calls the Timer_Init() function, and then calls the Enable function.
Timer_Stop()	Disables the Timer.
Timer_SetInterruptMode()	Enables or disables the sources of the interrupt output.
Timer_ReadStatusRegister()	Returns the current state of the status register.
Timer_ReadControlRegister()	Returns the current state of the control register.
Timer_WriteControlRegister()	Sets the bit-field of the control register.
Timer_WriteCounter()	Writes a new value directly into the counter register. (UDB only)
Timer_ReadCounter()	Forces a capture, and then returns the capture value.
Timer_WritePeriod()	Writes the period register.
Timer_ReadPeriod()	Reads the period register.
Timer_ReadCapture()	Returns the contents of the capture register or the output of the FIFO.
Timer_SetCaptureMode()	Sets the hardware or software conditions under which a capture will occur.
Timer_SetCaptureCount()	Sets the number of capture events to count before capturing the counter register to the FIFO.
Timer_ReadCaptureCount()	Reports the current setting of the number of capture events.
Timer_SoftwareCapture()	Forces a capture of the counter to the capture FIFO
Timer_SetTriggerMode()	Sets the hardware or software conditions under which a trigger will occur.
Timer_EnableTrigger()	Enables the trigger mode of the timer.
Timer_DisableTrigger()	Disables the trigger mode of the timer.
Timer_SetInterruptCount()	Sets the number of captures to count before an interrupt is triggered.

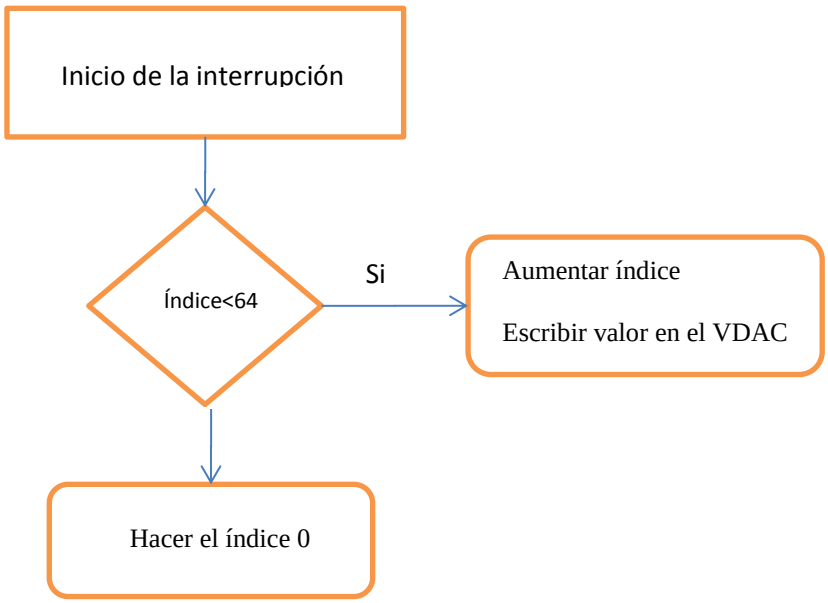
13.- Tabla con alguna APIS para hacer uso del timer

Como se puede ver en la tabla la Podemos hacer uso de la API start para inicializar nuestro timer, de manera similar existe la misma función para nuestro bloque DAC por lo que el código de inicialización queda de la siguiente forma:

```
#include "project.h"

int main(void)
{
    CyGlobalIntEnable; /* habilita interrupciones globales. */
    Timer_1_Start();
    VDAC8_1_Start();


    for(;;)
    {
            }
}
```

Con esto ya hemos inicializado nuestro timer y nuestro DAC, y están listos para funcionar, sin embargo nuestro dispositivo aun no hará nada ya que falta por programar la rutina de interrupción que es la que se encargara de actualizar el índice de nuestra tabla y de mandar este

valor a nuestro DAC para que este lo transformé en un voltaje en el pin de nuestro dispositivo, realizamos entonces un pequeño diagrama de flujo para tener una idea de lo que debemos de programar dentro de nuestra rutina de interrupción.



Nuestro código entonces quedaría de la siguiente forma

```
#include "project.h"
#include <math.h>
#define TABLE_LENGTH 64
int count=0;
CYCODE const uint8 sineTable[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,2,3
,33,
35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,220,188
};
CY_ISR(int_handler){
    if(count<64){
        VDAC8_1_SetValue(sineTable[count]);
        count++;
    }else{
        count=0;

    }

}
int main(void)
{
    CyGlobalIntEnable;     Timer_1_Start();
    VDAC8_1_Start();
    isr_1_StartEx(int_handler);
```

```
    for(;;)
    {

    }
}
```

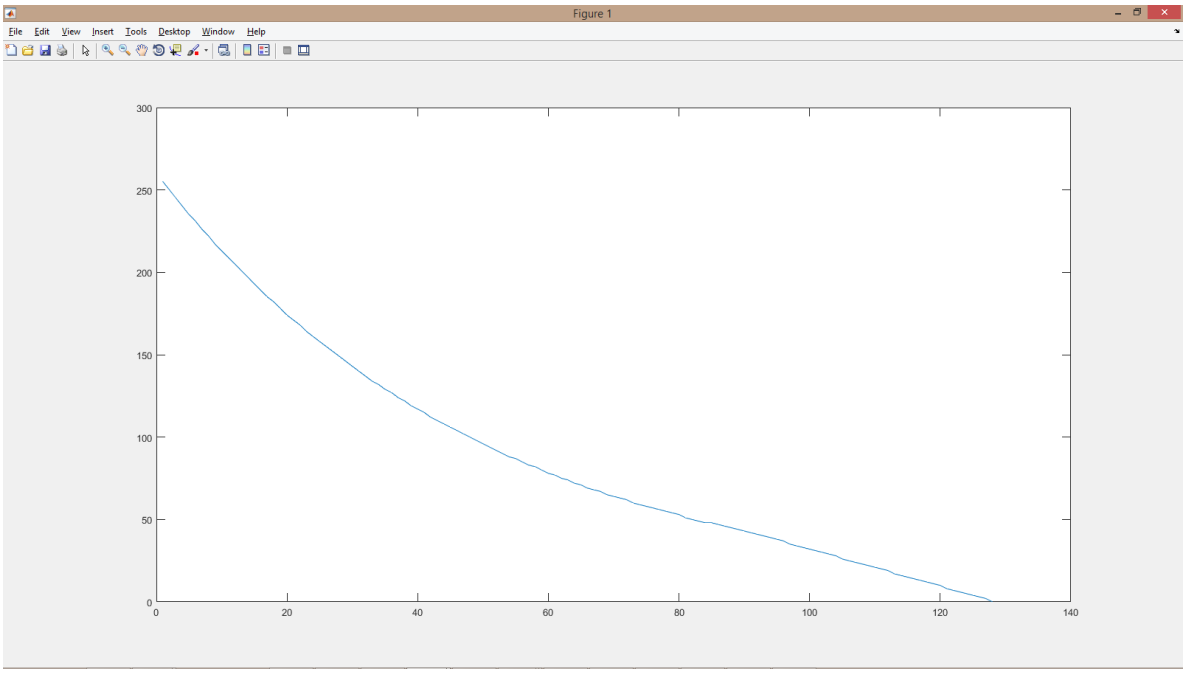
Con esto entonces obtuvimos una señal periódica, con la forma de onda almacenada en nuestra tabla y con una frecuencia de 440 Hz dado que la configuración inicial de nuestro timer está dada para dicha frecuencia. Con esto entonces tenemos ya una base para lo que será nuestro sintetizador.

Ahora si bien tenemos una señal periódica con la forma de onda deseada. Esto no es lo que se desea puesto que en un instrumento como la caja de música, la amplitud decae con el tiempo. Entonces es donde entra en juego lo que en la síntesis por tablas de onda se le conoce como envolvente. No entraremos mucho a detalle sobre esto dado que la envolvente en si puede ser bastante compleja y puede determinar en gran parte el contenido armónico dinámico de nuestro sonido, sin embargo para nuestros fines la envolvente basta con que sea una forma de onda exponencial que decae con el tiempo.

Ahora entonces generamos la tabla de la envolvente la cual queda de la siguiente forma

```
CYCODE const uint8 envelope[TABLE_LENGTH*2] = {

255, 250, 245, 240, 235, 231, 226, 222, 217, 213, 209, 205, 201, 197,
193, 189, 185, 182, 178, 174, 171, 168,
    164, 161, 158, 155, 152, 149, 146, 143, 140, 137, 134, 132, 129, 127,
124, 122, 119, 117, 115, 112, 110,
    108, 106, 104, 102, 100,  98,  96,  94,  92,  90,  88,  87,  85,  83,
82,  80,  78,  77,  75,  74,  72,
    71,  69,  68,  67,  65,  64,  63,  62,  60,  59,  58,  57,  56,  55,
54,  53,  51,  50,  49,  48,  48,
    47,  46,  45,  44,  43,  42,  41,  40,  39,  38,  37,  35,  34,  33,
32,  31,  30,  29,  28,  26,  25,
    24,  23,  22,  21,  20,  19,  17,  16,  15,  14,  13,  12,  11,  10,
8,   7,   6,   5,   4,   3,   2,   0

};
```

Y abajo podemos ver esta misma graficada en Matlab



14.- Grafica de la envolvente

Agregamos un bloque timer para actualizar los valores de la envolvente y procedemos a la programación de la rutina de interrupción de este, la rutina de interrupción de este queda entonces de la siguiente forma

```
CY_ISR(env_update){
env_index++;
Timer1_ReadStatusRegister();
}
```

Como se puede observar esta rutina de interrupción solo actualiza el índice de nuestra tabla de la envolvente.

Ya que tenemos esto solo queda multiplicar la tabla de la forma de onda de la señal por la envolvente, esto lo hacemos en la rutina de interrupción que genera nuestra señal y queda de la siguiente forma

```
CY_ISR(int_handler){

  if(env_index<=127){

     if(count<=TABLE_LENGTH){
        x=(sineTable[count]*envelope[env_index]);
        x=x/255;

        VDAC8_1_SetValue(x);


        count++;
    }else{
```

```
          count=0;
     }
    }else{
            VDAC8_1_SetValue(0);

         }

     Canal1_ReadStatusRegister();
}
```

Con esto entonces ya generamos una señal a una frecuencia determinada y cuya amplitud decae exponencialmente con el tiempo. Con esto ya tenemos la base principal de nuestro sintetizador.

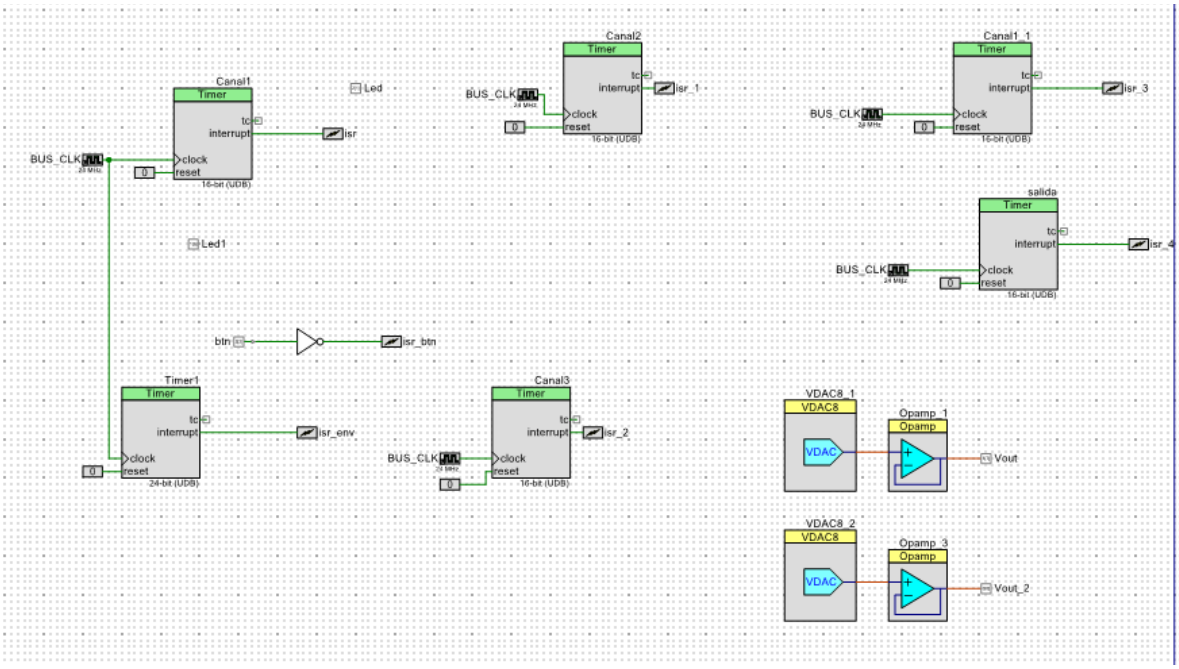
Ahora solo resta crear una función que nos permita controlar el inicio de la nota y su altura. Para ello generamos el siguiente bloque de código

```
 void tone(int canal,int frecuencia){
    int periodo;
      if(canal==1){

        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        env_index=0;
        count=0;
        Timer_1_WritePeriod(periodo);
        Timer_Start();
}
```

Como podemos observar lo que se hace en esta función es básicamente inicializar los parámetros del timer para que genere la nota a la frecuencia deseada. Ahora cada vez que se llama a la función tone el dispositivo toca la nota indicada y esta misma se apaga lentamente y no vuelve a sonar hasta que esta función es llamada nuevamente.

Ya contamos entonces con las herramientas necesarias para tocar una melodía monofónica sin embargo nuestro objetivo es lograr melodías con polifonía entonces para esto solo basta agregar más bloques timer y programar sus respectivas rutinas de interrupción.

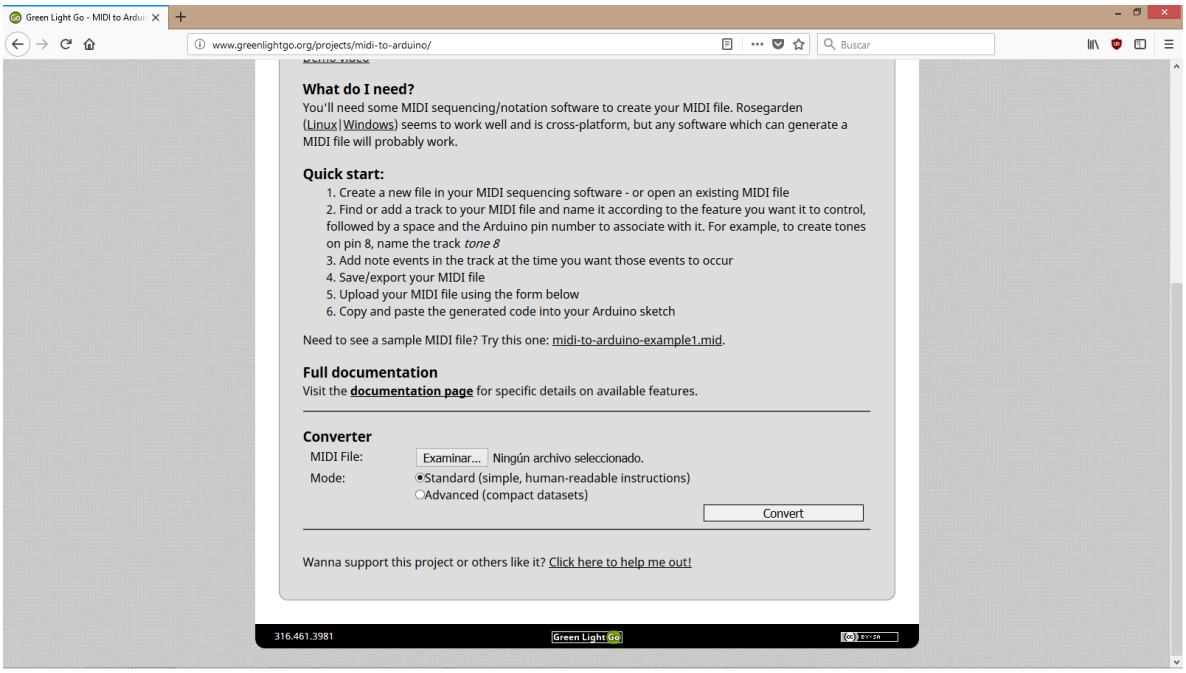
Para ello entonces configuramos el hardware y en nuestro IDE y el diagrama quedaría de la siguiente manera



15.- **Configuración del Hardware para polifonía de 3 notas**

Ahora entonces solo resta probar el dispositivo con alguna melodía esto en principio se podría hacer a mano mediante llamadas a la función tone y mediante delays sin embargo hacer esto para una melodía resulta demasiado tedioso por lo que investigando un poco encontramos una herramienta en internet la cual convierte archivos MIDI al formato anteriormente descrito. La dirección URL de dicha herramienta se muestra a continuación seguida de una captura de pantalla de dicha página.

http://www.greenlightgo.org/projects/midi-to-arduino/



16.- Herramienta utilizada para convertir archivos MIDI

Ahora que contamos con el código para la melodía proporcionado por esta herramienta se procedió a guardarlo en la memoria de programa y el código completo de esta etapa del proyecto donde el dispositivo puede tocar hasta 3 notas de polifonía se muestra a continuación.

```
#include "project.h"
#include <math.h>
#define TABLE_LENGTH 64
#define msPerTick 1
#define reloj 24000000

void tone(int canal, int  frecuencia);
void noTone(int canal);
void delay (uint32 ms);
int count=0,count2=0,count3=0,count4=0;
int env_index=0,env_index2=0,env_index3=0,env_index4=0;
int x;
int poly=0;
uint8 result=0;
CYCODE const uint8 sineTable[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,2,3
,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,220,
188

};
CYCODE const uint8 sineTable2[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,2,3
,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,220,
188
};
```

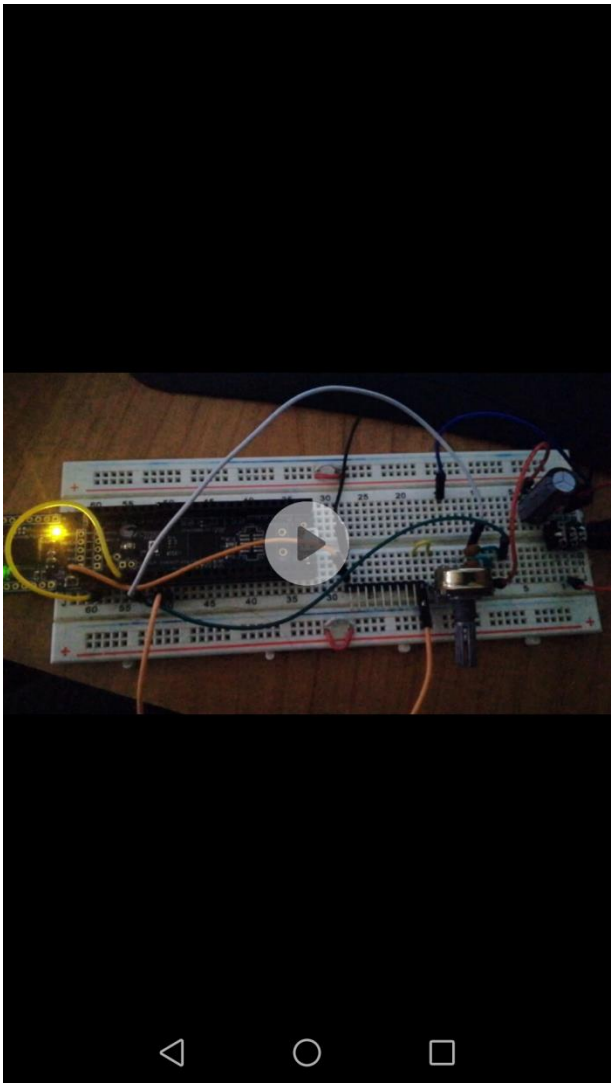
```
CYCODE const uint8 sineTable3[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,2,3
,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,220,
188
};

CYCODE const uint8 envelope[TABLE_LENGTH*2] = {

255, 250, 245, 240, 235, 231, 226, 222, 217, 213, 209, 205, 201, 197,
193, 189, 185, 182, 178, 174, 171, 168,
    164, 161, 158, 155, 152, 149, 146, 143, 140, 137, 134, 132, 129, 127,
124, 122, 119, 117, 115, 112, 110,
    108, 106, 104, 102, 100,  98,  96,  94,  92,  90,  88,  87,  85,  83,
82,  80,  78,  77,  75,  74,  72,
    71,  69,  68,  67,  65,  64,  63,  62,  60,  59,  58,  57,  56,  55,
54,  53,  51,  50,  49,  48,  48,
    47,  46,  45,  44,  43,  42,  41,  40,  39,  38,  37,  35,  34,  33,
32,  31,  30,  29,  28,  26,  25,
    24,  23,  22,  21,  20,  19,  17,  16,  15,  14,  13,  12,  11,  10,
8,   7,   6,   5,   4,   3,   2,   0

};

CY_ISR(int_handler){

  if(env_index<=127){

     if(count<=TABLE_LENGTH){
        x=(sineTable[count]*envelope[env_index]);
       result=x/255;


        VDAC8_2_SetValue(result);
        count++;
    }else{

         count=0;
    }
   }else{
           VDAC8_2_SetValue(0);

        }

    Canal1_ReadStatusRegister();
}
CY_ISR(int_handler2){

    if(env_index2<=127){

     if(count2<=TABLE_LENGTH){
         VDAC8_1_SetValue((sineTable[count2]*envelope[env_index2])/255);
        count2++;
    }else{

         count2=0;
    }
```

```
   }else{
       VDAC8_1_SetValue(0);

        }

   Canal2_ReadStatusRegister();
}
CY_ISR(int_handler3){

  if(env_index3<=127){

     if(count3<=TABLE_LENGTH){
        x=(sineTable[count3]*envelope[env_index3]);
       result=x/255;


        VDAC8_1_SetValue(result);
        count3++;
    }else{

         count3=0;
    }
   }else{
           VDAC8_1_SetValue(0);

        }

    Canal1_ReadStatusRegister();
}
CY_ISR(int_handler4){

  if(env_index4<=127){

     if(count4<=TABLE_LENGTH){
        x=(sineTable[count4]*envelope[env_index4]);
       result=x/255;


        VDAC8_2_SetValue(result);
        count4++;
    }else{

         count4=0;
    }
   }else{
           VDAC8_2_SetValue(0);

        }

    Canal1_1_ReadStatusRegister();
}
CY_ISR(btn_handler){

    env_index=0;
    env_index2=0;
    count=0;
    count2=0;
```

```
    Canal1_WritePeriod(2808);
    Canal2_WritePeriod(834);
    Canal1_Start();

    while(btn_Read()==0){
    }
    Canal1_Stop();
    Canal2_Stop();



}
CY_ISR(env_update){
env_index++;
env_index2++;
env_index3++;
env_index4++;
Timer1_ReadStatusRegister();
}
int main(void)
{
    Opamp_1_Start();
    Opamp_2_Start();
    Timer1_Start();
    VDAC8_1_Start();
    VDAC8_2_Start();
    Opamp_3_Start();
    Timer1_WritePeriod(200000);
    CyGlobalIntEnable;
    isr_StartEx(int_handler);
    isr_btn_StartEx(btn_handler);
    isr_env_StartEx(env_update);
    isr_1_StartEx(int_handler2);
    isr_2_StartEx(int_handler3);
    isr_3_StartEx(int_handler4);

    for(;;)
    {

        /* Inserte su melodía aqui */
    }
}
void tone(int canal,int frecuencia){
    int periodo;
    if (canal==1){
   if (poly>1)
    poly=0;
        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        switch(poly)
        {
        case 0:

        env_index=0;
        count=0;
        Canal1_WritePeriod(periodo);
        Canal1_Start();
        ++poly;
```

```
            break;
            case 1:
            env_index4=0;
            count4=0;
            Canal1_1_WritePeriod(periodo);
            Canal1_1_Start();
            ++poly;
            break;


            }
        }else if(canal==2){

            periodo=(reloj/frecuencia)/TABLE_LENGTH;
            env_index2=0;
            count2=0;
            Canal2_WritePeriod(periodo);
            Canal2_Start();
        }
        else if(canal==3){

            periodo=(reloj/frecuencia)/TABLE_LENGTH;
            env_index3=0;
            count3=0;
            Canal3_WritePeriod(periodo);
            Canal3_Start();

        }
    }
    void noTone(int canal){
        if(canal==1){

            switch(poly)
            {
            case 1:
            Canal1_Stop();
            poly--;
            break;
            case 2:
            Canal1_1_Stop();
            poly--;
            break;


            }


        }else if(canal==2){
            Canal2_Stop();
        }else if(canal==3){
            Canal3_Stop();
        }
      }
    void delay(uint32 ms){
        CyDelay(ms);
    }
```

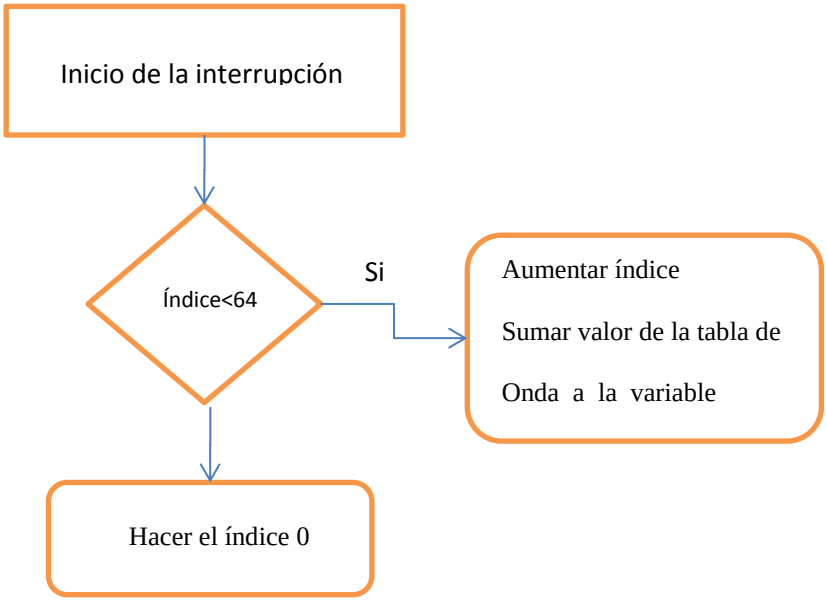
Cabe mencionar que en el código anterior se omitió la melodía dado que ocupa demasiadas líneas. Una foto del circuito de pruebas se muestra a continuación además si se da clic en ella te manda un enlace al video donde se puede ver el dispositivo en funcionamiento.



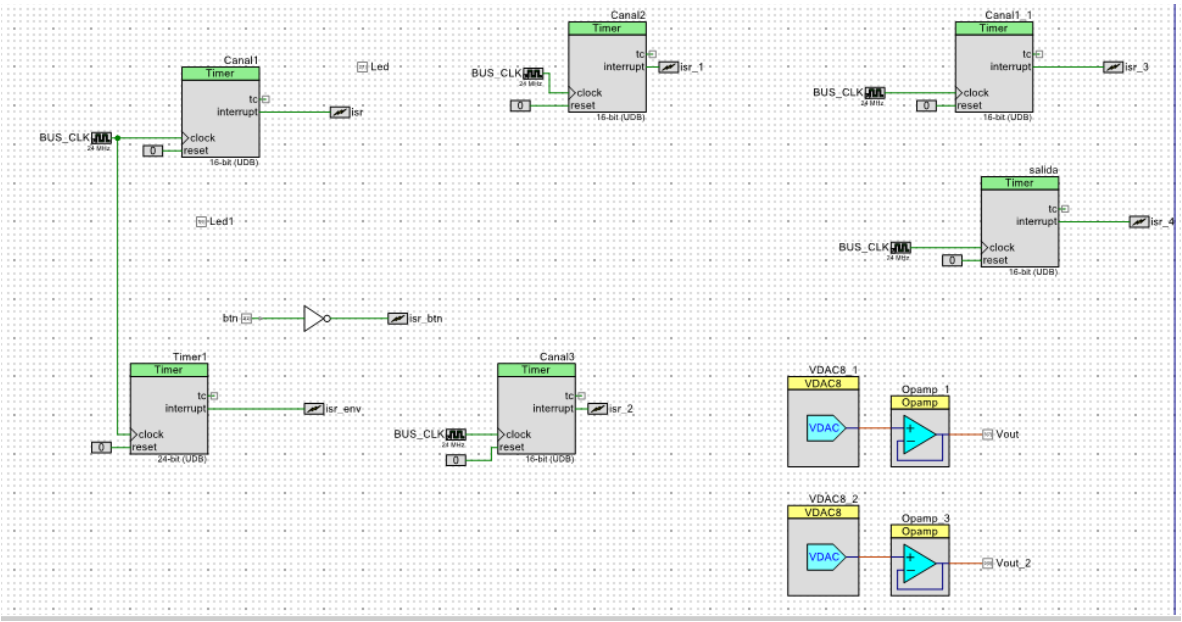
Mejorando los resultados obtenidos en la primera Etapa del sintetizador

Hasta ahora en esta etapa preliminar de nuestro proyecto tenemos entonces un dispositivo el cual puede tocar hasta 3 notas de polifonía a la vez, para agregar más polifonía bastaría entonces con agregar más bloques de hardware timers, y sus respectivas rutinas de interrupción. Sin embargo antes de continuar agregando polifonía debemos tratar con algunos inconvenientes que hemos detectado. Como ya hemos visto el enfoque utilizado para generar la polifonía es simplemente escribir el valor de la tabla de onda en el DAC en el momento en el que el índice de la tabla ha sido actualizado, si bien este enfoque nos da resultados correctos en cuanto a que el acorde generado es el deseado, este no es el enfoque que se emplea usualmente, y dado que nos hemos dado cuenta que con algunas melodías se generan algunos sonidos peculiares, se decidió usar el enfoque que se utiliza normalmente y comparar resultados.

Como normalmente se suele tratar la polifonía en este tipo de dispositivos es sumando las muestras de todos los canales y reproducirlas a una frecuencia determinada la cual es la frecuencia de muestreo que suele ser 32khz [5]. Para lograr esto en nuestro dispositivo entonces se decidió agregar un bloque timer y configurar su periodo para que genere una interrupción a 32khz esta interrupción entonces actualizara el valor del DAC con la suma de todas las notas. Esto entonces requiere modificar nuestra interrupción para cada uno de los canales y el diagrama de flujo anteriormente visto en la página 18 pasa a ser de la siguiente forma.



Agregamos entonces el nuevo bloque timer y nuestro hardware quedaría de la siguiente manera:



17.- Configuración de Hardware con Muestreo

Y el código entonces sería el siguiente

```
#include "project.h"
#include <math.h>
#define TABLE_LENGTH 64
#define msPerTick 0.5
#define reloj 24000000

void tone(int canal, int  frecuencia);
void noTone(int canal);
void delay (uint32 ms);
int count=0,count2=0,count3=0,count4=0;
int env_index=0,env_index2=0,env_index3=0,env_index4=0;
int x,x2,x3,x4;
int poly=0;
uint8 result=0,result2=0;
CYCODE const uint8 sineTable[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,2,3
,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,220,
188

};
CYCODE const uint8 sineTable2[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,2,3
,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,220,
188
};
CYCODE const uint8 sineTable3[TABLE_LENGTH] = {
```

```
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,2,3
,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,220,
188
};
CYCODE const uint8 envelope[TABLE_LENGTH*2] = {

255, 250, 245, 240, 235, 231, 226, 222, 217, 213, 209, 205, 201, 197,
193, 189, 185, 182, 178, 174, 171, 168,
    164, 161, 158, 155, 152, 149, 146, 143, 140, 137, 134, 132, 129, 127,
124, 122, 119, 117, 115, 112, 110,
    108, 106, 104, 102, 100,  98,  96,  94,  92,  90,  88,  87,  85,  83,
82,  80,  78,  77,  75,  74,  72,
    71,  69,  68,  67,  65,  64,  63,  62,  60,  59,  58,  57,  56,  55,
54,  53,  51,  50,  49,  48,  48,
    47,  46,  45,  44,  43,  42,  41,  40,  39,  38,  37,  35,  34,  33,
32,  31,  30,  29,  28,  26,  25,
    24,  23,  22,  21,  20,  19,  17,  16,  15,  14,  13,  12,  11,  10,
8,   7,   6,   5,   4,   3,   2,   0

};

CY_ISR(int_handler){

  if(env_index<=127){

     if(count<=TABLE_LENGTH){
        x=(sineTable[count]*envelope[env_index]);
        x=x/255;


        count++;
    }else{

         count=0;
    }
   }else{
           VDAC8_2_SetValue(0);

        }

    Canal1_ReadStatusRegister();
}
CY_ISR(int_handler2){

    if(env_index2<=127){

     if(count2<=TABLE_LENGTH){
         x3=((sineTable[count2]*envelope[env_index2])/255);
        count2++;
    }else{

         count2=0;
    }
   }else{
       VDAC8_1_SetValue(0);
```

```
        }

   Canal2_ReadStatusRegister();
}
CY_ISR(int_handler3){

  if(env_index3<=127){

     if(count3<=TABLE_LENGTH){
        x3=(sineTable[count3]*envelope[env_index3]);
       x3=x3/255;


        VDAC8_1_SetValue(result);
        count3++;
    }else{

         count3=0;
    }
   }else{
           VDAC8_1_SetValue(0);

        }

    Canal1_ReadStatusRegister();
}
CY_ISR(int_handler4){

  if(env_index4<=127){

     if(count4<=TABLE_LENGTH){
        x2=(sineTable[count4]*envelope[env_index4]);
       x2=x2/255;



        count4++;
    }else{

         count4=0;
    }
   }else{
           VDAC8_2_SetValue(0);

        }

    Canal1_1_ReadStatusRegister();
}
CY_ISR(btn_handler){

    env_index=0;
    env_index2=0;
    count=0;
    count2=0;
    Canal1_WritePeriod(2808);
    Canal2_WritePeriod(834);
    Canal1_Start();
```

```
    while(btn_Read()==0){
    }
    Canal1_Stop();
    Canal2_Stop();



}
CY_ISR(env_update){
env_index++;
env_index2++;
env_index3++;
env_index4++;
Timer1_ReadStatusRegister();
}
CY_ISR(DAC){
    result=((x+x2)/2);
    result2=(x3+x4)/2;
    VDAC8_2_SetValue(result);
    VDAC8_1_SetValue(result2);
    salida_ReadStatusRegister();
}
int main(void)
{
    Opamp_1_Start();
    Opamp_2_Start();
    Timer1_Start();
    VDAC8_1_Start();
    VDAC8_2_Start();
    Opamp_3_Start();
    Timer1_WritePeriod(200000);
    salida_Start();
    CyGlobalIntEnable; /* Habilita interrupciones globales. */

    /* codigo de inicializacion de hardware */

    isr_StartEx(int_handler);
    isr_btn_StartEx(btn_handler);
    isr_env_StartEx(env_update);
    isr_1_StartEx(int_handler2);
    isr_2_StartEx(int_handler3);
    isr_3_StartEx(int_handler4);
    isr_4_StartEx(DAC);

    for(;;)
    {
        /* aqui va el codigo de la melodia. */
    }
}
void tone(int canal,int frecuencia){
    int periodo;
    if (canal==1){
   if (poly>1)
    poly=0;
        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        switch(poly)
```

```
        {
        case 0:

        env_index=0;
        count=0;
        Canal1_WritePeriod(periodo);
        Canal1_Start();
        ++poly;
        break;
        case 1:
        env_index4=0;
        count4=0;
        Canal1_1_WritePeriod(periodo);
        Canal1_1_Start();
        ++poly;
        break;


        }
    }else if(canal==2){

        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        env_index2=0;
        count2=0;
        Canal2_WritePeriod(periodo);
        Canal2_Start();
    }
    else if(canal==3){

        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        env_index3=0;
        count3=0;
        Canal3_WritePeriod(periodo);
        Canal3_Start();

    }
}
void noTone(int canal){
    if(canal==1){

        switch(poly)
        {
        case 1:
        Canal1_Stop();
        poly--;
        break;
        case 2:
        Canal1_1_Stop();
        poly--;
        break;


        }


    }else if(canal==2){
        Canal2_Stop();
```

```
    }else if(canal==3){
        Canal3_Stop();
    }
  }
void delay(uint32 ms){
    CyDelay(ms);
}
```

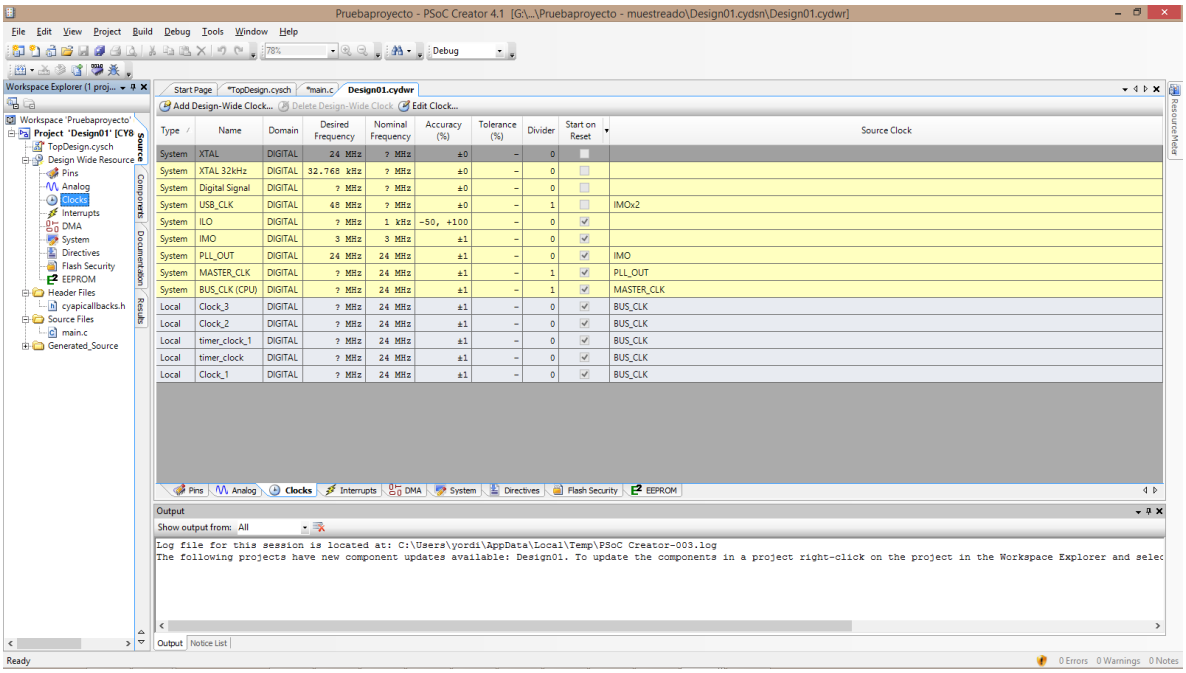
De nuevo hay que mencionar que la melodía de prueba se omitió en el código por razones de espacio.

Ahora entonces escuchando cada uno de los dispositivos y comparando los resultados de la etapa anterior con los de esta etapa, se determinó que aunque el dispositivo de la primera etapa genera un sonido muy rico en armónicos el cual por ende es muy agradable. Esto también podría estar produciendo un efecto de aliasing y generando nuevas componentes espectrales las cuales dan como resultado esos sonidos peculiares que se mencionaron anteriormente, esto último es simplemente una especulación hecha en base a los resultados.

En cuanto al segundo dispositivo si bien genera un sonido mucho menos rico en armónicos, su sonido es más limpio y elimina en gran parte los sonidos peculiares y no deseados del primer dispositivo, además entre más se incrementa la frecuencia de muestreo más limpio es este sonido. Sin embargo el agregar este muestreo, agrego más carga al procesador y provoca que el tempo de la melodía sea alterado ya que las operaciones no alcanzan a ser ejecutadas en el tiempo que se requiere. Para ello entonces hay que encontrar una manera de solucionar este problema.

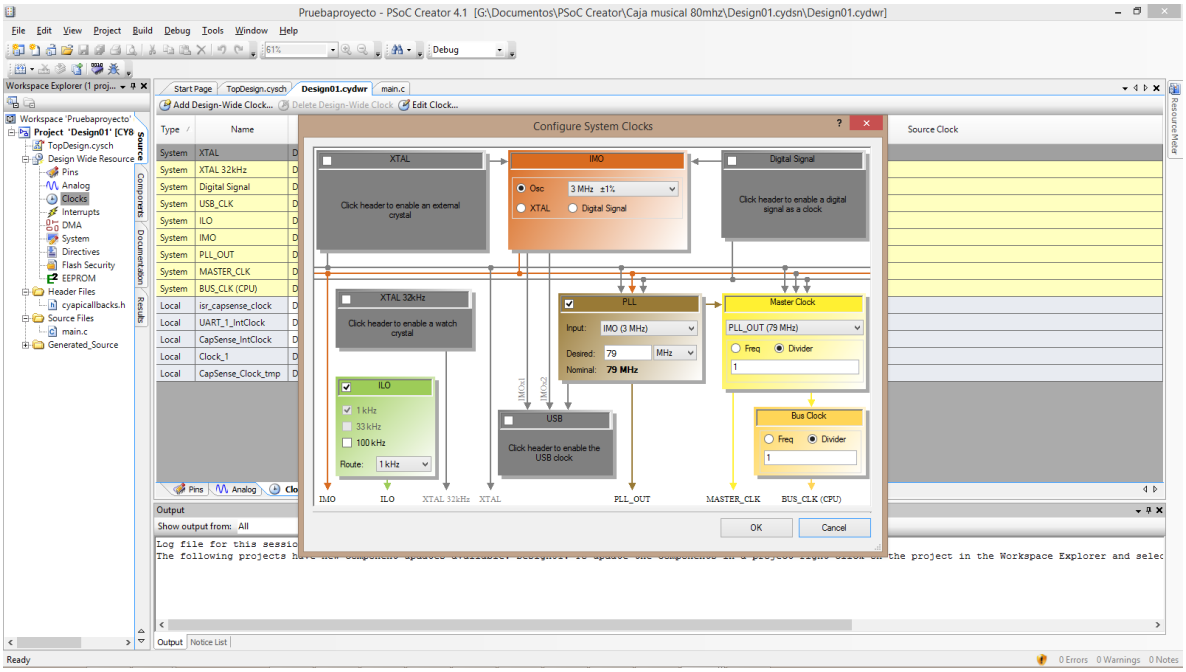
Leyendo la guía de usuario y el datasheet de nuestra tarjeta de desarrollo se encontró que por defecto el procesador está configurado para trabajar a una frecuencia de muestreo de 24mhz, por lo cual este no está trabajando a su máxima capacidad la cual es 80mhz, para ello entonces se procedió a configurar el dispositivo para que trabaje a la frecuencia de 80mhz.

Para esto hay que ubicar en nuestro entorno de desarrollo del lado izquierdo la pestaña que dice clocks y hacer doble clic sobre esta y aparecerá entonces una ventana como se muestra en la siguiente imagen.



18.- Configuración de los relojes del sistema

Una vez en esta ventana hacemos clic en la pestaña de Edit Clock y nos aparecerá algo similar a la siguiente imagen



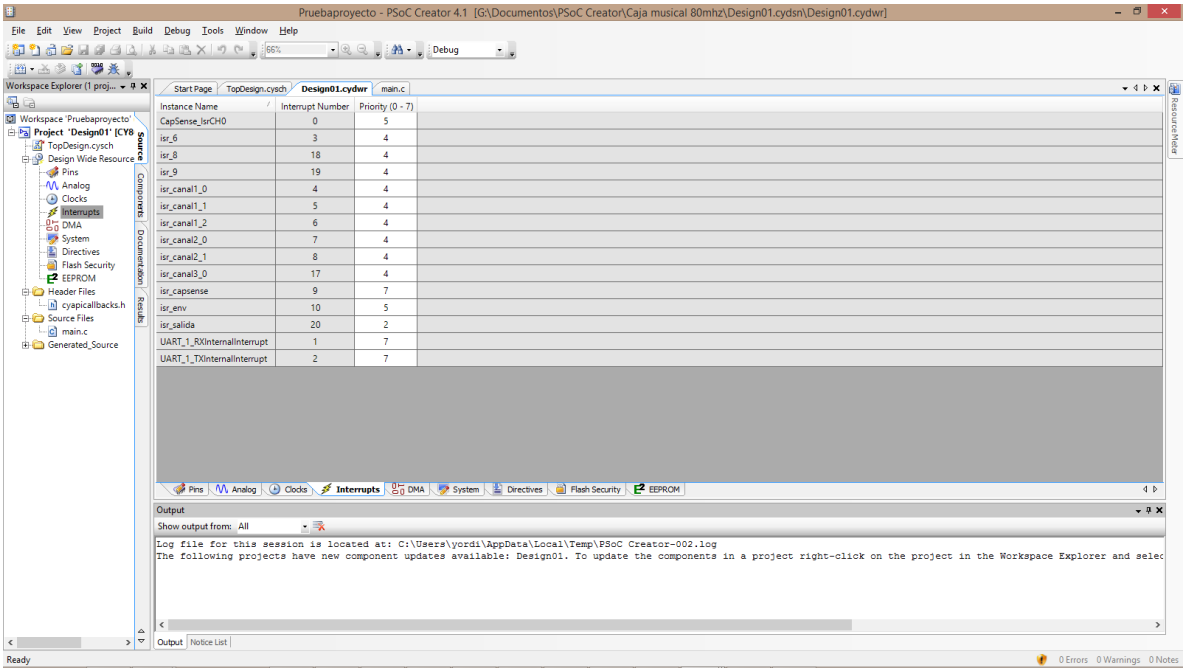
19.- Configurando reloj a 79 MHZ

Cambiamos el valor del PLL a 79 MHZ como se observa en la imagen anterior, no ponemos 80 MHZ dado que es el valor máximo que nos permite poner si marcar errores.

Una vez cambiada la configuración del reloj se procedió a probar nuevamente el dispositivo y se verifico que el dispositivo ahora toca la música con el tempo correcto.

Ahora que hemos aumentado la velocidad del procesador podemos aumentar también la frecuencia de muestreo hasta encontrar un equilibrio entre la calidad del sonido y el rendimiento del dispositivo, probando con diversos valores se decidió utilizar un valor para el timer el cual da aproximadamente una frecuencia de muestreo de 96 KHZ.

Otra cosa que se debe tomar en cuenta para evitar problemas es la configuración de la prioridad de interrupciones de nuestros timers, la mayor prioridad de interrupción deben tenerla las interrupciones que actualizan los valores de las tablas de onda, seguidas por la interrupción que actualiza la envolvente y por último la interrupción que escribe los valores a nuestro DAC, a continuación se muestra una imagen con la configuración de las interrupciones en esta etapa del proyecto.



20.- Configuración de la prioridad de interrupciones

Como se puede observar en la imagen anterior la prioridad de las interrupciones va desde el 0 a 7 donde el 0 tiene la mayor prioridad y el 7 la menor.

Almacenando y reproduciendo melodías desde una memoria microSD

Hasta esta etapa del proyecto tenemos un dispositivo el cual reproduce melodías almacenadas en la memoria de programa del dispositivo sin embargo esta memoria es bastante limitada, por lo cual uno de los objetivos de este proyecto es que las melodías sean almacenadas y reproducidas desde una memoria microSD.

Para lograr esto se analizaron múltiples opciones, una de estas es lograr esta implementación en el mismo dispositivo el cual estamos utilizando para la implementación de nuestro sintetizador de sonido, sin embargo esta opción presenta algunos inconvenientes los cuales se mencionan a continuación.

El primer inconveniente que se nos presenta es que hacer esta implementación en el mismo dispositivo agregaría más carga de trabajo al procesador. Si bien es posible que el procesador sea capaz de manejar esta carga es muy probable que se nos presenten problemas en el tempo de la reproducción de las melodías, ya que el dispositivo estaría leyendo las instrucciones desde la memoria microSD y sintetizando el sonido al mismo tiempo y si se presenta este problema ya no seriamos capaz de solucionarlo ya que actualmente el procesador ya está configurado para trabajar a su máxima capacidad.

El segundo inconveniente es que para la comunicación con la memoria microSD se requiere del uso del protocolo SPI y para implementar este protocolo en nuestro dispositivo es necesario el uso de las UDB recurso que es preciado para nosotros dado que lo ocupamos para la polifonía.

La segunda opción es emplear otro dispositivo para dedicarlo a la tarea de leer las melodías desde la memoria microSD y enviar los comandos mediante algún protocolo de comunicación al dispositivo donde esta implementado el sintetizador, así no agregaríamos mucho más carga a este dado que el sintetizar los sonidos en tiempo real ya consume por si solo bastante tiempo del procesador.

Analizando las 2 opciones la segunda es la que mejor nos pareció implementar dado que no deseamos cargar al procesador del sintetizador con demasiado trabajo, ya que la calidad del sonido depende en gran parte de la velocidad con la que este ejecuta las instrucciones.

Ahora que se han analizado estos puntos también hay que decidir un formato para el almacenamiento de nuestras melodías, en un principio podrían ser almacenadas como archivos de texto que contengan las melodías en formato de instrucciones como las que nuestro sintetizador ya reproduce. Sin embargo esto requeriría hacer uso de la herramienta anteriormente usada para cada melodía que se dese almacenar además de que el formato es poco eficiente en cuestión de espacio de almacenamiento.

Por otra parte podemos hacer uso del formato MIDI que es un formato muy conveniente para este tipo de aplicación ya fue creado para este propósito. Por lo cual entonces se decidido que las melodías serán almacenadas en formato MIDI.

El formato MIDI es un formato utilizado para almacenar melodías este formato no almacena sonido solo contiene una representación de cómo se debe tocar la melodía, esta representación está dada por eventos midi los cuales son de longitud variable y pueden ocupar desde 2 a 8 bytes, algunos eventos MIDI y se presentan en la siguiente tabla.

Byte de estado	Significado	Datos
ox8-	Liberación de nota	1 byte de tono 1 byte de velocidad
ox9-	Pulsación de nota	1 byte de tono 1 byte de velocidad
oxA-	Presión de tecla	1 byte de tono 1 byte de presión (después de pulsar)
oxB-	Parámetro	1 byte de número de parámetro 1 byte de valor
oxC-	Programa	1 byte de programa seleccionado
oxD-	Presión del canal	1 byte de presión (después de pulsar)
oxE-	*Pitch wheel*	2 bytes que proporcionan 14 bits, los 7 de menor peso primero.

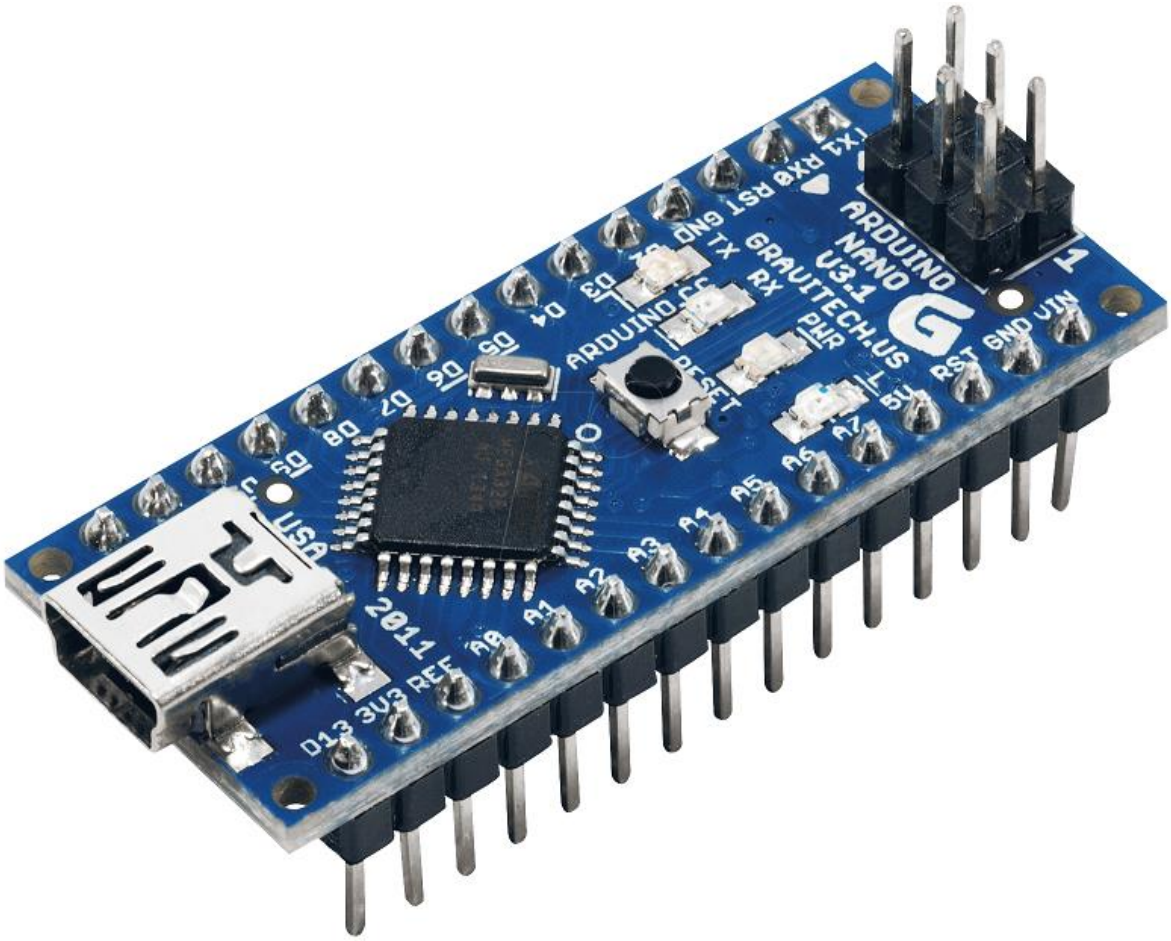
21.- Tabla con algunos eventos MIDI

No daremos más detalles sobre este formato dado que es bastante extenso y si se quiere saber más sobre este consultar las referencias [6] y [7].

Ahora que se ha decidido el formato en el que se almacenara la melodía, y que se ha decidido utilizar otro dispositivo para implementar la lectura y reproducción de las melodías desde la memoria microSD, hay que seleccionar un dispositivo para realizar esto. Dado que el formato MIDI es sencillo de entender pero bastante extenso, la programación para implementar la reproducción de este formato sería bastante y requeriría mucho tiempo de codificación y depuración. Por lo cual entonces hay que seleccionar un dispositivo que contenga alguna librería disponible para esta tarea, con el fin de facilitar este trabajo dado que tan solo la implementación de este formato podría requerir el tiempo que tenemos destinado para la realización de este proyecto.

Analizando diversas opciones se encontró que la plataforma Arduino cuenta con una librería para lograr nuestro propósito.

Arduino es un proyecto que se compone tanto de software como de hardware opensource y es una plataforma para la cual mucha gente desarrolla código y diversas librerías. Tiene varias plataformas de desarrollo en ellas el más popular es el arduino uno sin embargo nosotros no elegiremos este dado que deseamos un dispositivo más compacto para poder encajar todo nuestro sistema dentro de una pequeña caja. Entonces para esto elegimos el arduino nano que tiene prestaciones similares a las del arduino uno pero en un factor de forma mucho más pequeño a continuación se muestra una imagen de este dispositivo.



22.- Arduino nano

La librería a utilizar es una librería llama MD_MIDIFile cuya liga url se puede encontrar en la sección de referencias al final de este documento.

Primero para aprender a utilizar esta librería probamos uno de los códigos de ejemplo el código es el siguiente

```
#include <SdFat.h>
#include <MD_MIDIFile.h>

#define USE_MIDI       1

#if USE_MIDI // set up for direct MIDI serial output

#define DEBUG(x)
#define DEBUGX(x)
#define SERIAL_RATE    31250

#else // don't use MIDI to allow printing debug statements

#define DEBUG(x)       Serial.print(x)
#define DEBUGX(x)      Serial.print(x, HEX)
#define SERIAL_RATE    57600

#endif // USE_MIDI


// SD chip select pin for SPI comms.
// Arduino Ethernet shield, pin 4.
// Default SD chip select is the SPI SS pin (10).
// Other hardware will be different as documented for that hardware.
#define  SD_SELECT  10

// LED definitions for user indicators
#define READY_LED                7     // when finished
#define SMF_ERROR_LED  6       // SMF error
#define SD_ERROR_LED   5       // SD error
#define BEAT_LED                 6     // toggles to the 'beat'

#define WAIT_DELAY     2000    // ms

#define ARRAY_SIZE(a)   (sizeof(a)/sizeof(a[0]))

// The files in the tune list should be located on the SD card
// or an error will occur opening the file and the next in the
// list will be opened (skips errors).
char *tuneList[] =
{
  "1.MID",
/*      "LOOPDEMO.MID",  // simplest and shortest file
  "ELISE.MID",
        "TWINKLE.MID",
        "GANGNAM.MID",
        "FUGUEGM.MID",
        "POPCORN.MID",
        "AIR.MID",
        "PRDANCER.MID",
        "MINUET.MID",
        "FIRERAIN.MID",
        "MOZART.MID",
```

```
        "FERNANDO.MID",
        "SONATAC.MID",
        "SKYFALL.MID",
        "XMAS.MID",
        "GBROWN.MID",
        "PROWLER.MID",
        "IPANEMA.MID",
        "JZBUMBLE.MID",
*/};

// These don't play as they need more than 16 tracks but will run if
MIDIFile.h is changed
//#define MIDI_FILE  "SYMPH9.MID"            // 29 tracks
//#define MIDI_FILE  "CHATCHOO.MID"          // 17 tracks
//#define MIDI_FILE  "STRIPPER.MID"          // 25 tracks

SdFat   SD;
MD_MIDIFile SMF;

void midiCallback(midi_event *pev)
// Called by the MIDIFile library when a file event needs to be processed
// thru the midi communications interface.
// This callback is set up in the setup() function.
{
#if USE_MIDI
        if ((pev->data[0] >= 0x80) && (pev->data[0] <= 0xe0))
        {
                Serial.write(pev->data[0] | pev->channel);
                Serial.write(&pev->data[1], pev->size-1);
        }
        else
                Serial.write(pev->data, pev->size);
#endif
  DEBUG("\n");
  DEBUG(millis());
  DEBUG("\tM T");
  DEBUG(pev->track);
  DEBUG(":  Ch ");
  DEBUG(pev->channel+1);
  DEBUG(" Data ");
  for (uint8_t i=0; i<pev->size; i++)
  {
        DEBUGX(pev->data[i]);
    DEBUG(' ');
  }
}

void sysexCallback(sysex_event *pev)
// Called by the MIDIFile library when a system Exclusive (sysex) file
event needs
// to be processed through the midi communications interface. Most sysex
events cannot
// really be processed, so we just ignore it here.
// This callback is set up in the setup() function.
{
  DEBUG("\nS T");
  DEBUG(pev->track);
```

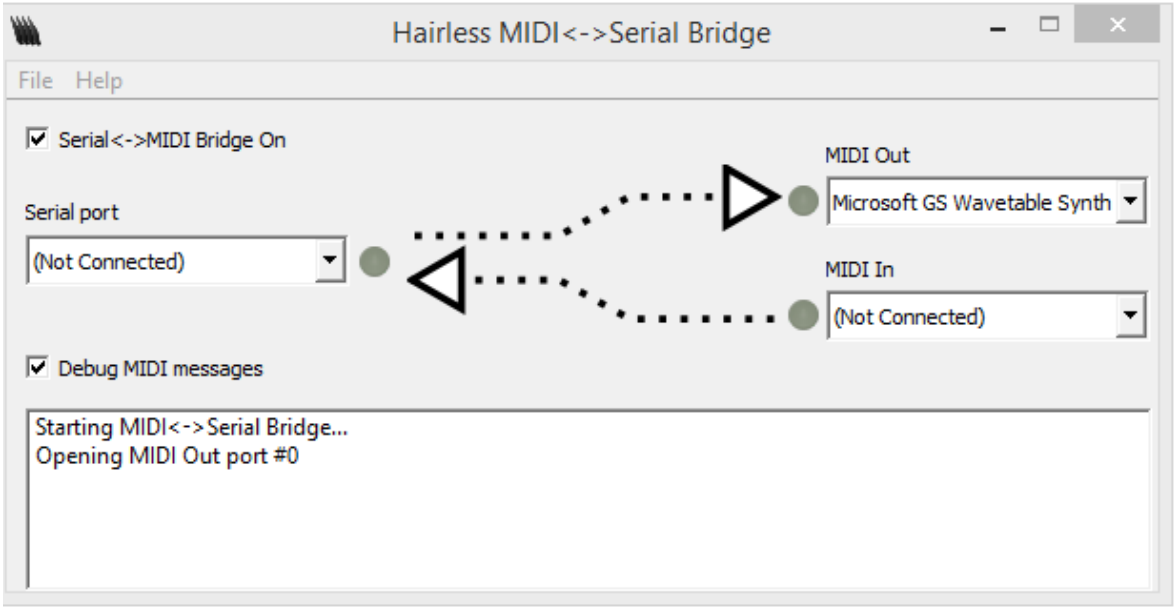
```
  DEBUG(": Data ");
  for (uint8_t i=0; i<pev->size; i++)
  {
    DEBUGX(pev->data[i]);
        DEBUG(' ');
  }
}

void midiSilence(void)
// Turn everything off on every channel.
// Some midi files are badly behaved and leave notes hanging, so between
songs turn
// off all the notes and sound
{
        midi_event      ev;

        // All sound off
        // When All Sound Off is received all oscillators will turn off,
and their volume
        // envelopes are set to zero as soon as possible.
        ev.size = 0;
        ev.data[ev.size++] = 0xb0;
        ev.data[ev.size++] = 120;
        ev.data[ev.size++] = 0;

        for (ev.channel = 0; ev.channel < 16; ev.channel++)
                midiCallback(&ev);
}

void setup(void)
{
  // Set up LED pins
  pinMode(READY_LED, OUTPUT);
  pinMode(SD_ERROR_LED, OUTPUT);
  pinMode(SMF_ERROR_LED, OUTPUT);

  Serial.begin(SERIAL_RATE);

  DEBUG("\n[MidiFile Play List]");

  // Initialize SD
  if (!SD.begin(SD_SELECT, SPI_FULL_SPEED))
  {
    DEBUG("\nSD init fail!");
    digitalWrite(SD_ERROR_LED, HIGH);
    while (true) ;
  }

  // Initialize MIDIFile
  SMF.begin(&SD);
  SMF.setMidiHandler(midiCallback);
  SMF.setSysexHandler(sysexCallback);

  digitalWrite(READY_LED, HIGH);
}

void tickMetronome(void)
```

```
// flash a LED to the beat
{
        static uint32_t lastBeatTime = 0;
        static boolean  inBeat = false;
        uint16_t        beatTime;

        beatTime = 60000/SMF.getTempo();              // msec/beat =
((60sec/min)*(1000 ms/sec))/(beats/min)
        if (!inBeat)
        {
                if ((millis() - lastBeatTime) >= beatTime)
                {
                        lastBeatTime = millis();
                        digitalWrite(BEAT_LED, HIGH);
                        inBeat = true;
                }
        }
        else
        {
                if ((millis() - lastBeatTime) >= 100) // keep the flash on
for 100ms only
                {
                        digitalWrite(BEAT_LED, LOW);
                        inBeat = false;
                }
        }

}

void loop(void)
{
    int  err;

        for (uint8_t i=0; i<ARRAY_SIZE(tuneList); i++)
        {
          // reset LEDs
          digitalWrite(READY_LED, LOW);
          digitalWrite(SD_ERROR_LED, LOW);

          // use the next file name and play it
    DEBUG("\nFile: ");
    DEBUG(tuneList[i]);
          SMF.setFilename(tuneList[i]);
          err = SMF.load();
          if (err != -1)
          {
                DEBUG("\nSMF load Error ");
                DEBUG(err);
                digitalWrite(SMF_ERROR_LED, HIGH);
                delay(WAIT_DELAY);
          }
          else
          {
                // play the file
                while (!SMF.isEOF())
                {
                        if (SMF.getNextEvent())
```

```
                tickMetronome();
            }

            // done with this one
            SMF.close();
            midiSilence();

            // signal finish LED with a dignified pause
            digitalWrite(READY_LED, HIGH);
            delay(WAIT_DELAY);
        }
    }
}
```

Este código reproduce continuamente una lista de canciones las cuales están definidas en el programa, para hacer uso de este código se añadió un archivo midi a la microSD con el nombre 1.MIDI el cual está definido en el código de ejemplo.

Para comprobar que efectivamente está reproduciendo los archivos midi hacemos uso de una herramienta llamada Hairless MIDI<->Serial Bridge a continuación se muestra una captura de pantalla de la interface de esta herramienta y si se le da clic te envía a un vínculo para su descarga

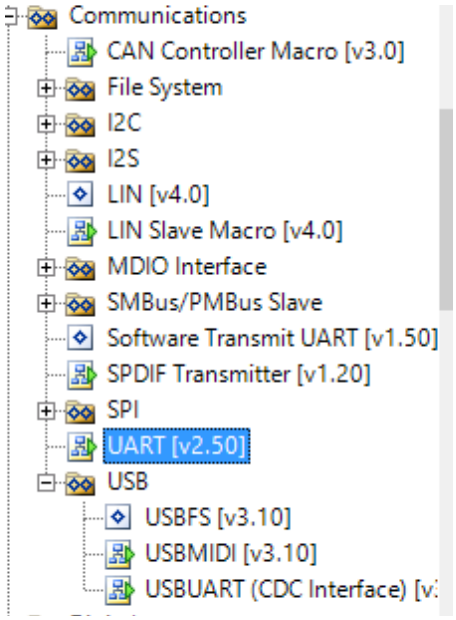


Esta herramienta es totalmente opensource y lo que hace es crear un puente entre el puerto de comunicación serial del arduino y la computadora funcionando así como un dispositivo midi.

En la parte de abajo se pueden comprobar los mensajes midi que se están recibiendo por parte del arduino así como escuchar la melodía.

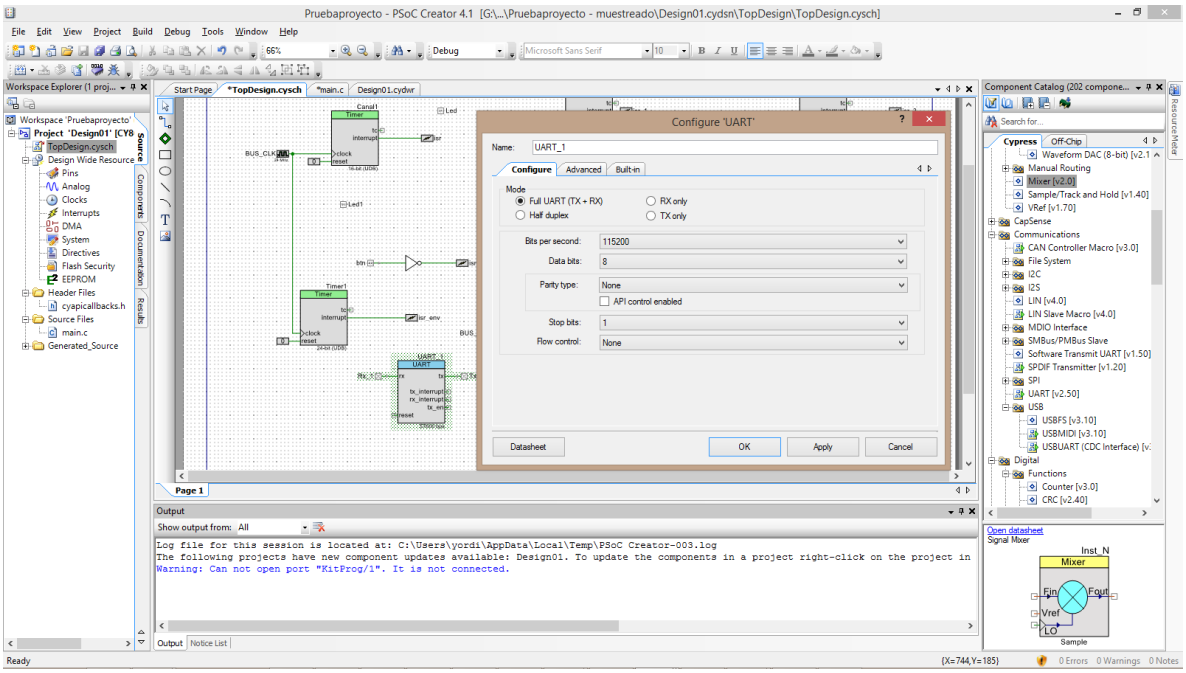
Una vez comprobado que el código de ejemplo funciona hay que encontrar una forma de enviar estos comandos hacia nuestro sintetizador que está en nuestro psoc 5. Para esto vamos a utilizar la comunicación serial por lo que lo primero que se debe hacer es configurar el hardware del psoc para que este implemente esta configuración en hardware dentro de los bloques de lógica programable que contiene.

Primero dentro del IDE de Psoc Creator ubicamos la pestaña comunications y arrastramos nuestro bloque a nuestro diagrama de hardware como ya hemos hecho anteriormente.



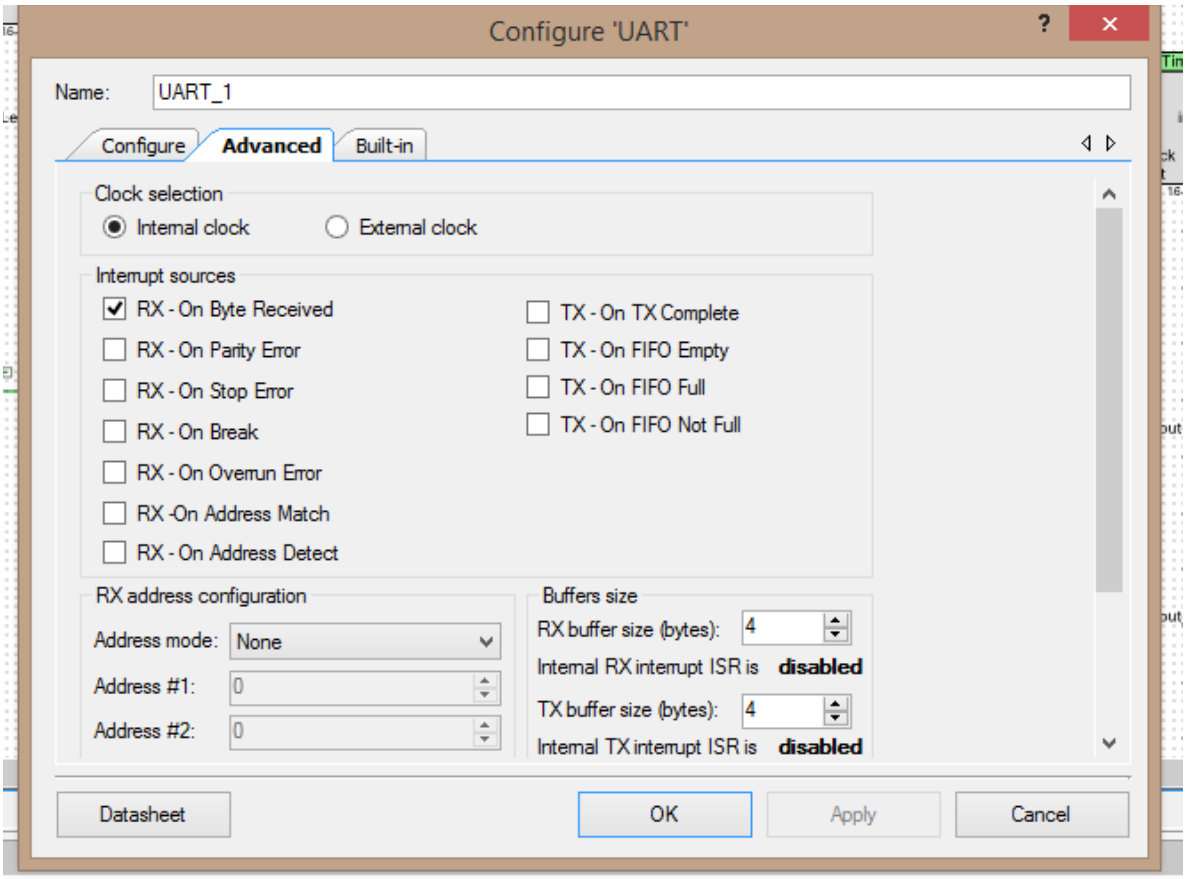
23.- Agregando un bloque de comunicación serial a nuestro diseño

Una vez que tenemos el bloque de hardware agregado en nuestro diseño hay que configurarlo para ello hacemos doble clic en este y nos saldrá una ventana de configuración como la que se muestra a continuación.



Dentro de esta pestaña hay diversos parámetros que se pueden configurar como el tipo de comunicación que se desea realizar puede ser solo de transmisión o solo de recepción, halfduplex o full duplex dependiendo del tipo de comunicación que se desee realizar, la implementación ocupara más o menos bloques UDB. Otro parámetro es la velocidad de transmisión que está dada en baudios, un parámetro común para esta es 115200 baudios, después nos encontramos con el número de bits a enviar por transmisión que normalmente son 8, los siguientes parámetro son el bit de paridad para checar errores, el bit de parada y el control de flujo.

Después de esta pestaña mostrada tenemos la configuración avanzada donde se pueden configurar las interrupciones y la cantidad de bits que se puede almacenar en el buffer de salida y de entrada. Para entender todo esto se puede consultar el datasheet del componente que se encuentra disponible dentro del mismo IDE al seleccionar el componente, una imagen de esta ventana de configuración y una tabla con algunas APIS obtenidas del datasheet se muestran a continuación.



24.- Configuración avanzada del componente UART

Como se puede observar esta configuracion avanzada requiere de un mejor entendimiento del bloque de cominicacion serial por lo que para utilizar esta es necesario el consultar el datasheet de este componente.

Function	Description
UART_DisableRxInt()	Disables the internal interrupt irq
UART_SetRxInterruptMode()	Configures the RX interrupt sources enabled
UART_ReadRxData()	Returns the data in the RX Data register
UART_ReadRxStatus()	Returns the current state of the status register
UART_GetChar()	Returns the next byte of received data
UART_GetByte()	Reads the UART RX buffer immediately and returns the received character and error condition
UART_GetRxBufferSize()	Returns the number of received bytes available in the RX buffer.
UART_ClearRxBuffer()	Clears the memory array of all received data
UART_SetRxAddressMode()	Sets the software-controlled Addressing mode used by the RX portion of the UART
UART_SetRxAddress1()	Sets the first of two hardware-detectable addresses
UART_SetRxAddress2()	Sets the second of two hardware-detectable addresses
UART_EnableTxInt()	Enables the internal interrupt irq
UART_DisableTxInt()	Disables the internal interrupt irq
UART_SetTxInterruptMode()	Configures the TX interrupt sources enabled
UART_WriteTxData()	Sends a byte without checking for buffer room or status
UART_ReadTxStatus()	Reads the status register for the TX portion of the UART
UART_PutChar()	Puts a byte of data into the transmit buffer to be sent when the bus is available
UART_PutString()	Places data from a string into the memory buffer for transmitting
UART_PutArray()	Places data from a memory array into the memory buffer for transmitting
UART_PutCRLF()	Writes a byte of data followed by a Carriage Return and Line Feed to the transmit buffer
UART_GetTxBufferSize()	Returns the number of bytes in the TX buffer which are waiting to be transmitted.
UART_ClearTxBuffer()	Clears all data from the TX buffer
UART_SendBreak()	Transmits a break signal on the bus
UART_SetTxAddressMode ()	Configures the transmitter to signal the next bytes as address or data
UART_LoadRxConfig()	Loads the receiver configuration. Half Duplex UART is ready for receive byte
UART_LoadTxConfig()	Loads the transmitter configuration. Half Duplex UART is ready for transmit byte
UART_Sleep()	Stops the UART operation and saves the user configuration

25.- Tabla con algunas APIS disponibles para uso del UART

Como se puede observar el entorno de desarrollo provee varias APIS para la fácil manipulación de este componente y para simplificar la implementación de la comunicación serial.

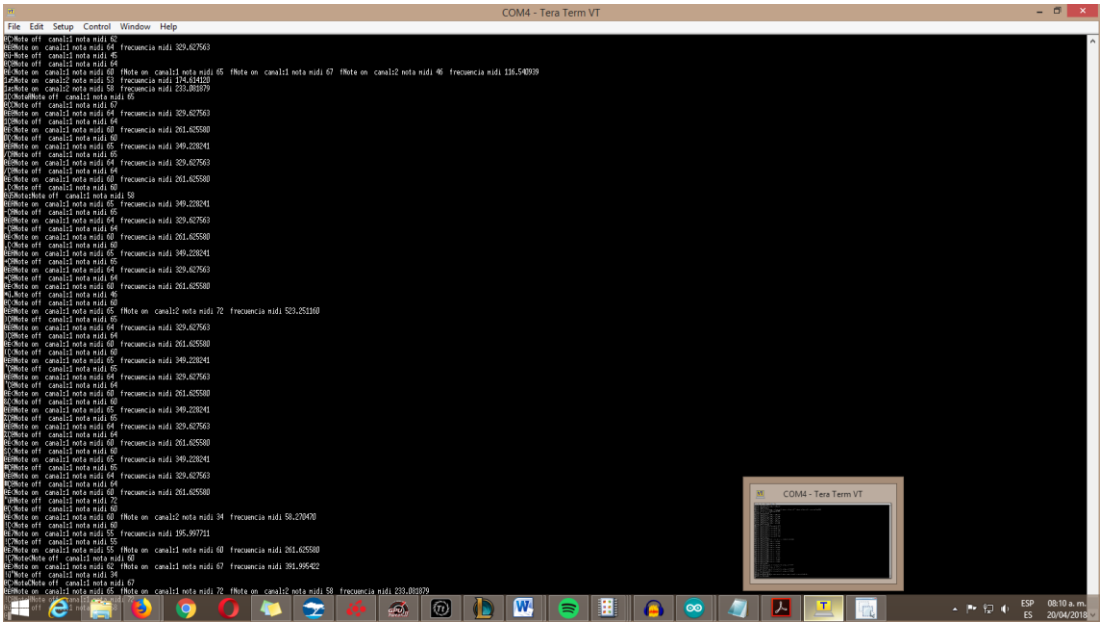
Integrando Arduino y Psoc 5

Después de hacer la configuración del componente UART se trabajó en el código para obtener una buena comunicación entre nuestro Psoc 5 y el arduino se hicieron diversas pruebas hasta obtener el código siguiente.

```
#include "project.h"
#include <stdio.h>
#include <stdbool.h>


char str[50],str2[3];
int i=0,j=0;
int canal=0;
int bytes=0;
_Bool note_on=false,note_off=false;
uint8 count=0,data[3],recibir;
CY_ISR(rx){



}
int main(void)
{
    asm (".global _scanf_float");
    CyGlobalIntEnable; /* Enable global interrupts. */

    isr_RX_StartEx(rx);
    UART_1_Start();
    /* Place your initialization/startup code here (e.g. MyInst_Start())
*/

    for(;;)
    {

  while(UART_1_GetRxBufferSize()>=1){
 recibir=UART_1_GetChar();

        UART_1_WriteTxData(recibir);


    if((recibir&0x0f0)==0x90){
        data[i]=recibir;

       i=1;
    }else{
        if(i==1){
        data[i]=recibir;
        canal=data[i-1]&0x0f;
        sprintf(str,"Note on  canal:%d",canal);
        UART_1_PutString(str);
        sprintf(str," nota midi %d ",data[i]);
        UART_1_PutString(str);
        UART_1_PutCRLF(1);
        i=0;
```

```
            }
        }
         if((recibir&0x0f0)==0x80){
            data[i]=recibir;

           j=1;
        }else{
            if(j==1){
            data[j]=recibir;
            canal=data[j-1]&0x0f;
            sprintf(str,"Note off  canal:%d",canal);
            UART_1_PutString(str);
            sprintf(str," nota midi %d ",data[j]);
            UART_1_PutString(str);
            UART_1_PutCRLF(1);
            j=0;
            }
        }

    }
  }
}
```

Cabe mencionar que para obtener el código Anterior hubo que hacer bastante depuración y estar trabajando con la herramienta utilizada para establecer el puente entre el arduino y la computadora y también con una terminal serial y estar comparando los valores enviados desde el Arduino hacia el Psoc y hacer un tratamiento de estos mismos datos para que el psoc pueda interpretarlos. Una imagen de como se ve la depuración de los mensajes midi en la terminal se muestra a continuación.



26.- Depuración de los mensajes MIDI con ayuda de una terminal

Una vez establecida la comunicación serial entre nuestro arduino y nuestro Psoc solo queda integrar el código con el del sintetizador para que cada vez que se reciba un mensaje MIDI que el sintetizador tenga que interpretar, este haga lo correspondiente dicho mensaje y si no simplemente lo ignore.

Para esto entonces haremos uso de las funciones ya creadas en nuestro sintetizador la cual es la función Tone. Mientras el buffer del UART no este vacío el código buscara por comandos MIDI de su interés como lo son el comando de nota encendida y nota apagada cuyo valor hexadecimal es 0x90 y 0x80 respectivamente cualquier otro mensaje midi que no sea estos 2 el sintetizador simplemente los ignora. Para esto entonces el código desarrollado es el siguiente:

```
while(UART_1_GetRxBufferSize()>=1){
    recibir=UART_1_GetChar();
    UART_1_WriteTxData(recibir);

    if((recibir&0x0f0)==0x90){
        data[i]=recibir;

       i=1;
    }else{
        if(i==1){
        data[i]=recibir;
        canal=(data[i-1]&0x0f)+1;
        sprintf(str,"Note on  canal:%d",canal);
        UART_1_PutString(str);
        sprintf(str," nota midi %d ",data[i]);
        UART_1_PutString(str);
        freq=notes[data[i]];
        sprintf(str," frecuencia midi %f ",freq);
        UART_1_PutString(str);
        UART_1_PutCRLF(1);
        tone(canal,freq);
        i=0;
        }
    }
     if((recibir&0x0f0)==0x80){
        data[i]=recibir;

       j=1;
    }else{
        if(j==1){
        data[i]=recibir;
        canal=(data[i-1]&0x0f)+1;
        sprintf(str,"Note off  canal:%d",canal);
        UART_1_PutString(str);
        sprintf(str," nota midi %d ",data[i]);
        UART_1_PutString(str);
        UART_1_PutCRLF(1);
        noTone(canal);
        j=0;
        }
    }

  }
```

El código del sintetizador entonces sería el siguiente:

```
#include "project.h"
#include <math.h>
#include <stdio.h>
#define TABLE_LENGTH 64
#define msPerTick 1
#define reloj 39500000

void tone(int canal, float  frecuencia);
void noTone(int canal);
void delay (uint32 ms);
int count=0,count2=0,count3=0,count4=0,count5=0;
int env_index=0,env_index2=0,env_index3=0,env_index4=0,env_index5=0;
int x=0,x2=0,x3=0,x4=0,x5=0;
int poly=0,poly2=0;
uint8 result=0,result2=0;
char str[100],str2[3];
int i=0,j=0;
int canal=0;
int acc_canal1[3]={0,0,0},acc_canal2[3]={0,0,0},acc_canal3[3]={0,0,0};
float freq;

uint8 data[3],recibir;
CYCODE const uint8 sineTable[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,
2,3,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,2
20,188
};


CYCODE const uint8 sineTable2[TABLE_LENGTH] = {
142,121,115,92,55,46,89,137,138,128,133,136,126,124,129,139,139,106,76,67
,52,25,2,27,71,79,76,88,98,100,102,113,141,167,157,132,114,100,71,41,50,
84,106,120,138,151,154,155,157,180,201,185,155,137,132,108,63,46,64,79,89
,97,101,107
};

CYCODE const uint8 envelope[TABLE_LENGTH*2] = {

255, 250, 245, 240, 235, 231, 226, 222, 217, 213, 209, 205, 201, 197,
193, 189, 185, 182, 178, 174, 171, 168,
    164, 161, 158, 155, 152, 149, 146, 143, 140, 137, 134, 132, 129, 127,
124, 122, 119, 117, 115, 112, 110,
    108, 106, 104, 102, 100,  98,  96,  94,  92,  90,  88,  87,  85,  83,
82,  80,  78,  77,  75,  74,  72,
    71,  69,  68,  67,  65,  64,  63,  62,  60,  59,  58,  57,  56,  55,
54,  53,  51,  50,  49,  48,  48,
    47,  46,  45,  44,  43,  42,  41,  40,  39,  38,  37,  35,  34,  33,
32,  31,  30,  29,  28,  26,  25,
    24,  23,  22,  21,  20,  19,  17,  16,  15,  14,  13,  12,  11,  10,
8,   7,   6,   5,   4,   3,   2,   0

};
CYCODE const float notes[TABLE_LENGTH*2] = {
```

```
8.1757994,8.6619568,9.1770239,9.7227182,10.300861,10.913383,11.562325,12.
249857,12.978271,13.750000,14.567617,15.433853,
16.351599,17.323914,18.354048,19.445436,20.601723,21.826765,23.124651,24.
499714,25.956543,27.500000,29.135235,30.867706,32.703197,
34.647827,36.708096,38.890873,41.203445,43.653530,46.249302,48.999428,51.
913086,55,58.270470,61.735413,65.406395,69.295654,73.416191,
77.781746,82.406891,87.307060,92.498604,97.998856,103.82617,110,116.54094
,123.47083,130.81279,138.59131,146.83238,155.56349,164.81378,
174.61412,184.99721,195.99771,207.65234,220,233.08188,246.94165,261.62558
,277.18262,293.66476,311.12698,329.62756,349.22824,369.99442,
391.99542,415.30469,440,466.16376,493.88330,523.25116,554.36523,587.32953
,622.25397,659.25513,698.45648,739.98883,783.99084,830.60938,880,
932.32751,987.76660,1046.5023,1108.7305,1174.6591,1244.5079,1318.5103,139
6.9130,1479.9777,1567.9817,1661.2188,1760,1864.6550,1975.5332,
2093.0046,2217.4609,2349.3181,2489.0159,2637.0205,2793.8259,2959.9553,313
5.9634,3322.4375,3520,3729.3101,3951.0664,4186.0093,4434.9219,
4698.6362,4978.0317,5274.0410,5587.6519,5919.9106,6271.9268,6644.8750,704
0,7458.6201,7902.1328,8372.0186,8869.8438,9397.2725,9956.0635,
10548.082,11175.304,11839.821,12543.854
};

CY_ISR(int_canal1_0){

  if(env_index<=127){

     if(count<=TABLE_LENGTH){
        acc_canal1[0]=(sineTable[count]*envelope[env_index])/255;



        count++;
    }else{

         count=0;
    }
   }else{
         x=0;

        }

    Canal1_ReadStatusRegister();
}
CY_ISR(int_canal2_0){

    if(env_index2<=127){

     if(count2<=TABLE_LENGTH){
         x3=((sineTable[count2]*envelope[env_index2])/255);

        count2++;
    }else{

         count2=0;
    }
   }else{
     x3=0;
```

```
        }

   Canal2_ReadStatusRegister();
}
CY_ISR(int_canal3_0){

  if(env_index3<=127){

     if(count3<=TABLE_LENGTH){
        x4=(sineTable[count3]*envelope[env_index3]);
       x4=x4/255;


        VDAC8_1_SetValue(result);
        count3++;
    }else{

         count3=0;
    }
   }else{

       x4=0;
        }

    Canal1_ReadStatusRegister();
}
CY_ISR(int_canal1_1){

  if(env_index4<=127){

     if(count4<=TABLE_LENGTH){
        acc_canal1[1]=(sineTable[count4]*envelope[env_index4])/255;




        count4++;
    }else{

         count4=0;
    }
   }else{
        x2=0;

        }

    Canal1_1_ReadStatusRegister();
}
CY_ISR(int_canal2_1){

  if(env_index5<=127){

     if(count5<=TABLE_LENGTH){
        x5=(sineTable[count5]*envelope[env_index5]);
       x5=x5/255;
```

```
        count5++;
    }else{

         count5=0;
    }
   }else{
        x5=0;

        }

    Canal2_1_ReadStatusRegister();
}


CY_ISR(env_update){
env_index++;
env_index2++;
env_index3++;
env_index4++;
env_index5++;
envelop_ReadStatusRegister();
}
CY_ISR(DAC){
    result=(acc_canal1[0]+acc_canal1[1])/2;
    result2=(x3+x4+x5)/3;
    VDAC8_2_SetValue(result);
    VDAC8_1_SetValue(result2);
    salida_ReadStatusRegister();
}
int main(void)
{
     asm (".global _scanf_float");
    //Opamp_1_Start();
    PGA_1_Start();
    PGA_2_Start();
    envelop_Start();
    VDAC8_1_Start();
    VDAC8_2_Start();
    //Opamp_3_Start();
    envelop_WritePeriod(400000);
    salida_WritePeriod(200);
    salida_Start();
    PGA_1_SetGain(1);
    PGA_1_SetGain(1);
    UART_1_Start();
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g. MyInst_Start())
*/

    isr_canal1_0_StartEx(int_canal1_0);

    isr_canal2_0_StartEx(int_canal2_0);
    isr_canal3_0_StartEx(int_canal3_0);
    isr_canal1_1_StartEx(int_canal1_1);
    isr_canal2_1_StartEx(int_canal2_1);
```

```
    isr_salida_StartEx(DAC);
    isr_env_StartEx(env_update);

    for(;;)
    {
         while(UART_1_GetRxBufferSize()>=1){
 recibir=UART_1_GetChar();
UART_1_WriteTxData(recibir);
    if((recibir&0x0f0)==0x90){
        data[i]=recibir;

       i=1;
    }else{
        if(i==1){
        data[i]=recibir;
        canal=(data[i-1]&0x0f)+1;
        sprintf(str,"Note on  canal:%d",canal);
        UART_1_PutString(str);
        sprintf(str," nota midi %d ",data[i]);
        UART_1_PutString(str);
        freq=notes[data[i]];
        sprintf(str," frecuencia midi %f ",freq);
        UART_1_PutString(str);
        UART_1_PutCRLF(1);
        tone(canal,freq);
        i=0;
        }
    }
     if((recibir&0x0f0)==0x80){
        data[i]=recibir;

       j=1;
    }else{
        if(j==1){
        data[i]=recibir;
        canal=(data[i-1]&0x0f)+1;
        sprintf(str,"Note off  canal:%d",canal);
        UART_1_PutString(str);
        sprintf(str," nota midi %d ",data[i]);
        UART_1_PutString(str);
        UART_1_PutCRLF(1);
        noTone(canal);
        j=0;
        }
    }

  }


        /* Place your application code here. */
    }
}
void tone(int canal,float frecuencia){
    int periodo;
    if (canal==1){
        if (poly>1)
        poly=0;
```

```
        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        switch(poly)
        {
        case 0:

        env_index=0;
        count=0;
        Canal1_WritePeriod(periodo);
        Canal1_Start();
        ++poly;
        break;
        case 1:
        env_index4=0;
        count4=0;
        Canal1_1_WritePeriod(periodo);
        Canal1_1_Start();
        ++poly;
        break;


        }
    }else if(canal==2){
           if (poly2>1)
           poly2=0;
        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        switch(poly2)
        {
        case 0:

        env_index2=0;
        count2=0;
        Canal2_WritePeriod(periodo);
        Canal2_Start();
        poly2++;
        break;
        case 1:
        env_index5=0;
        count5=0;
        Canal2_1_WritePeriod(periodo);
        Canal2_1_Start();
        poly2++;
        break;
        }
    }
    else if(canal==3){

        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        env_index3=0;
        count3=0;
        Canal3_WritePeriod(periodo);
        Canal3_Start();

    }
}
void noTone(int canal){
    if(canal==1){
```

```
        switch(poly)
        {
        case 1:
        Canal1_Stop();
        poly--;
        break;
        case 2:
      Canal1_1_Stop();
        poly--;
        break;
        }


    }else if(canal==2){
        switch(poly2)
        {
        case 1:
        Canal2_Stop();
        poly2--;
        break;
        case 2:
        Canal2_1_Stop();
        poly2--;
        break;


        }


    }else if(canal==3){
        Canal3_Stop();
    }
  }
void delay(uint32 ms){
    CyDelay(ms);
}
```

Con esto entonces ya tenemos un sintetizador el cual recibe comandos MIDI por parte del Arduino y los interpreta para reproducir melodías.

Desarrollando un menú para la selección de la melodía en el arduino

Con el trabajado realizado hasta ahora tenemos un dispositivo el reproduce melodías almacenadas desde una memoria microSD, sin embargo estas melodías son reproducidas continuamente y tienen que estar establecidas previamente en una lista dentro de la memoria de programa del arduino. Este es un inconveniente dado que cada vez que se deseen agregar melodías para almacenar en la microSD, estas deberán ser declaradas en la lista dentro del programa del arduino y el arduino deberá ser reprogramado. Además de esto existe el hecho de que no se puede seleccionar la melodía a reproducir, ni se puede reproducir cuando uno lo desea. Por esto entonces se procedió a realizar un código que permita visualizar el nombre de la melodía que se desea reproducir en alguna pantalla y mediante algún botón seleccionar este archivo para su reproducción.

Para esto entonces se decidido utilizar en un principio una pantalla de tipo Oled la cual integra una ranura microSD por lo cual es bastante conveniente una imagen de esta pantalla se muestra a continuación.

27.- Pantalla OLED con ranura microSD

Para la realización de este menú se partió de nuevo la base de uno de los códigos de ejemplo de la biblioteca MIDI. El código en un principio estaba hecho para funcionar con una shield de arduino que contiene una pantalla LCD de 16x2 y unos cuantos botones, Para poder hacerlo funcionar con nuestra pantalla hubo que hacer bastantes cambios sin embargo estos se facilitaron gracias a las librerías que nos facilitaron el uso de la pantalla Oled la cual se comunica por medio del protocolo SPI con el arduino.

No se hará mucho énfasis en los detalles sobre qué cambios se hicieron en el código de ejemplo y solo nos limitaremos a mostrar el código desarrollado el cual es el siguiente.

```
#include <SdFat.h>
#include <MD_MIDIFile.h>
#include <MD_AButton.h>

#include "FSMtypes.h"          // FSM enumerated types
#define  LCD_KEYS  KEY_ADC_PORT
#define DEBUG_MODE     0
#define SPI_SPEED SD_SCK_MHZ(4)
#if DEBUG_MODE
#define  LCD_KEYS  KEY_ADC_PORT
#define DEBUG(x)       Serial.print(x)
#define DEBUGX(x)      Serial.print(x, HEX)
#define SERIAL_RATE    115200

#else

#define DEBUG(x)
#define DEBUGX(x)
#define SERIAL_RATE    115200

#endif
#define sclk 15
#define mosi 16
#define cs   10
#define rst  9
#define dc   8
#define  BLACK           0x0000
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1331.h>
#include <SPI.h>

// Option 1: use any pins but a little slower
Adafruit_SSD1331 display = Adafruit_SSD1331(cs, dc, rst);
// SD Hardware defines ---------
// SPI select pin for SD card (SPI comms).
// Arduino Ethernet shield, pin 4.
// Default SD chip select is the SPI SS pin (10).
// Other hardware will be different as documented for that hardware.
#define  SD_SELECT  4
int c;
// LCD display defines ---------

// LCD user defined characters




SdFat   SD;
MD_MIDIFile SMF;
MD_AButton  LCDKey(LCD_KEYS);

// Playlist handling -----------
#define FNAME_SIZE            30                          // 8.3 +
'\0' character file names
#define PLAYLIST_FILE  "PLAYLIST.TXT" // file of file names
```

```
#define MIDI_EXT                ".mid"                  // MIDI file
extension
uint16_t        plCount = 0;

// MIDI callback functions for MIDIFile library ---------------

void midiCallback(midi_event *pev)
// Called by the MIDIFile library when a file event needs to be processed
// thru the midi communications interface.
// This callback is set up in the setup() function.
{
#if !DEBUG_MODE
        if ((pev->data[0] >= 0x80) && (pev->data[0] <= 0xe0))
        {
                Serial1.write(pev->data[0] | pev->channel);
                Serial1.write(&pev->data[1], pev->size-1);
        }

#endif
  DEBUG("\nM T");
  DEBUG(pev->track);
  DEBUG(":  Ch ");
  DEBUG(pev->channel+1);
  DEBUG(" Data ");
  for (uint8_t i=0; i<=pev->size; i++)
  {
        DEBUGX(pev->data[i]);
    DEBUG(' ');
  }
}



void midiSilence(void)
// Turn everything off on every channel.
// Some midi files are badly behaved and leave notes hanging, so between
songs turn
// off all the notes and sound
{
        midi_event      ev;

        // All sound off
        // When All Sound Off is received all oscillators will turn off,
and their volume
        // envelopes are set to zero as soon as possible.
        ev.size = 0;
        ev.data[ev.size++] = 0xb0;
        ev.data[ev.size++] = 120;
        ev.data[ev.size++] = 0;

        for (ev.channel = 0; ev.channel < 16; ev.channel++)
                midiCallback(&ev);
}

// LCD Message Helper functions -----------------
void LCDMessage( const char *msg, bool clrEol = false)
```

```
// Display a message on the LCD screen with optional spaces padding the
end
{

        Serial.print(msg);
        if (clrEol)
        {

                        Serial.write(' ');
        }
}

void LCDErrMessage(const char *msg, bool fStop)
{
        LCDMessage(msg, true);
  DEBUG("\nLCDErr: ");
  DEBUG(msg);
        while (fStop) ;         // stop here if told to
        delay(2000);            // if not stop, pause to show message
}

// Create list of files for menu --------------

uint16_t createPlaylistFile(void)
// create a play list file on the SD card with the names of the files.
// This will then be used in the menu.
{
        SdFile          plFile;         // play list file
        SdFile          mFile;          // MIDI file
        uint16_t        count = 0;// count of files
        char              fname[FNAME_SIZE];

        // open/create the play list file
        if (!plFile.open(PLAYLIST_FILE, O_CREAT|O_WRITE))
                LCDErrMessage("PL create fail", true);

  SD.vwd()->rewind();
  while (mFile.openNext(SD.vwd(), O_READ))
        {
    mFile.getName(fname, FNAME_SIZE);

    DEBUG("\nFile ");
    DEBUG(count);
    DEBUG(" ");
    DEBUG(fname);

                if (mFile.isFile())
                {
                        if (strcmp(MIDI_EXT, &fname[strlen(fname)-
strlen(MIDI_EXT)]) == 0)
                        // only include files with MIDI extension
                        {
                                plFile.write(fname, FNAME_SIZE);
                                count++;
                        }
                }
                mFile.close();
```

```
        }
  DEBUG("\nList completed");

        // close the play list file
        plFile.close();

        return(count);
}

// FINITE STATE MACHINES ------------------------------

seq_state lcdFSM(seq_state curSS)
// Handle selecting a file name from the list (user input)
{
        static lcd_state s = LSBegin;
        static uint8_t plIndex = 0;
        static char    fname[FNAME_SIZE];
        static SdFile  plFile;         // play list file

        // LCD state machine
        switch (s)
        {
        case LSBegin:
                LCDMessage("Select music:", true);
    display.setCursor(0,17);
    display.print("selecciona tu ");
    display.setCursor(0,24);
    display.print("musica: ");
                if (!plFile.isOpen())
                {
                        if (!plFile.open(PLAYLIST_FILE, O_READ))
                                LCDErrMessage("PL file no open", true);
                }
                s = LSShowFile;
                break;

        case LSShowFile:
                plFile.seekSet(FNAME_SIZE*plIndex);
                plFile.read(fname, FNAME_SIZE);

                LCDMessage( fname, true);
    display.setCursor(0,35);
    display.fillRect(0,35, 96, 20, BLACK);
    display.setCursor(0,35);
    display.print(fname);
                Serial.print(plIndex == 0 ? ' ' : '<');
                Serial.print(plIndex == plCount-1 ? ' ' : '>');
                s = LSSelect;
                break;

        case LSSelect:
                switch (LCDKey.getKey())
                // Keys are mapped as follows:
                // Select:     move on to the next state in the state
machine
                // Left:       use the previous file name (move back one
file name)
```

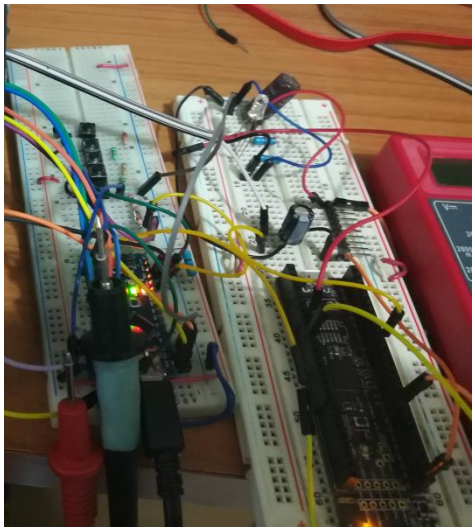
```
                // Right:       use the next file name (move forward one
file name)
                // Up:          move to the first file name
                // Down:        move to the last file name
                {
                        case 'S':       // Select
                                s = LSGotFile;
        display.setCursor(0,35);
        display.print("                       ");
                                break;

                        case 'L':       // Left
                                if (plIndex != 0)
                                        plIndex--;
                                s = LSShowFile;
        display.setCursor(0,35);
        display.print("                       ");
                                break;

                        case 'U':       // Up
                                plIndex = 0;
                                s = LSShowFile;
        display.setCursor(0,35);
        display.print("                       ");
                                break;

                        case 'D':       // Down
                                plIndex = plCount-1;
                                s = LSShowFile;
        display.setCursor(0,35);
        display.print("                       ");
                                break;

                        case 'R':       // Right
                                if (plIndex != plCount-1)
                                        plIndex++;
                                s = LSShowFile;
        display.setCursor(0,35);
        display.print("                       ");
                                break;
        display.setCursor(0,35);
        display.print("                       ");
                        }
                        break;

        case LSGotFile:
                // copy the file name and switch mode to playing MIDI
                SMF.setFilename(fname);
                curSS = MIDISeq;
                // fall through to default state

        default:
                s = LSBegin;
                break;
        }

        return(curSS);
```

```
}

seq_state midiFSM(seq_state curSS)
// Handle playing the selected MIDI file
{
        static midi_state s = MSBegin;

        switch (s)
        {
        case MSBegin:
                // Set up the LCD
                LCDMessage(SMF.getFilename(), true);
                LCDMessage( "K  \xdb  \1  >", true);          // string of
user defined characters
                s = MSLoad;
                break;

        case MSLoad:
                // Load the current file in preparation for playing
                {
                        int  err;

                        // Attempt to load the file
                        if ((err = SMF.load()) == -1)
                        {
                                s = MSProcess;
                        }
                        else
                        {
                                char    aErr[16];

                                sprintf(aErr, "SMF error %03d", err);
                                LCDErrMessage(aErr, false);
                                s = MSClose;
                        }
                }
                break;

        case MSProcess:
                // Play the MIDI file
                if (!SMF.isEOF())
    {
                        if (SMF.getNextEvent())
      {
        char  sBuf[10];

        sprintf(sBuf, "%3d", SMF.getTempo());
        LCDMessage( sBuf, true);
        sprintf(sBuf, "%d/%d", SMF.getTimeSignature()>>8,
SMF.getTimeSignature() & 0xf);
        LCDMessage( sBuf, true);
      };
    }
                else
                        s = MSClose;

                // check the keys
```

```
                switch (LCDKey.getKey())
                {
                        case 'S':       SMF.restart();          break;  //
Rewind
                        case 'L':       s = MSClose;              break;        // Stop
                        case 'U':       SMF.pause(true);        break;  //
Pause
                        case 'D':       SMF.pause(false);       break;  //
Start
                        case 'R':                                                       brea
                }

                break;

        case MSClose:
                // close the file and switch mode to user input
                SMF.close();
                midiSilence();
                curSS = LCDSeq;
                // fall through to default state

        default:
                s = MSBegin;
                break;
        }

        return(curSS);
}

void setup(void)
{
        int  err;
  pinMode(cs, OUTPUT);
  digitalWrite(cs, HIGH);
        // initialise MIDI output stream
        Serial.begin(SERIAL_RATE);
  Serial1.begin(SERIAL_RATE);
  display.begin();
        display.fillScreen(BLACK);
  display.setTextColor(0x101F);
  display.setCursor(0,0);
  display.print("Digital");
  display.setCursor(0,10);
  display.print("Caja de música digital");

        LCDMessage( "  Midi  Player  ", false);
        LCDMessage( "  ------------  ", false);

        // Load characters to the LCD

        pinMode(LCD_KEYS, INPUT);

        // initialise SDFat
        if (!SD.begin(SD_SELECT, SPI_FULL_SPEED)){
                LCDErrMessage("SD init fail!", true);
    display.fillScreen(BLACK);
     display.setCursor(0,10);
```

```
      display.print("Error en la SD");
         }
         plCount = createPlaylistFile();
         if (plCount == 0)
                 LCDErrMessage("No files", true);

         // initialise MIDIFile
         SMF.begin(&SD);
         SMF.setMidiHandler(midiCallback);


         delay(750);             // allow the welcome to be read on the LCD
}

void loop(void)
// only need to look after 2 things - the user interface (LCD_FSM)
// and the MIDI playing (MIDI_FSM). While playing we have a different
// mode from choosing the file, so the FSM will run alternately,
depending
// on which state we are currently in.
{
         static seq_state       s = LCDSeq;

         switch (s)
         {
                 case LCDSeq:  s = lcdFSM(s);   break;
                 case MIDISeq: s = midiFSM(s);  break;
                 default: s = LCDSeq;
         }
}
```

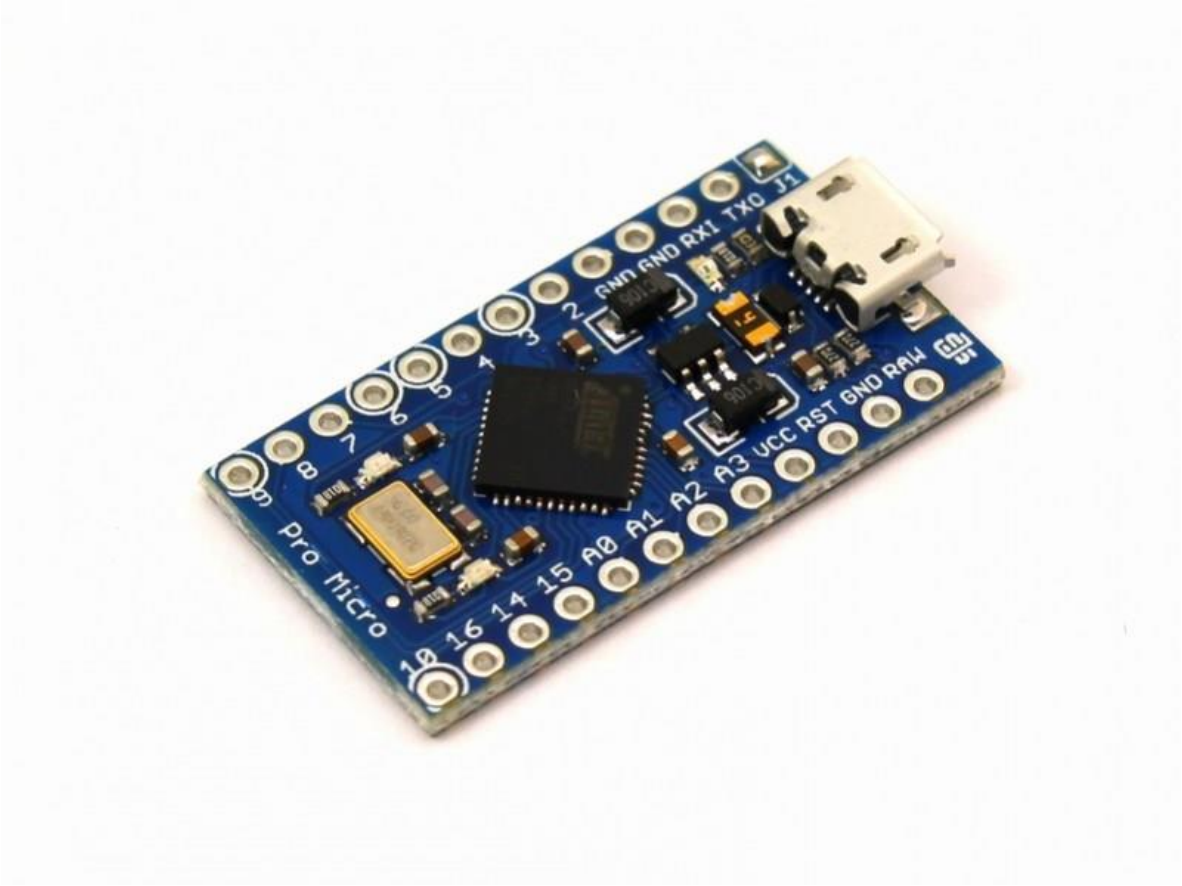
Una imagen de nuestro circuito de prueba hasta el momento se muestra a continuación



28.- Circuito de pruebas Arduino nano y Psoc 5

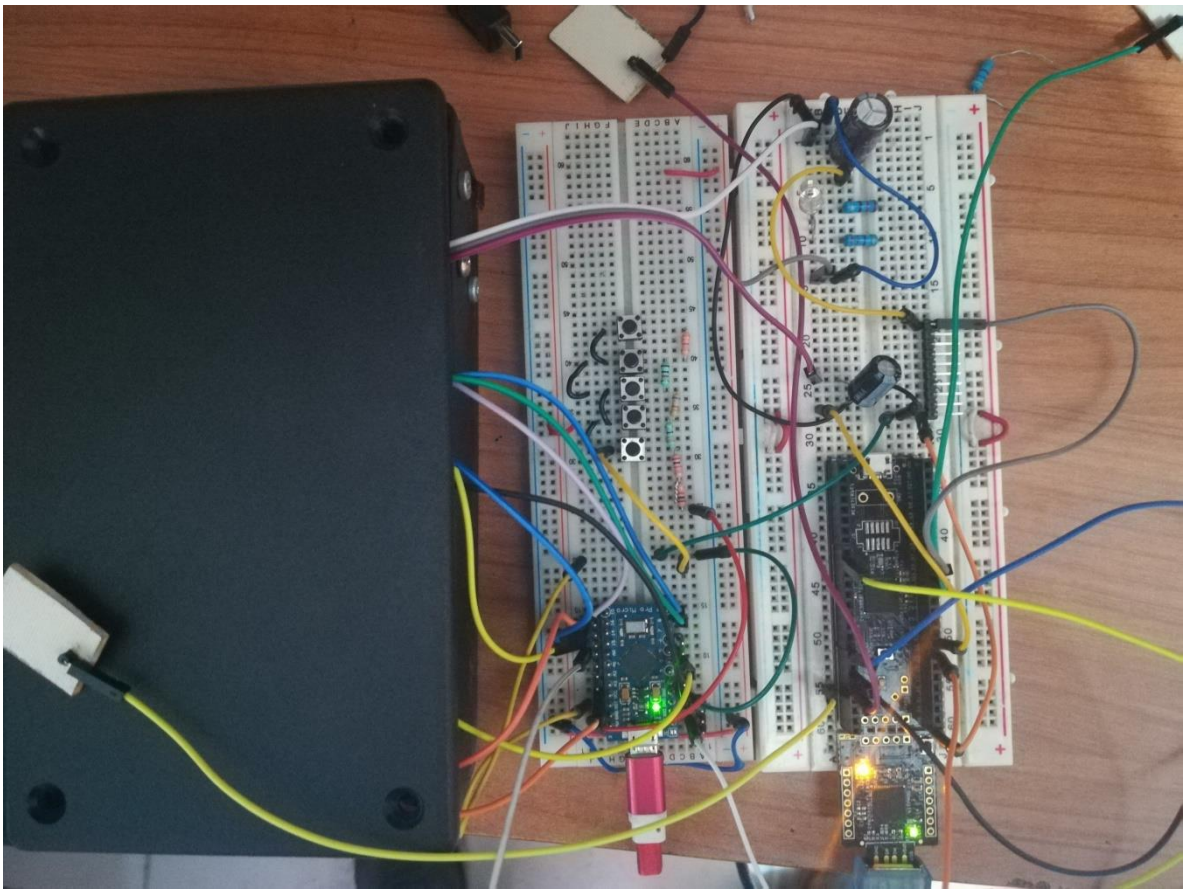
Los comentarios en ingles son comentarios del código de ejemplo y se dejaron en este por practicidad. Aunque se partió de la base del código de ejemplo los cambios hechos al código son significativos y aunque el código se probó y funciono, este trajo consigo un pequeño inconveniente el cual es que dado que el código estaba pensado para el uso con una pantalla LCD las bibliotecas para el manejo de esta ocupan mucho menos espacio y dado que el manejo de una pantalla grafica OLED es mucho más complejo estas bibliotecas almacenan más memoria dinámica y consume más RAM por lo que el uso de la memoria del arduino está en el límite y el IDE nos da la advertencia que se pueden producir errores de estabilidad.

Para intentar solucionar estos problemas de estabilidad debido a la poca memoria RAM del arduino nano se decidió cambiar a un dispositivo con mayores prestaciones, esto en principio fue difícil dado a que se desea un dispositivo de más o menos el mismo tamaño que nuestro arduino nano y la disponibilidad de estos dentro de México es casi nula. Se decidió probar en un principio con un arduino pro micro el cual posee un microcontrolador similar al nano pero con un poco más de memoria RAM en concreto .5 kb más lo cual es muy poco pero es un dispositivo de bajo costo y al cual tenemos acceso por lo cual vale la pena intentar ver si esto soluciona los problemas de estabilidad una imagen de este dispositivo se muestra a continuación como una referencia.



29.- Arduino Pro Micro

Para probar el código en este nuevo arduino no se requirió modificación alguna solo se requirió recopilar el código para que trabajara con esta plataforma una vez hecho esto se procedió a cablear nuestro circuito de pruebas con el nuevo dispositivo y el resultado se en la siguiente imagen.



30.- Circuito de pruebas con Arduino Pro Micro

Aunque los problemas de estabilidad se solucionaron de cierta manera sustituyendo el arduino nano por el Pro Micro este dispositivo aún está demasiado limitado y no nos permite crecer nuestro código para agregar más funcionalidad a este por lo cual se decidió cambiar a un dispositivo con prestaciones aún mayores.

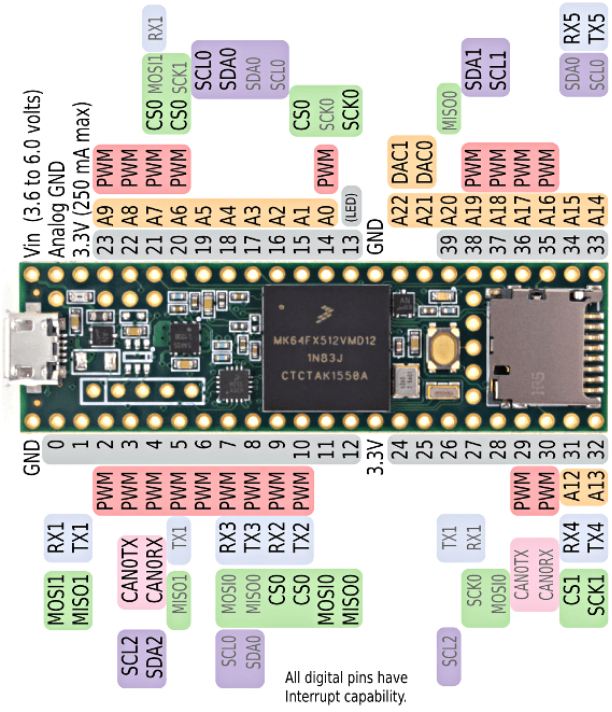
Indagando sobre que dispositivo es más conveniente utilizar en cuanto a tamaño, prestaciones y compatibilidad con nuestro código desarrollado hasta el momento, tras analizar muchas opciones encontramos que la plataforma más conveniente y a la cual podemos tener acceso es una plataforma llamada Teensy 3.5 la cual dice ser compatible con el entorno de desarrollo arduino.

Sustituyendo Arduino por un Teensy 3.5

La plataforma Teensy 3.5 posee un poderoso microcontrolador basado en un núcleo Córtex ARM 4 corriendo a una velocidad de 120MHZ lo cual la hace una poderosa plataforma de desarrollo. No indagaremos mucho en las características de esta plataforma solo haremos hincapié en los puntos más importantes y las mejoras que presenta con respecto al arduino.

Esta nueva plataforma de hardware nos provee de un microcontrolador mucho más robusto y rápido que los de los Arduinos utilizados hasta el momento, las capacidades del microcontrolador de esta plataforma no tienen punto de comparación con las del arduino dado que tan solo el microcontrolador que posee esta tarjeta de desarrollo es de una arquitectura de 32 bits a diferencia de los del arduino nano y Pro Micro los cuales poseen una arquitectura de 8 bits [8].

Dentro de las muchas ventajas que nos provee esta plataforma es la incorporación de una ranura para nuestra tarjeta microSD la cual está conectada directamente a un periférico dedicado dentro del microcontrolador llamado SDIO. No indagaremos en detalle sobre este periférico solo nos limitaremos a mencionar que provee una mejor latencia para la lectura de los datos con nuestra memoria microSD en comparación con la comunicación SPI que se realizaba entre la memoria y nuestro arduino. Una imagen de referencia de esta tarjeta se muestra a continuación.



31.- Tarjeta de desarrollo Teensy 3.5

Una vez que hemos estudiado un poco esta nueva tarjeta de desarrollo y hemos entendido la manera de programarla con nuestro entorno arduino procedimos a probar este nuevo hardware con los códigos anteriormente realizados. Sin embargo nuestros códigos en primera instancia no copilaron. Por lo cual se procedió a realizar modificaciones a estos códigos para hacer que estos mismos funcionaran con el nuevo hardware. Una de las modificaciones que hubo que hacer se presentó en la librería utilizada para el manejo de los archivos MIDI, dado que ahora no se está utilizando la comunicación SPI sino que se está utilizando el periférico dedicado mencionado anteriormente esta biblioteca no está preparada para trabajar con este, por lo cual se hicieron una serie de modificaciones necesarias para que la biblioteca utilizada trabaje con este mismo.

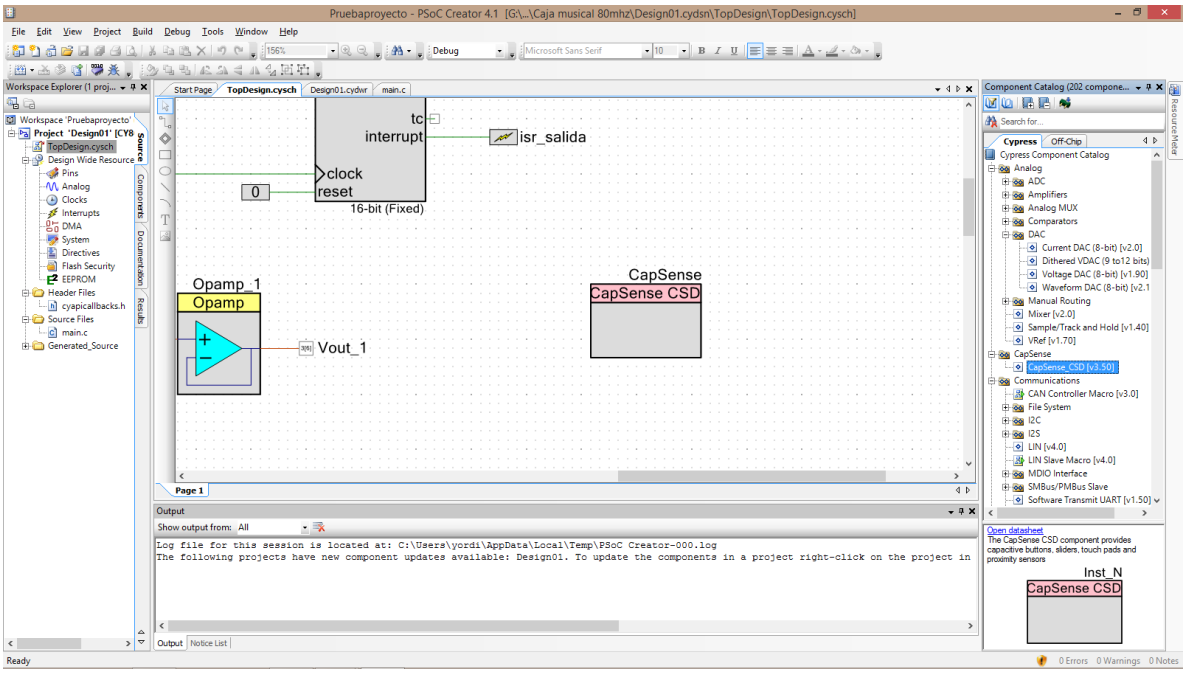
Una vez arreglados los errores se probaron los códigos nuevamente apoyándonos en la herramienta hairless midi, una vez que se comprobó que estos estaban funcionando correctamente se procedió a la etapa final del proyecto.

Etapa Final Del proyecto

Una vez solucionados los problemas con que se han presentado hasta el momento y con la experiencia adquirida hasta ahora, podemos proceder a desarrollar de lo que será el hardware final de este proyecto.

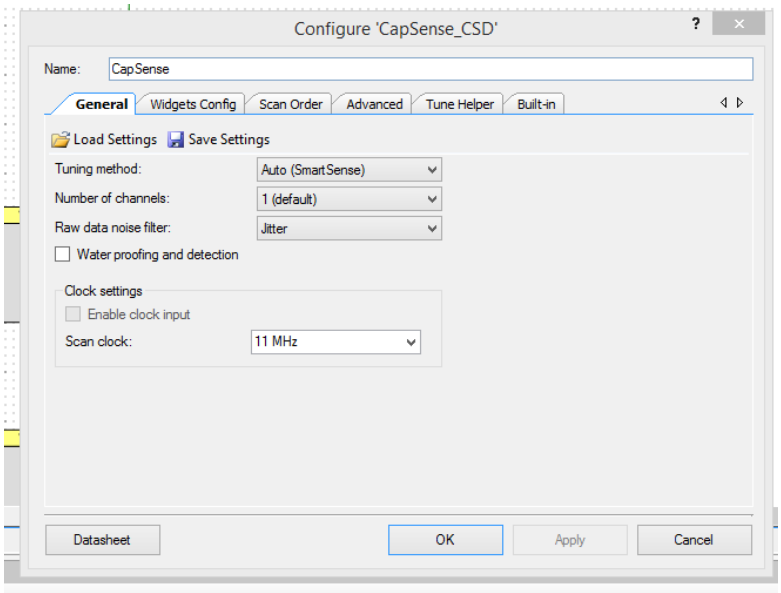
Primero Empezaremos con la configuración final del hardware que se usara en nuestro Psoc. El microcontrolador del Psoc 5 posee hardware especializado para el uso de botones capacitivos haremos uso de esta característica para los botones que controlan nuestro dispositivo.

Lo primero que hay que hacer es agregar un bloque Capsense al diagrama de hardware dentro de nuestro entorno de desarrollo como se muestra en la siguiente imagen.



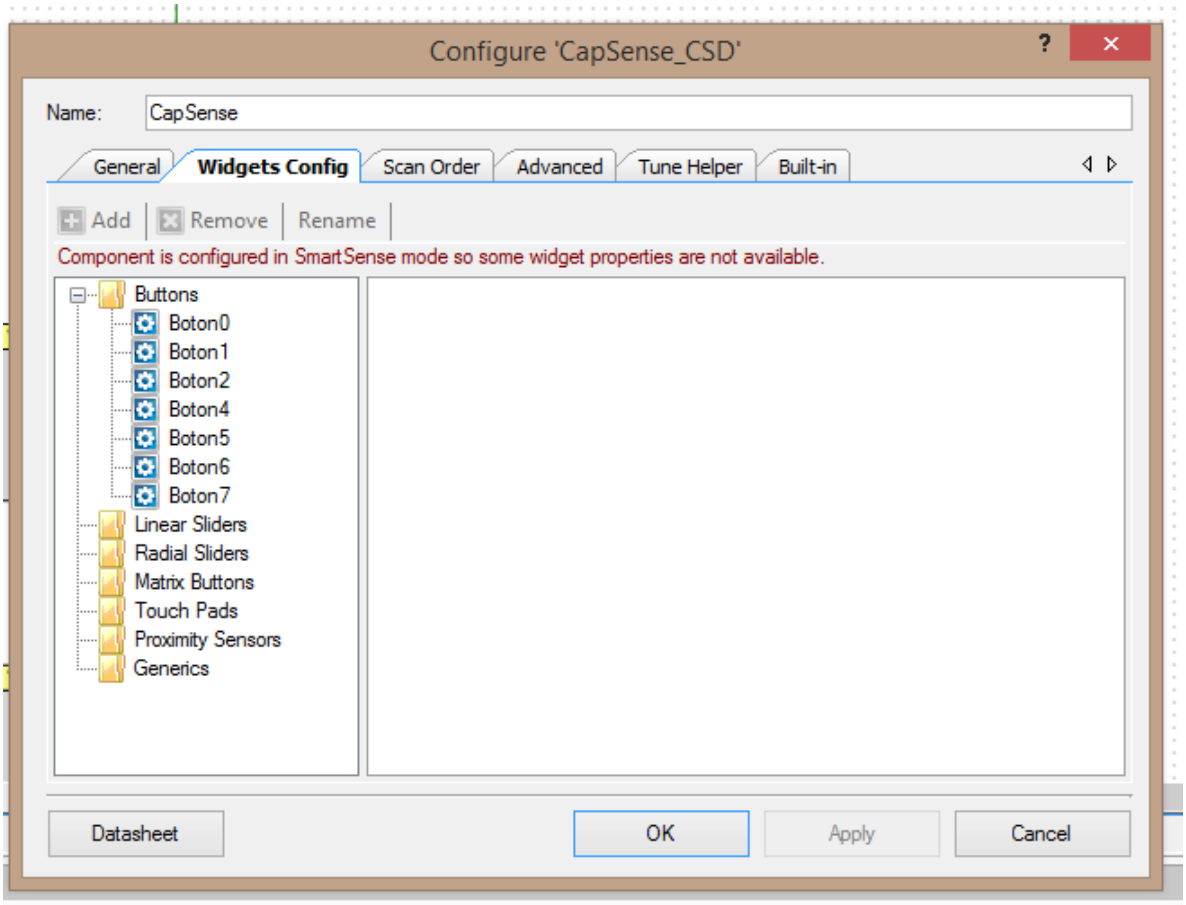
32.- Agregando un bloque Capsense a nuestro diseño

Después de esto procedemos a su configuración. En la primera ventana podemos encontrar parámetros como el método de afinación de este dispositivo así como la selección del tipo de filtro para eliminar el ruido y la velocidad de reloj de este bloque, dejaremos todos estos parámetros configurados como se muestra a continuación.



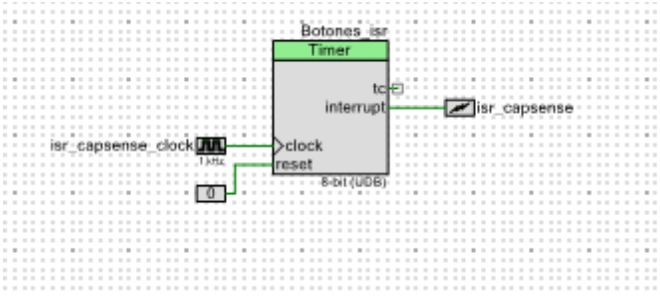
33.- Parámetros de configuración generales del componente Capsense

La siguiente ventana nos permite elegir de entre una serie de componentes como botones, sliders etc. Nosotros solo utilizaremos botones y quedaran configurados de la siguiente forma.



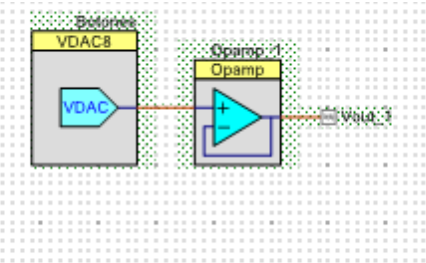
34.- Configuración de Widgets del bloque Capsense

Las siguientes ventanas las dejaremos con su configuración por defecto. Una vez que se ha agregado el bloque del Capsense agregaremos un timer que generara una interrupción a 1khz esta interrupción se encargara de escanear los botones y detectar si alguno fue tocado.



35.- Bloque Timer para escanear botones

Después de esto agregamos un bloque DAC que se encargara de generar determinados voltajes cada vez que se presione un botón estos voltajes serán interpretados por el ADC de nuestra tarjeta Teensy 3.5 y se encargaran de el control de la reproducción y la selección de las melodías.



36.- DAC encargado de la generación de voltajes para el control del Menú en la tarjeta Teensy

Una vez que se han configurado los bloques de Hardware que se usaran para el control de nuestro Menú dentro de nuestra tarjeta Teensy se procede a la programación del software para su funcionamiento.

Primero programamos la rutina de interrupción que escanea los botones del Capsense:

```
CY_ISR (isr_botones){
 CapSense_UpdateEnabledBaselines();
        CapSense_ScanEnabledWidgets();
        if (CapSense_CheckIsWidgetActive(CapSense_BOTON0__BTN)) {
            Led_Write(1);
            boton='s';
            boton_flag=true;
        } else if (CapSense_CheckIsWidgetActive(CapSense_BOTON1__BTN)) {
            Led_Write(1);
            boton='u';
            boton_flag=true;
        }else if (CapSense_CheckIsWidgetActive(CapSense_BOTON2__BTN)) {
            Led_Write(1);
            boton='d';
            boton_flag=true;
        }else if (CapSense_CheckIsWidgetActive(CapSense_BOTON4__BTN)) {
            Led_Write(1);
            boton='l';
            boton_flag=true;
        }
        else if (CapSense_CheckIsWidgetActive(CapSense_BOTON5__BTN)) {
            Led_Write(1);
            boton='r';
            boton_flag=true;
        } else if (CapSense_CheckIsWidgetActive(CapSense_BOTON6__BTN)) {
            Led_Write(1);
            Pin_1_Write(1);
            boton_flag=true;

        }else if (CapSense_CheckIsWidgetActive(CapSense_BOTON7__BTN)) {
            Led_Write(1);
            Pin_2_Write(1);
            boton_flag=true;
```

```
        }else{
            Led_Write(0);
            boton='p';
            Pin_1_Write(0);
            Pin_2_Write(0);
            boton_flag=false;
        }
```

Como se puede observar en esta rutina de interrupción haciendo uso de las APIS proporcionadas por el fabricante lo que se hace es verificar que botón fue presionado, asignar a una variable un carácter que representa dicho botón y activar una bandera para posteriormente en el main verificar que botón es el que fue presionado. Así mismo se prende el led integrado en la tarjeta para verificar que nuestra rutina de interrupción este siendo ejecutada correctamente. El código para comprobar los Botones y generar el voltaje que después será interpretado por nuestro arduino se muestra a continuación.

```
if(boton_flag==true){
   switch(boton)
    {
        case ('s'|'S'):
         Botones_SetValue(185);
         CyDelay(5);
         Botones_SetValue(255);

         break;
        case ('d'|'D'):
         Botones_SetValue(82);
         CyDelay(5);
         Botones_SetValue(255);

         break;
        case ('r'|'R'):
         Botones_SetValue(0);
         CyDelay(20);
         Botones_SetValue(255);

         break;
        case ('l'|'L'):
         Botones_SetValue(126);
         CyDelay(5);
         Botones_SetValue(255);

         break;
        case ('u'|'U'):
         Botones_SetValue(36);
         CyDelay(5);
         Botones_SetValue(255);

         break;
        default:
        Botones_SetValue(255);
        Led_Write(0);
        boton_flag=false;
    }
```

```
}
```

Como se puede observar este código verifica si la bandera que indica que un botón está presionado está encendida, entonces verifica que botón es el que fue presionado y escribe en el DAC el valor que nuestra Tarjeta Teensy por medio del ADC interpretara como un comando.

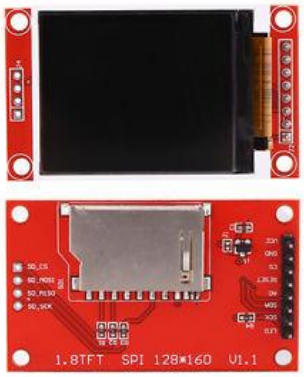
La librería MD_AButton incluida con la librería MD_MidiFile ambas opensource es la que nos permite hacer que la tarjeta Teensy interprete estos voltajes generados por nuestro DAC. Hubo que hacer algunas modificaciones a esta librería para que trabajara con nuestro hardware por lo que el código modificado de la librería se muestra a continuación.

```
#include <MD_AButton.h>

typedef struct
{
  uint16_t  adcThreshold; // Average analog value (PULLUP and no PULLUP)
  uint8_t   adcTolerance; // Valid range will be adcThreshold +/- Tolerance
  uint8_t   value;        // value returned to the caller
} TKEYDEF;

TKEYDEF keyDefTable[] =
{
      { 10, 100, 'R' },  // Right
      { 180, 100, 'U' },  // Up
      { 412, 100, 'D' },  // Down
      { 625, 100, 'L' },  // Left
      { 917, 80, 'S' },  // Select
      { 1025, 0, KEY_NONE }, // No Key
};

void MD_AButton::setKeyId(uint8_t id, char c)
{
  if ((id >= 0) && (id < ARRAY_SIZE(keyDefTable)))
  {
    keyDefTable[id].value = c;
  }
}

char MD_AButton::getKey(void)
// Convert ADC value to character
// return the key code defined in the keyDefTable
// Only read a key every _timeDetect ms. Effectively debounces.
// If a key is kept pressed, auto repeat every 300 milliseconds.
{
  int   input = analogRead(_keyPin);
  char  key = KEY_NONE;

  if (millis()-_lastReadTime > _timeDetect)
  {
    _lastReadTime = millis();

    // get the analog value and work out what key it is
    for (int k = 0; k < ARRAY_SIZE(keyDefTable); k++)
    {
```

```
      if ((input >= (keyDefTable[k].adcThreshold - keyDefTable[k].adcTolerance)) &&
        (input <= (keyDefTable[k].adcThreshold + keyDefTable[k].adcTolerance)))
      {
        key = keyDefTable[k].value;
        break;
      }
    }

    // check if this is the same as before and do auto repeat if timing is right
    if ((key == _lastKey) && (_lastKey != KEY_NONE))
    {
      if (millis() - _lastKeyTime < _timeRepeat)
        // don't repeat yet
        key = KEY_NONE;
      else
        // remember this repeat time for next time
        _lastKeyTime = millis();
    }
    else
    {
      // save the key for next time
      _lastKey = key;
      _lastKeyTime = millis();
    }
  }

  return(key);
}
```

Ahora que tenemos solucionada la parte del control del Menú de nuestra tarjeta Teensy procedemos a la codificación para el despliegue de este.

Para el Hardware final de nuestro proyecto se decidió sustituir la pantalla Oled anteriormente usada por una pantalla TFT mas grande la pantalla elegida fue la siguiente.



37.- Pantalla TFT 1.8 pulgadas

Una vez cableada la pantalla se procedió a probar un código de ejemplo para el uso de esta pantalla. Se comprobó que la pantalla funcionaba correctamente y se procedió a la modificación del código que se tenía hasta el momento, para que funcione con esta nueva pantalla.

El código de la tarjeta Teensy entonces quedo de la siguiente forma:

```
#include <SdFat.h>
#include <MD_MIDIFile.h>
#include <MD_AButton.h>
#include <stdio.h>

#include "FSMtypes.h"          // FSM enumerated types
#define  LCD_KEYS  KEY_ADC_PORT
#define DEBUG_MODE      0

#if DEBUG_MODE
#define  LCD_KEYS  KEY_ADC_PORT
#define DEBUG(x)        Serial.print(x)
#define DEBUGX(x)       Serial.print(x, HEX)
#define SERIAL_RATE     115200
#else

#define DEBUG(x)
#define DEBUGX(x)
#define SERIAL_RATE     115200

#endif
#define TFT_SCLK 13  // SCLK can also use pin 14
#define TFT_MOSI 11  // MOSI can also use pin 7
#define TFT_CS   10  // CS & DC can use pins 2, 6, 9, 10, 15, 20, 21, 22,
23
#define TFT_DC    9  //  but certain pairs must NOT be used: 2+10, 6+9,
20+23, 21+22
#define TFT_RST   8  // RST can use any pin
#define SD_CS     4  // CS for SD card, can use any pin
#define  BLACK           0x0000
//librerias para la pantalla
#include <Adafruit_GFX.h>    // Core graphics library
#include <Fonts/FreeSans9pt7b.h>
#include <Fonts/Digital.h>
#include <Fonts/Manga.h>
#include <ST7735_t3.h> // Hardware-specific library
#include <SPI.h>
//Librerias para el reloj///////////////
#include <Wire.h>
#include <TimeLib.h>
#include <DS1307RTC.h>
////////////////////////////////////////
// Option 1: use any pins but a little slower
ST7735_t3 display = ST7735_t3(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK,
TFT_RST);
// SD Hardware defines ---------
// SPI select pin for SD card (SPI comms).
// Arduino Ethernet shield, pin 4.
// Default SD chip select is the SPI SS pin (10).
// Other hardware will be different as documented for that hardware.
#define  SD_SELECT  4
```

```
int c;
// LCD display defines ---------

// LCD user defined characters
char str[100],str2[100];
int flag=0;


SdFatSdio  SD;
MD_MIDIFile SMF;
MD_AButton  LCDKey(LCD_KEYS);

// Playlist handling -----------
#define FNAME_SIZE             30                              // 8.3 +
'\0' character file names
#define PLAYLIST_FILE  "PLAYLIST.TXT" // file of file names
#define MIDI_EXT               ".mid"                  // MIDI file
extension
uint16_t       plCount = 0;
// volumen
const int slave_Select_Pin  = 31;
volatile int       level1            = 255;
volatile int       level2            = 255;
const int BUTTON1 = 2,BUTTON2 = 3;
volatile boolean flagB1;
//
void MSP42010PotWrite(int slaveSelectPin, byte address, int value) {
     // take the SS pin low to select the chip:
     digitalWrite(slaveSelectPin,LOW);
     //  send in the address and value via SPI:
     SPI1.transfer(address);
     SPI1.transfer(value);
     // take the SS pin high to de-select the chip:
     digitalWrite(slaveSelectPin,HIGH);
}
void isrButton1 ()
{
  noInterrupts();
  if(level1<255){
  level1=level1+5;
  flagB1 = true;
  MSP42010PotWrite(slave_Select_Pin, B00010001, level1);
  MSP42010PotWrite(slave_Select_Pin, B00010010, level1);

  }
  else{
  }
  interrupts();
  }

void isrButton2 ()
{
  noInterrupts();
  if(level1>0){
  level1=level1-5;
  flagB1 = true;
  MSP42010PotWrite(slave_Select_Pin, B00010001, level1);
```

```
  MSP42010PotWrite(slave_Select_Pin, B00010010, level1);
  }
  else{}
   interrupts();
}
// MIDI callback functions for MIDIFile library ---------------
IntervalTimer UpdateClock;
void Update_Clock(){
    tmElements_t tm;

  if (RTC.read(tm)) {
     if(tm.Second<10)
    sprintf(str,"%2d:%2d:0%1d",tm.Hour,tm.Minute,tm.Second);
    else
    sprintf(str,"%2d:%2d:%2d",tm.Hour,tm.Minute,tm.Second);
    display.setCursor(10,90);
    display.fillRect(10,80, 120, 30, BLACK);
    display.setFont(&Digital12pt7b);
    display.println(str);
    display.setFont();
    flag=0;
  } else {
    if (RTC.chipPresent()) {
      Serial.println("The DS1307 is stopped.  Please run the SetTime");
      Serial.println("example to initialize the time and begin
running.");
      Serial.println();
    } else {
      Serial.println("DS1307 read error!  Please check the circuitry.");
      Serial.println();
    }
    delay(9000);
  }
  }
void midiCallback(midi_event *pev)
// Called by the MIDIFile library when a file event needs to be processed
// thru the midi communications interface.
// This callback is set up in the setup() function.
{
#if !DEBUG_MODE
        if ((pev->data[0] >= 0x80) && (pev->data[0] <= 0x90))
        {
                Serial1.write(pev->data[0] | pev->channel);
                Serial1.write(&pev->data[1], pev->size-1);
        }

#endif
  DEBUG("\nM T");
  DEBUG(pev->track);
  DEBUG(":  Ch ");
  DEBUG(pev->channel+1);
  DEBUG(" Data ");
  for (uint8_t i=0; i<=pev->size; i++)
  {
        DEBUGX(pev->data[i]);
    DEBUG(' ');
  }
```

```
}



void midiSilence(void)
// Turn everything off on every channel.
// Some midi files are badly behaved and leave notes hanging, so between
songs turn
// off all the notes and sound
{
        midi_event      ev;

        // All sound off
        // When All Sound Off is received all oscillators will turn off,
and their volume
        // envelopes are set to zero as soon as possible.
        ev.size = 0;
        ev.data[ev.size++] = 0xb0;
        ev.data[ev.size++] = 120;
        ev.data[ev.size++] = 0;

        for (ev.channel = 0; ev.channel < 16; ev.channel++)
                midiCallback(&ev);
}

// LCD Message Helper functions -----------------
void LCDMessage( const char *msg, bool clrEol = false)
// Display a message on the LCD screen with optional spaces padding the
end
{

        Serial.print(msg);
        if (clrEol)
        {

                        Serial.write(' ');
        }
}

void LCDErrMessage(const char *msg, bool fStop)
{
        LCDMessage(msg, true);
  DEBUG("\nLCDErr: ");
  DEBUG(msg);
        while (fStop) ;         // stop here if told to
        delay(2000);            // if not stop, pause to show message
}

// Create list of files for menu --------------

uint16_t createPlaylistFile(void)
// create a play list file on the SD card with the names of the files.
// This will then be used in the menu.
{
        SdFile          plFile;         // play list file
        SdFile          mFile;          // MIDI file
        uint16_t        count = 0;// count of files
```

```
        char              fname[FNAME_SIZE];

        // open/create the play list file
        if (!plFile.open(PLAYLIST_FILE, O_CREAT|O_WRITE))
                LCDErrMessage("PL create fail", true);

  SD.vwd()->rewind();
  while (mFile.openNext(SD.vwd(), O_READ))
        {
    mFile.getName(fname, FNAME_SIZE);

    DEBUG("\nFile ");
    DEBUG(count);
    DEBUG(" ");
    DEBUG(fname);

                if (mFile.isFile())
                {
                        if (strcmp(MIDI_EXT, &fname[strlen(fname)-
strlen(MIDI_EXT)]) == 0)
                        // only include files with MIDI extension
                        {
                                plFile.write(fname, FNAME_SIZE);
                                count++;
                        }
                }
                mFile.close();
        }
  DEBUG("\nList completed");

        // close the play list file
        plFile.close();

        return(count);
}

// FINITE STATE MACHINES ------------------------------

seq_state lcdFSM(seq_state curSS)
// Handle selecting a file name from the list (user input)
{
        static lcd_state s = LSBegin;
        static uint8_t plIndex = 0;
        static char    fname[FNAME_SIZE];
        static SdFile  plFile;        // play list file

        // LCD state machine
        switch (s)
        {
        case LSBegin:
    flag=0;
    display.fillRect(0,30, 160, 120, BLACK);
                LCDMessage("Select music:", true);
    display.setCursor(0,30);
    display.print("selecciona tu ");
    display.setCursor(0,40);
    display.print("musica: ");
```

```
                if (!plFile.isOpen())
                {
                        if (!plFile.open(PLAYLIST_FILE, O_READ))
                                LCDErrMessage("PL file no open", true);
                }
                s = LSShowFile;
                break;

        case LSShowFile:
                plFile.seekSet(FNAME_SIZE*plIndex);
                plFile.read(fname, FNAME_SIZE);

                LCDMessage( fname, true);
    display.setCursor(0,55);
    display.fillRect(0,55, 160, 20, BLACK);
    display.setCursor(0,55);
    display.print(fname);
                Serial.print(plIndex == 0 ? ' ' : '<');
                Serial.print(plIndex == plCount-1 ? ' ' : '>');
                s = LSSelect;
                break;

        case LSSelect:
                switch (LCDKey.getKey())
                // Keys are mapped as follows:
                // Select:      move on to the next state in the state
machine
                // Left:        use the previous file name (move back one
file name)
                // Right:       use the next file name (move forward one
file name)
                // Up:          move to the first file name
                // Down:        move to the last file name
                {
                        case 'S':       // Select
                                s = LSGotFile;
                                break;

                        case 'L':       // Left
                                if (plIndex != 0)
                                        plIndex--;
                                s = LSShowFile;

                                break;

                        case 'U':       // Up
                                plIndex = 0;
                                s = LSShowFile;

                                break;

                        case 'D':       // Down
                                plIndex = plCount-1;
                                s = LSShowFile;

                                break;
```

```
                        case 'R':       // Right
                                if (plIndex != plCount-1)
                                        plIndex++;
                                s = LSShowFile;
                                break;

                }
                break;

        case LSGotFile:
                // copy the file name and switch mode to playing MIDI
                SMF.setFilename(fname);
                curSS = MIDISeq;
                // fall through to default state

        default:
                s = LSBegin;
                break;
        }

        return(curSS);
}

seq_state midiFSM(seq_state curSS)
// Handle playing the selected MIDI file
{
        static midi_state s = MSBegin;

        switch (s)
        {
        case MSBegin:
                // Set up the LCD
                LCDMessage(SMF.getFilename(), true);
                LCDMessage( "K  \xdb  \1  >", true);          // string of
user defined characters
                s = MSLoad;
                break;

        case MSLoad:
                // Load the current file in preparation for playing
                {
                        int  err;

                        // Attempt to load the file
                        if ((err = SMF.load()) == -1)
                        {
                                s = MSProcess;
                        }
                        else
                        {
                                char    aErr[16];

                                sprintf(aErr, "SMF error %03d", err);
                                LCDErrMessage(aErr, false);
                                s = MSClose;
                        }
                }
```

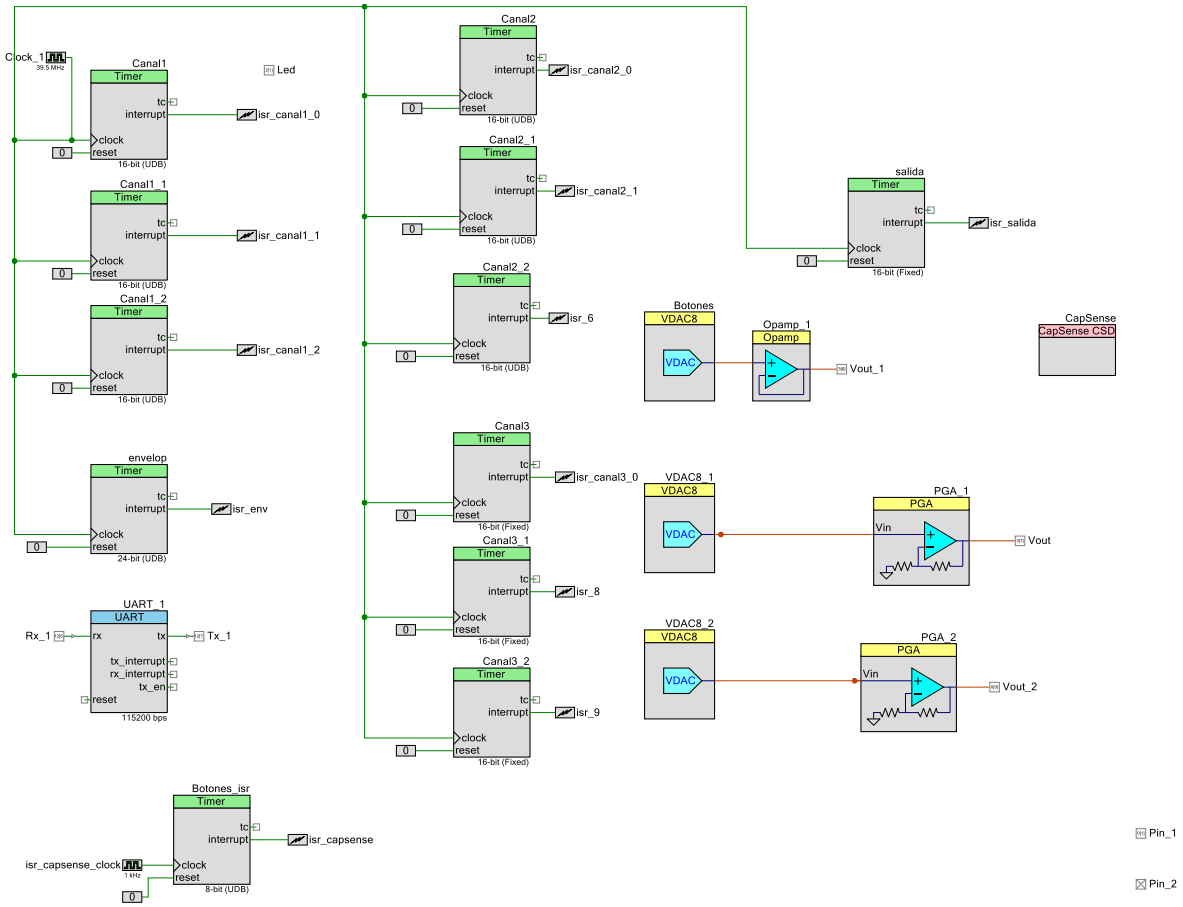
```
                break;

        case MSProcess:
                // Play the MIDI file
                if (!SMF.isEOF())
    {
                        if (SMF.getNextEvent())
      {
        char  sBuf[10];

        sprintf(sBuf, "%3d", SMF.getTempo());
        LCDMessage( sBuf, true);
        sprintf(sBuf, "%d/%d", SMF.getTimeSignature()>>8,
SMF.getTimeSignature() & 0xf);
        LCDMessage( sBuf, true);
        if(flag==0){
        display.fillRect(0,30, 160, 50, BLACK);
        sprintf(str," playing ....  Tempo:%3d    Compas:%s
",SMF.getTempo(),sBuf);
        display.setCursor(0,40);
        display.println(str);
        flag=1;
        }
      };
    }
                else
                        s = MSClose;



                // check the keys
                switch (LCDKey.getKey())
                {
                        case 'S':       SMF.restart();          break;  //
Rewind
                        case 'L':       s = MSClose;              break;        // Stop
                        case 'U':       SMF.pause(true);        break;  //
Pause
                        case 'D':       SMF.pause(false);       break;  //
Start
                        case 'R':                                                       brea
                }

                break;

        case MSClose:
                // close the file and switch mode to user input
                SMF.close();
                midiSilence();
                curSS = LCDSeq;
                // fall through to default state

        default:
                s = MSBegin;
                break;
        }
```

```
        return(curSS);
}

void setup(void)
{
        int  err;
  pinMode(TFT_CS, OUTPUT);
  digitalWrite(TFT_CS, HIGH);
        // initialise MIDI output stream

  display.initR(INITR_BLACKTAB);
  display.setRotation(3);
        display.fillScreen(BLACK);
  display.setTextColor(0x101F);
  display.setFont(&Manga8pt7b);
  display.setCursor(0,22);
  display.println("Digital Music Box:");

  display.setFont();

        LCDMessage( "  Midi  Player  ", false);
        LCDMessage( "  ------------  ", false);

        // Load characters to the LCD

        pinMode(LCD_KEYS, INPUT);

        // initialise SDFat
        if (!SD.begin()){
                LCDErrMessage("SD init fail!", true);
    display.fillScreen(BLACK);
     display.setCursor(0,40);
     display.print("Error en la SD");
        }
        plCount = createPlaylistFile();
        if (plCount == 0)
                LCDErrMessage("No files", true);

        // initialise MIDIFile
        SMF.begin(&SD);
        SMF.setMidiHandler(midiCallback);

  interrupts();
  UpdateClock.begin(Update_Clock,1000000);
     pinMode (slave_Select_Pin, OUTPUT);
     SPI1.setMOSI(21);
     // initialize SPI:
     SPI1.begin();
     MSP42010PotWrite(slave_Select_Pin, B00010001, level1);
     MSP42010PotWrite(slave_Select_Pin, B00010010, level2);
     pinMode(BUTTON1, INPUT);
     attachInterrupt (BUTTON1, isrButton1, RISING);
     attachInterrupt (BUTTON2, isrButton2, RISING);
      Serial.begin(SERIAL_RATE);
  Serial1.begin(SERIAL_RATE);
        delay(750);            // allow the welcome to be read on the LCD
```

```
}

void loop(void)
// only need to look after 2 things - the user interface (LCD_FSM)
// and the MIDI playing (MIDI_FSM). While playing we have a different
// mode from choosing the file, so the FSM will run alternately,
depending
// on which state we are currently in.
{
    if (flagB1 == true)
  {
    Serial.println("BUTTON 1 interrupt has occurred");
    sprintf(str2,"nivel: %d",level1);
    Serial.println(str2);
     flagB1=false;
  }

        static seq_state       s = LCDSeq;

        switch (s)
        {
                case LCDSeq:  s = lcdFSM(s);   break;
                case MIDISeq: s = midiFSM(s);  break;
                default: s = LCDSeq;
        }
}
```

Se cargo el codigo en la tarjeta y se procedio a verificar su funcionamiento. El codigo funciono perfectamente como se puede ver en la siguiente imagen.

Posteriormente se procedio a la configuracion del hardware final dentro del Psoc el diagrama entonces quedo de la siguiente manera.



Y el codigo final entonces es el siguiente:

```
#include "project.h"
#include <math.h>
#include <stdio.h>
#include <stdbool.h>
#define TABLE_LENGTH 64
#define msPerTick 1
#define reloj 39500000

void tone(int canal, float  frecuencia);
void noTone(int canal);
void delay (uint32 ms);
volatile int count=0,count2=0,count3=0,count4=0,count5=0;
volatile int count_canal1[3]={0,0,0};
volatile int
env_index_canal1[3]={0,0,0},env_index_canal2[3]={0,0,0},env_index_canal3[
3]={0,0,0};
volatile char boton;
int poly=0,poly2=0;
volatile uint8 result=0,result2=0;
```

```
char str[100],str2[3];
int i=0,j=0;
int canal=0;
volatile int
acc_canal1[3]={0,0,0},acc_canal2[3]={0,0,0},acc_canal3[3]={0,0,0};
float freq;
_Bool boton_flag=0;

volatile uint8 data[3],recibir;
CYCODE const uint8 sineTable[TABLE_LENGTH] = {
133,103,83,73,48,23,23,20,8,5,23,33,35,55,88,108,123,158,188,200,213,235,
245,240,250,255,248,235,235,220,188,160,133,103,83,73,48,23,23,20,8,5,
2,3,33,35,55,88,108,123,158,188,200,213,235,245,240,250,255,248,235,235,2
20,188
};


CYCODE const uint8 sineTable2[TABLE_LENGTH] = {
142,121,115,92,55,46,89,137,138,128,133,136,126,124,129,139,139,106,76,67
,52,25,2,27,71,79,76,88,98,100,102,113,141,167,157,132,114,100,71,41,50,
84,106,120,138,151,154,155,157,180,201,185,155,137,132,108,63,46,64,79,89
,97,101,107
};
CYCODE const uint8 sineTable3[TABLE_LENGTH] = {
121,132,140,139,137,131,123,122,128,136,142,142,138,134,130,123,120,129,1
39,150,161,166,166,162,155,154,162,173,181,186,184,177,170,159,148,146,
149,155,161,161,159,156,149,144,148,159,170,181,188,189,187,179,170,168,1
71,177,181,180,172,164,154,140,130,128
};
CYCODE const uint8 sineTable4[TABLE_LENGTH] = {
113,133,158,180,191,193,187,174,155,136,121,111,107,103,98,93,89,91,98,10
9,125,141,157,167,173,174,172,166,159,150,142,135,129,123,119,117,117,117
,
118,119,121,124,126,127,129,131,131,131,129,128,128,126,125,123,122,121,1
21,122,125,129,134,138,140,139
};

CYCODE const uint8 envelope[TABLE_LENGTH*2] = {

255, 250, 245, 240, 235, 231, 226, 222, 217, 213, 209, 205, 201, 197,
193, 189, 185, 182, 178, 174, 171, 168,
    164, 161, 158, 155, 152, 149, 146, 143, 140, 137, 134, 132, 129, 127,
124, 122, 119, 117, 115, 112, 110,
    108, 106, 104, 102, 100,  98,  96,  94,  92,  90,  88,  87,  85,  83,
82,  80,  78,  77,  75,  74,  72,
    71,  69,  68,  67,  65,  64,  63,  62,  60,  59,  58,  57,  56,  55,
54,  53,  51,  50,  49,  48,  48,
    47,  46,  45,  44,  43,  42,  41,  40,  39,  38,  37,  35,  34,  33,
32,  31,  30,  29,  28,  26,  25,
    24,  23,  22,  21,  20,  19,  17,  16,  15,  14,  13,  12,  11,  10,
8,   7,   6,   5,   4,   3,   2,   0

};
CYCODE const float notes[TABLE_LENGTH*2] = {

8.1757994,8.6619568,9.1770239,9.7227182,10.300861,10.913383,11.562325,12.
249857,12.978271,13.750000,14.567617,15.433853,
```

```
16.351599,17.323914,18.354048,19.445436,20.601723,21.826765,23.124651,24.
499714,25.956543,27.500000,29.135235,30.867706,32.703197,
34.647827,36.708096,38.890873,41.203445,43.653530,46.249302,48.999428,51.
913086,55,58.270470,61.735413,65.406395,69.295654,73.416191,
77.781746,82.406891,87.307060,92.498604,97.998856,103.82617,110,116.54094
,123.47083,130.81279,138.59131,146.83238,155.56349,164.81378,
174.61412,184.99721,195.99771,207.65234,220,233.08188,246.94165,261.62558
,277.18262,293.66476,311.12698,329.62756,349.22824,369.99442,
391.99542,415.30469,440,466.16376,493.88330,523.25116,554.36523,587.32953
,622.25397,659.25513,698.45648,739.98883,783.99084,830.60938,880,
932.32751,987.76660,1046.5023,1108.7305,1174.6591,1244.5079,1318.5103,139
6.9130,1479.9777,1567.9817,1661.2188,1760,1864.6550,1975.5332,
2093.0046,2217.4609,2349.3181,2489.0159,2637.0205,2793.8259,2959.9553,313
5.9634,3322.4375,3520,3729.3101,3951.0664,4186.0093,4434.9219,
4698.6362,4978.0317,5274.0410,5587.6519,5919.9106,6271.9268,6644.8750,704
0,7458.6201,7902.1328,8372.0186,8869.8438,9397.2725,9956.0635,
10548.082,11175.304,11839.821,12543.854
};

CY_ISR(int_canal1_0){

  if(env_index_canal1[0]<=127){

     if(count_canal1[0]<=TABLE_LENGTH){

acc_canal1[0]=(sineTable[count_canal1[0]]*envelope[env_index_canal1[0]])/
255;



        count_canal1[0]++;
    }else{

         count_canal1[0]=0;
    }
   }else{
         acc_canal1[0]=0;

        }

    Canal1_ReadStatusRegister();
}
CY_ISR(int_canal2_0){

    if(env_index_canal2[0]<=127){

     if(count2<=TABLE_LENGTH){

acc_canal2[0]=((sineTable[count2]*envelope[env_index_canal2[0]])/255);

        count2++;
    }else{

         count2=0;
    }
   }else{
     acc_canal2[0]=0;
```

```
        }

   Canal2_ReadStatusRegister();
}
CY_ISR(int_canal3_0){

  if(env_index_canal3[0]<=127){

     if(count3<=TABLE_LENGTH){
        acc_canal3[0]=(sineTable[count3]*envelope[env_index_canal3[0]]);
       acc_canal3[0]=(acc_canal3[0])/255;



        count3++;
    }else{

         count3=0;
    }
   }else{

       acc_canal3[0]=0;
        }

    Canal1_ReadStatusRegister();
}
CY_ISR(int_canal1_1){

  if(env_index_canal1[1]<=127){

     if(count_canal1[1]<=TABLE_LENGTH){

acc_canal1[1]=(sineTable[count_canal1[1]]*envelope[env_index_canal1[1]])/
255;




        count_canal1[1]++;
    }else{

         count_canal1[1]=0;
    }
   }else{
         acc_canal1[1]=0;

        }

    Canal1_1_ReadStatusRegister();
}
CY_ISR(int_canal2_1){

  if(env_index_canal2[1]<=127){

     if(count5<=TABLE_LENGTH){
```

```
acc_canal2[1]=((sineTable[count5]*envelope[env_index_canal2[1]])/255);



        count5++;
    }else{

         count5=0;
    }
   }else{
         acc_canal2[1]=0;

        }

    Canal2_1_ReadStatusRegister();
}
CY_ISR(int_canal1_2){

  if(env_index_canal1[2]<=127){

     if(count_canal1[2]<=TABLE_LENGTH){

acc_canal1[2]=(sineTable[count_canal1[2]]*envelope[env_index_canal1[2]]);
       acc_canal1[2]= (acc_canal1[2])/255;



       count_canal1[2]++;
    }else{

         count_canal1[2]=0;
    }
   }else{
         acc_canal1[2]=0;

        }

    Canal2_1_ReadStatusRegister();
}

CY_ISR(env_update){

    env_index_canal1[0]++;
    env_index_canal2[0]++;
    env_index_canal3[0]++;
    env_index_canal1[1]++;
    env_index_canal2[1]++;
    env_index_canal3[1]++;
    env_index_canal1[2]++;
    env_index_canal2[2]++;
    env_index_canal3[2]++;
envelop_ReadStatusRegister();
}
CY_ISR(DAC){
    result=(acc_canal1[0]+acc_canal1[1]+acc_canal1[2])/3;
```

```
    result2=((acc_canal2[1]+acc_canal2[0]+acc_canal3[0])/3);
    VDAC8_2_SetValue(result);
    VDAC8_1_SetValue(result2);
    salida_ReadStatusRegister();
}
CY_ISR (isr_botones){
 CapSense_UpdateEnabledBaselines();
        CapSense_ScanEnabledWidgets();
        if (CapSense_CheckIsWidgetActive(CapSense_BOTON0__BTN)) {
            Led_Write(1);
            boton='s';
            boton_flag=true;
        } else if (CapSense_CheckIsWidgetActive(CapSense_BOTON1__BTN)) {
            Led_Write(1);
            boton='u';
            boton_flag=true;
        }else if (CapSense_CheckIsWidgetActive(CapSense_BOTON2__BTN)) {
            Led_Write(1);
            boton='d';
            boton_flag=true;
        }else if (CapSense_CheckIsWidgetActive(CapSense_BOTON4__BTN)) {
            Led_Write(1);
            boton='l';
            boton_flag=true;
        }
        else if (CapSense_CheckIsWidgetActive(CapSense_BOTON5__BTN)) {
            Led_Write(1);
            boton='r';
            boton_flag=true;
        } else if (CapSense_CheckIsWidgetActive(CapSense_BOTON6__BTN)) {
            Led_Write(1);
            Pin_1_Write(1);
            boton_flag=true;

        }else if (CapSense_CheckIsWidgetActive(CapSense_BOTON7__BTN)) {
            Led_Write(1);
            Pin_2_Write(1);
            boton_flag=true;

        }else{
            Led_Write(0);
            boton='p';
            Pin_1_Write(0);
            Pin_2_Write(0);
            boton_flag=false;
        }


   Botones_isr_ReadStatusRegister();
}
int main(void)
{
       CyGlobalIntEnable; /* Enable global interrupts. */
     asm (".global _scanf_float");
    //Opamp_1_Start();
    PGA_1_Start();
    PGA_2_Start();
```

```
    envelop_Start();
    VDAC8_1_Start();
    VDAC8_2_Start();
    //Opamp_3_Start();
    salida_Start();
    PGA_1_SetGain(2);
    PGA_2_SetGain(2);
    UART_1_Start();
    Botones_Start();
    Opamp_1_Start();
    CapSense_Start();
    Botones_isr_Start();
    isr_capsense_clock_Start();
    CapSense_InitializeAllBaselines();
    CapSense_ScanEnabledWidgets();

    /* Place your initialization/startup code here (e.g. MyInst_Start())
*/
    /*inicilizar la ejecucion de las interrupciones*/
    isr_canal1_0_StartEx(int_canal1_0);
    isr_canal2_0_StartEx(int_canal2_0);
    isr_canal3_0_StartEx(int_canal3_0);
    isr_canal1_1_StartEx(int_canal1_1);
    isr_canal2_1_StartEx(int_canal2_1);
    isr_salida_StartEx(DAC);
    isr_env_StartEx(env_update);
    isr_canal1_2_StartEx(int_canal1_2);
    isr_capsense_StartEx(isr_botones);


/*////////////////////////////////////////////////////////////////////*/
    Botones_SetValue(255);
    envelop_WritePeriod(400000);
    salida_WritePeriod(400);
    Botones_isr_WritePeriod(100);

    for(;;)
    {

    while(UART_1_GetRxBufferSize()>=1){
    recibir=UART_1_GetChar();
    UART_1_WriteTxData(recibir);

    if((recibir&0x0f0)==0x90){
        data[i]=recibir;

       i=1;
    }else{
        if(i==1){
        data[i]=recibir;
        canal=(data[i-1]&0x0f)+1;
        sprintf(str,"Note on  canal:%d",canal);
        UART_1_PutString(str);
        sprintf(str," nota midi %d ",data[i]);
        UART_1_PutString(str);
        freq=notes[data[i]];
        sprintf(str," frecuencia midi %f ",freq);
```

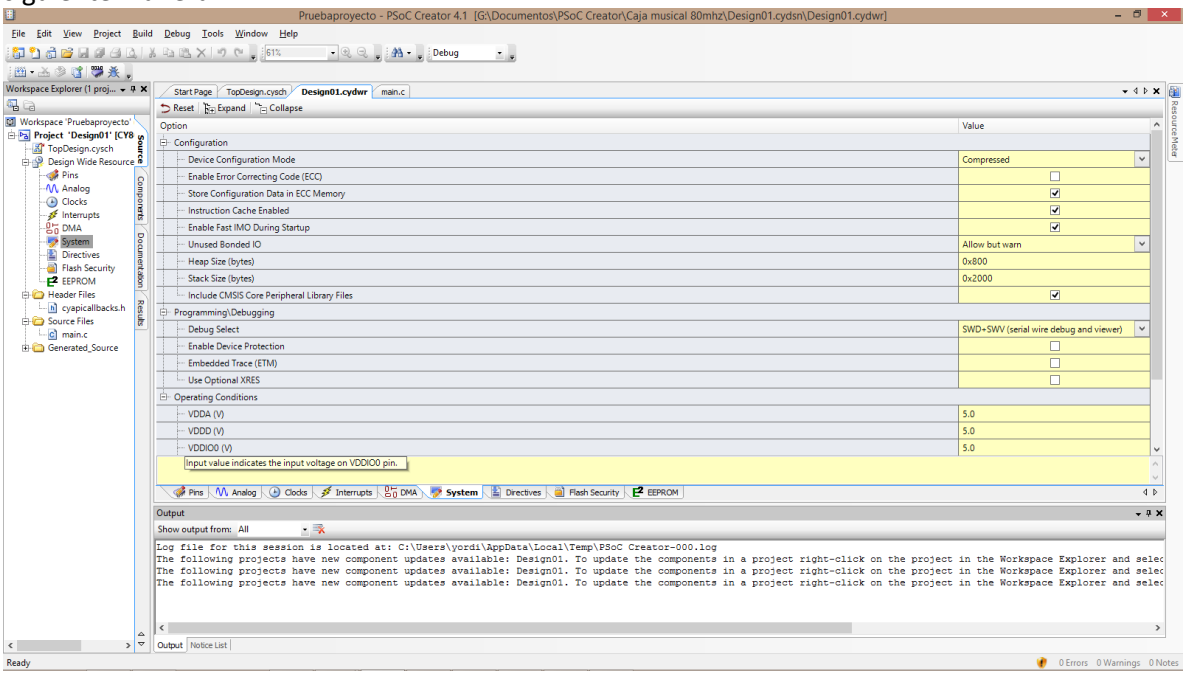
```
        UART_1_PutString(str);
        UART_1_PutCRLF(1);
        tone(canal,freq);
        i=0;
        }
   }
    if((recibir&0x0f0)==0x80){
        data[i]=recibir;

       j=1;
   }else{
        if(j==1){
        data[i]=recibir;
        canal=(data[i-1]&0x0f)+1;
        sprintf(str,"Note off  canal:%d",canal);
        UART_1_PutString(str);
        sprintf(str," nota midi %d ",data[i]);
        UART_1_PutString(str);
        UART_1_PutCRLF(1);
        noTone(canal);
        j=0;
        }
   }

}

 if(boton_flag==true){
 switch(boton)
  {
      case ('s'|'S'):
       Botones_SetValue(185);
       CyDelay(5);
       Botones_SetValue(255);

       break;
      case ('d'|'D'):
       Botones_SetValue(82);
       CyDelay(5);
       Botones_SetValue(255);

       break;
      case ('r'|'R'):
       Botones_SetValue(0);
       CyDelay(20);
       Botones_SetValue(255);

       break;
      case ('l'|'L'):
       Botones_SetValue(126);
       CyDelay(5);
       Botones_SetValue(255);

       break;
      case ('u'|'U'):
       Botones_SetValue(36);
       CyDelay(5);
       Botones_SetValue(255);
```

```
         break;
        default:
        Botones_SetValue(255);
        Led_Write(0);
        boton_flag=false;
    }

}




        /* Place your application code here. */
    }
}
void tone(int canal,float frecuencia){
    int periodo;
    if (canal==1){
        if (poly>=2)
        poly=0;

        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        switch(poly)
        {
        case 0:

        env_index_canal1[0]=0;
        count_canal1[0]=0;
        Canal1_WritePeriod(periodo);
        Canal1_Start();
        poly++;
        break;
        case 1:
        env_index_canal1[1]=0;
        count_canal1[1]=0;
        Canal1_1_WritePeriod(periodo);
        Canal1_1_Start();
        poly++;
        break;
        case 2:
        env_index_canal1[2]=0;
        count_canal1[2]=0;
        Canal1_2_WritePeriod(periodo);
        Canal1_2_Start();
        poly++;
        break;

        }
    }else if(canal==2){
           if (poly2>1)
           poly2=0;
        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        switch(poly2)
        {
```

```
        case 0:

        env_index_canal2[0]=0;
        count2=0;
        Canal2_WritePeriod(periodo);
        Canal2_Start();
        poly2++;
        break;
        case 1:
        env_index_canal2[1]=0;
        count5=0;
        Canal2_1_WritePeriod(periodo);
        Canal2_1_Start();
        poly2++;
        break;
        }
    }
    else if(canal==3){

        periodo=(reloj/frecuencia)/TABLE_LENGTH;
        env_index_canal3[0]=0;
        count3=0;
        Canal3_WritePeriod(periodo);
        Canal3_Start();

    }
}
void noTone(int canal){
   /* if(canal==1){

        switch(poly)
        {
        case 1:
        Canal1_2_Stop();
        poly--;
        break;
        case 2:
      Canal1_1_Stop();
        poly--;
        break;
      case 3:
      Canal1_Stop();
        poly--;
        break;
        }



    }else if(canal==2){
        switch(poly2)
        {
        case 1:
        Canal2_Stop();
        poly2--;
        break;
        case 2:
        Canal2_1_Stop();
```

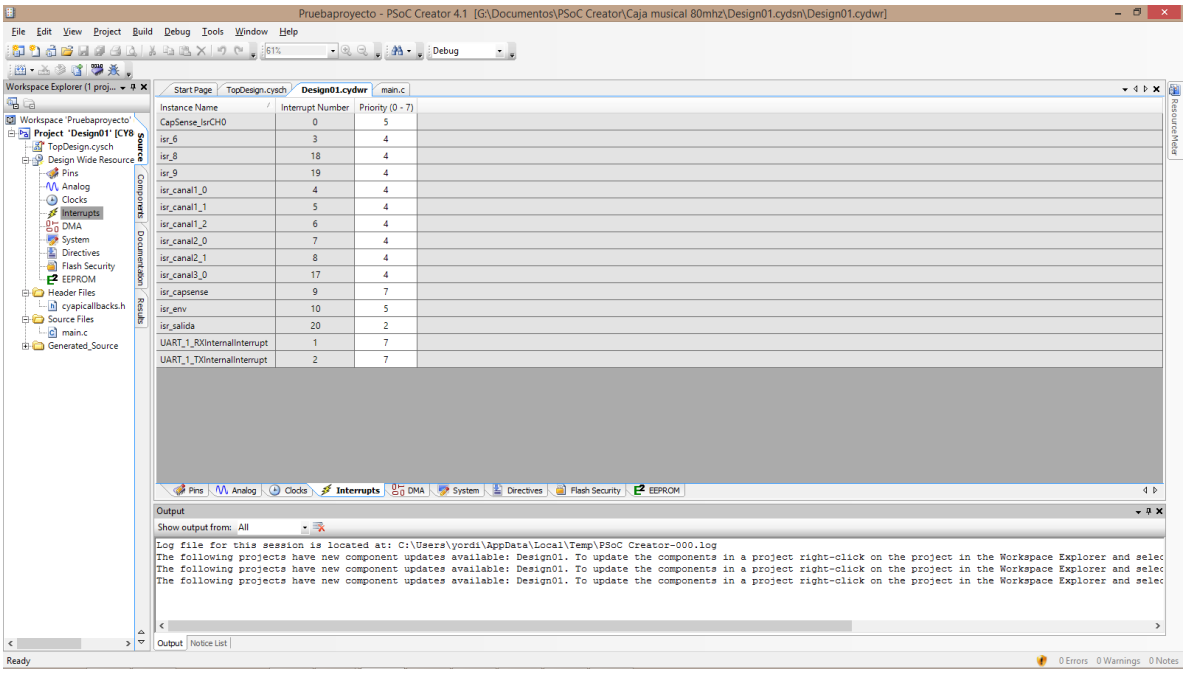
```
            poly2--;
            break;


            }


    }else if(canal==3){
        Canal3_Stop();
    }*/
  }
void delay(uint32 ms){
    CyDelay(ms);
}
```

Tambien se procedio a modificación del stack y el heap del microprocesador para permitir un mayor rendimiento esta configuracion se hace dentro de la ventana system y queda de la siguiente manera.



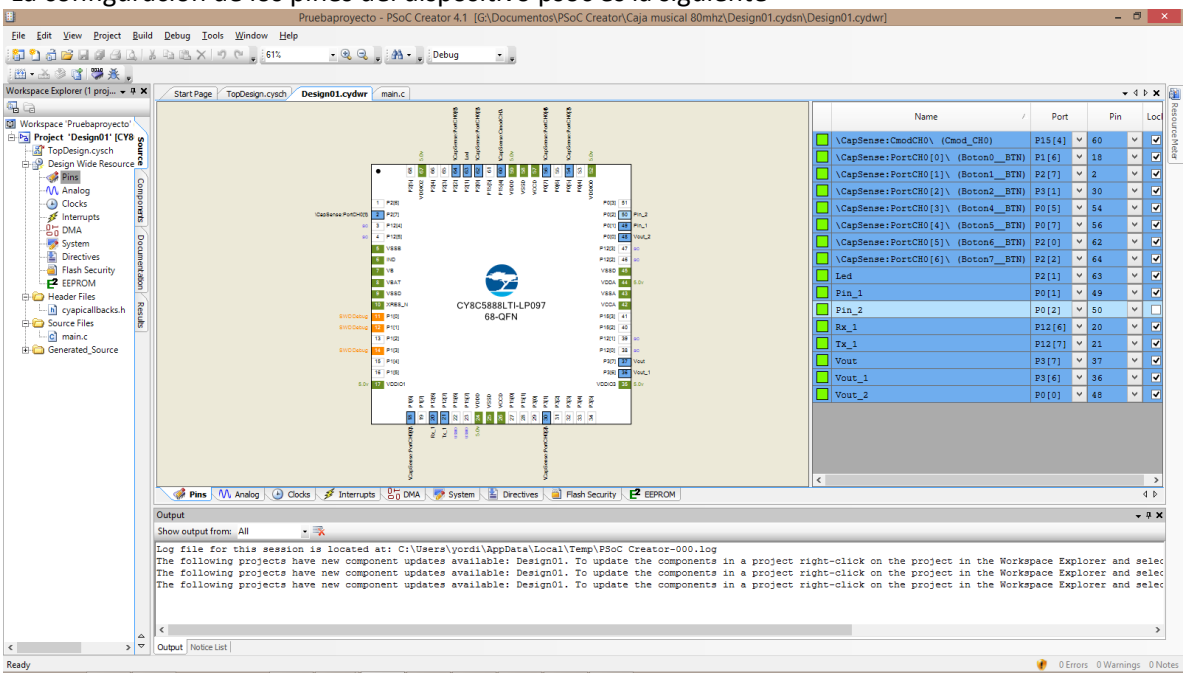
38.- Incremento en el tamaño del stack y el heap

Tambien hay que tener en cuenta la configuracion de la prioridad de interripciones que queda de la siguiente forma.



39.- Configuracion final de la prioridad de interrupciones

La configuracion de los pines del dispositivo psoc es la siguiente



40.- Configuracion de Pines dentro del psoc

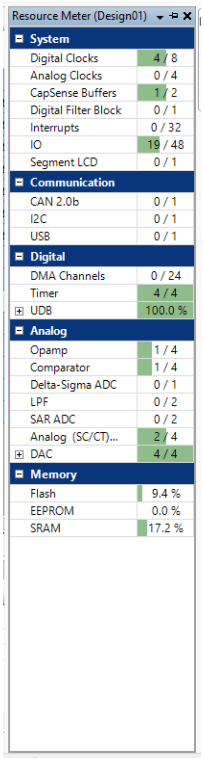
Hasta aquí entonces tenemos desarrollado nuestro dispositivo final de este proyecto es capaz de generar hasta 9 notas de polifonía pero actualmente las rutinas de interricion de los canales solo están programadas para generar 6. Un video con el funcionamiento de este dispositivo se muestra en el siguiente vínculo asociado a la imagen.

Analisis y discusión de los resultados

Como se puede observar en el video del funcionamiento de la etapa final de nuestro proyecto, se consiguió realizar el diseño e implementación de un sintetizador de sonidos el cual se basa en la técnica de síntesis por tablas de onda. La cantidad máxima de polifonía que puede generar esta dada por nuestro hardware. Dado que nuestra implementación para la polifonía se basa en el uso de Timers el numero máximo de polifnia que nuestro dispositivo puede generar es hasta 9 notas sin embargo solo se programo para generar hasta 6.

La calidad del sonido generado esta dada tanto por la tabla de ondas usada para la síntesis , como por la frecuencia de muestreo que se utiliza. Sin embargo el aumentar la frecuencia de muestreo genera un compromiso entre el desempeño de el dispositivo dado que amenta la carga de trabajo del microntrolador y produce que este tenga un comportamiento erróneo al momento de reproducir las melodias.

La cantidad de hardware digital utilizado para la implementación del sintetizador dentro de nuestro dispositivo PSOC es bastante elevada dado que se utlizaron el 100% de los recursos digitales presentes en este dispositivo como se puede observar en la siguiente imagen.



41.- Reporte de los recursos Utilizados dentro del PSOC 5

El uso de un segundo dispositivo para la lectura de los archivos MIDI fue una gran ventaja, dado que permitio no amuentar carga al microprocesador del sintetizador. Ademas que las bibliotecas utilizadas en este dispositivo permitieron que la implementación de la lectura de los archivos MIDI se facilitara y esto permitio que el dispositivo pudiera ser completado en el tiempo requerido.

El dispositivo actual genera un timbre agradable similar al de una caja de música el cual era el objetivo de este proyect.También sobrepasa la polifonía planteada en un incio en la propuesta sin embargo aun no se programa para funcionar a su máxima capacidad.

Hubo que hacer varias modificaciones y experimentos con varias plataformas de hadware para llegar al resultado final sin embargo toda esa experimentación permitio desarrollar el dispositivo que se tiene actualmente.

Conclusiones:

En conclusión el dispositivo desarrollado en el presente proyecto cumple con las características planteadas en un inicio, aunque hubo varios inconvenietes en el desarrollo de este se lograron superar. El desarrollo de este proyecto perimitio adquiri experiencia en el diseño y desarrollo de un sistema embebido asi como la interconexión entre 2 plataformas de desarrollo.

Referencias Bibliográficas:

[1] Sharon Ganske, *Making Marvelous Music Boxes*, New York: Sterling Publishing Company, 1997.

[2] AVNET, Zedboard Getting Started Guide, Internet: http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7-1.pdf

[3] Texas Instruments, Tiva C series Description, Internet: http://www.ti.com/tool/EK-TM4C123GXL#descriptionArea

[4] Cypress Semiconductors, CY8C58LP Family Datasheet, Available: http://www.cypress.com/file/45906/download

[5] "Concepts, Applications, and Science," *Digital Sound & Music*. [Online]. Available: http://digitalsoundandmusic.com/.

[6] K. Steiglitz, *A Digital signal processing primer: with applications to digital audio and computer music*. Menlo Park, Ca.: Addison-Wesley, 1996.

[7] P. R. Cook, Real sound synthesis for interactive applications. Wellesley, MA: Peters, 2007.

[8] "Teensy 3.5," DEV-14055 - SparkFun Electronics. [Online]. Available: https://www.sparkfun.com/products/14055.

Referencias a herramientas y Software utilizado:

Audacity disponible en: https://www.audacityteam.org/

Hairless MIDI disponible en: http://projectgus.github.io/hairless-midiserial/

Todo el software mencionado anteriormente es de uso libre.