Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Programación de un sintetizador MIDI usando el microcontrolador MSP430FR2355

Autor: Pilar Martínez Serrano
Tutor: Manuel Ángel Perales Esteve

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

[1]

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Programación de un sintetizador MIDI usando el microcontrolador MSP430FR2355

Autor:
Pilar Martínez Serrano

Tutor:
Manuel Ángel Perales Esteve
Profesor Titular de Universidad

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

[1] Imagen de portada [16].

Trabajo Fin de Grado: Programación de un sintetizador MIDI usando el microcontrolador MSP430FR2355

Autor: Pilar Martínez Serrano

Tutor: Manuel Ángel Perales Esteve

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis amigos

AMDG

Agradecimientos

A papá y mamá, porque en éste, y en cada proyecto que emprenda en la vida, él me sostiene desde el cielo, y ella me enseña a poner los pies en la tierra.

A Javier, mi ejemplo y mi mayor apoyo, incluso a 9000km de distancia.

A toda mi familia: mis hermanas, Maria y Carmen; mis abuelos, tíos primos… en especial a Cris y Rafa; por tanto interés mostrado durante los últimos meses en la marcha de mi trabajo.

A tía Mapino, por darme su empujón de entusiasmo durante la recta final de este proyecto. También por llevarme con ella hasta la China (quería agradecérselo por escrito en alguna parte).

A Angelita, Chío, María y Sofi, por su amistad incondicional, aunque en este último curso, entre TFGs, exámenes y otras obligaciones, nos hayan quedado muchas cervecitas pendientes.

A Irina, por sus lecciones de piano y de vida.

A Manolo, por la paciencia y el tiempo que me ha dedicado, por sus consejos, y por prestarme su sintetizador como semilla para desarrollar este proyecto.

A Dios, por poner a todas estas maravillosas personas a mi lado.

Pilar Martínez Serrano

Sevilla, 2020

Resumen

Fue el azar, y no su afición por la música, lo que llevó al violonchelista y científico ruso Lev Serguéyevich Termén a inventar el primer sintetizador de la Historia: trataba de construir un detector de ondas electromagnéticas, cuando observó que el artefacto emitía un sonido variable según desplazaba su mano sobre el circuito. Así nació el Theremin, en 1920. Su idea fue desarrollada por la *Radio Corporation of America (RCA*), que emprendió la construcción de los primeros sintetizadores modulares: osciladores, filtros y amplificadores constituían módulos que, conectados de forma diversa, daban lugar a un número de posibilidades sonoras casi infinitas. Estos circuitos podían llegar a ocupar una habitación, hasta que, en 1970, Robert Moog concibió el *MiniMoog*, logrando reducir el tamaño de tan monstruosos instrumentos. Integró los módulos de su sintetizador en dispositivos más portables que hicieron posible su comercialización. Su modelo sirvió de inspiración a numerosos fabricantes, expandiéndose por entonces la tecnología de los instrumentos electrónicos. Pero, ¿quién le diría a Moog que el entonces considerado minúsculo *MiniMoog* podría llegar a caber en la palma de su mano? La evolución de la electrónica ha hecho posible programar los distintos módulos físicos que conforman el sintetizador en las reducidas dimensiones de un microcontrolador. El objetivo de este proyecto será precisamente la programación de un sintetizador digital, que hemos bautizado como *Midisynth 2.0*, al tratarse de un intento por mejorar el sintetizador *Midisynth 1.0*, desarrollado por el profesor Manuel Perales. El dispositivo se conecta a un controlador, como puede ser un teclado electrónico, del que recibe las notas a reproducir, codificadas en mensajes estándar *MIDI*. Implementaremos el sintetizador con el microcontrolador *MSP430FR2355*, en lugar del *MSP430G2553* utilizado por la versión anterior. La razón principal de este cambio es el convertidor digital-analógico que incorpora, con el que generaremos una onda sonora de mayor resolución a la que antes se obtenía mediante modulación *PWM.* Otras prestaciones ofrecidas por el nuevo microcontrolador nos permitirán introducir mejoras adicionales.

Abstract

It was chance, and not his fondness for music, that led Russian cellist and scientist Lev Sergeyyevich Termen to invent the first synthesizer in history: he tried to build an electromagnetic wave detector, when he observed that the device emitted a variable sound as he moved his hand over the circuit. Thus the Theremin was born in 1920. His idea was developed by the *Radio Corporation of America (RCA),* which undertook the construction of the first modular synthesizers: oscillators, filters and amplifiers constituted modules that, connected in different ways, gave rise to a number of almost infinite sound possibilities. These circuits could even occupy a room, until, in 1970, Robert Moog conceived the *MiniMoog*, managing to reduce the size of such monstrous instruments. He integrated the modules of his synthesizer into more portable devices that made its commercialization possible. His model inspired numerous manufacturers, expanding the technology of electronic instruments. But who would tell Moog that the, considered by the time tiny, *MiniMoog* could fit in the palm of his hand? The evolution of electronics has made it possible to program the different physical modules that make up the synthesizer in the small dimensions of a microcontroller. The objective of this project will be precisely the programming of a digital synthesizer, which we have named *Midisynth 2.0*, as it is an attempt to improve the *Midisynth 1.0* synthesizer, developed by Professor Manuel Perales. The device is connected to a controller, such as an electronic keyboard, from which it receives the notes to be played, encoded in standard *MIDI* messages. We will implement the synthesizer with the *MSP430FR2355* microcontroller, instead of the *MSP430G2553* used by the previous version. The main reason for this change is the digital-analog converter that it incorporates, with which we will generate a higher resolution sound wave than was previously obtained through PWM modulation. Other features offered by the new microcontroller will allow us to introduce additional improvements.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

<<	Desplazamiento de bits a la izquierda
>>	Desplazamiento de bits a la derecha
Clk	Señal de reloj
GND	Conexión a tierra
I/Os	Pines de entrada/salida
kbps	kilobit /segundo
LCD	Pantalla: *"Liquid Crystal Display"*
RX	Línea de recepción de datos
TX	Línea de transmisión de datos
Vcc	Tensión de alimentación

1 PREFACIO

Ars longa, vita brevis

- Hipócrates -

Este acercamiento a la síntesis del sonido toma como punto de partida el sintetizador digital programado Midisynth 1.0. Trataremos de perfeccionarlo adaptándolo al microcontrolador *MSP430FR2355*, que ofrece funcionalidades novedosas respecto al microcontrolador utilizado anteriormente. El granito de arena que aportará el proyecto al sintetizador referido será la implantación del nuevo microcontrolador, junto a ciertas mejoras posibilitadas por el mismo. Adicionalmente, esto supondrá un avance potencial para el dispositivo, pues las ventajosas prestaciones del chip elegido nos abren la puerta a un sinfín de posibilidades de trabajo futuro para el proyecto. Pero como decía a sus pupilos el griego Hipócrates, *el arte es infinito, pero nuestra vida corta.*

1.1 Estado del arte: Midisynth 1.0

El dispositivo Midisynth 1.0 es un sintetizador digital de audio, basado en la suma de ondas elementales, y controlado mediante protocolo de comunicación MIDI. Su elemento constituyente principal es el microprocesador de bajo consumo MSP430G2553 de Texas Instruments. Éste se integra en una placa PCB ("Printed Circuit Board") junto a diversos periféricos, conformando una interfaz sencilla, que permitirá al usuario crear diversos efectos musicales sobre la secuencia de notas recibida vía MIDI.

Este sintetizador fue diseñado e implementado en 2017 por Manuel Perales, profesor de la Escuela de Ingenieros de la Universidad de Sevilla. Actualmente constituye una herramienta docente en asignaturas impartidas por el Departamento de Electrónica.

La producción del sonido se lleva a cabo mediante la generación de diversas ondas digitales, que se sumarán produciendo cierta onda resultante, también digital. Opcionalmente, dicha onda podrá ser modulada mediante varias técnicas para obtener determinados efectos sonoros. Finalmente, una *modulación por anchura de pulsos, PWM,* modificará la naturaleza digital de la onda, produciendo la onda analógica que, debidamente amplificada alcanzará nuestros oídos.

Podemos producir hasta ocho ondas básicas para conformar la onda deseada, como fruto de su suma. Estas ondas elementales se encuentran almacenadas en una tabla. Su frecuencia fundamental ó tono dependerá de la información recibida por el canal MIDI, es decir, la secuencia de notas musicales a generar será indicada por mensajes MIDI de entrada al programa. Estos mensajes informarán además de la fuerza con la que son ejecutadas las notas, condicionando así la amplitud de las ondas.

El programa de síntesis está implementado en lenguaje C, utilizando el entorno de programación Code Composer, específico para microcontroladores del fabricante Texas Instruments.

1.2 Alcance del proyecto

Este proyecto tiene como objetivo el perfeccionamiento del sintetizador que acabamos de presentar, creando

una nueva versión del mismo: Midisynth 2.0. La novedad más importante será la sustitución del microcontrolador MSP430G2553 por otro de mayores prestaciones, el MSP430FR2355. Dicho dispositivo presenta una funcionalidad de gran interés para el tratamiento de las ondas: el convertidor digital-analógico, *DAC*. Generando la onda analógica de salida mediante el convertidor se aumentará considerablemente la resolución de la misma, ya que antes se obtenía mediante modulación *PWM,* con resultados de poca calidad, incluso pese a incluir una etapa de filtrado.

El nuevo microcontrolador constituirá además una apertura a mayores posibilidades para el programa de síntesis. Con una capacidad de memoria RAM mayor, nos será posible condensar los cuatro efectos de modulación seleccionables, en un mismo programa C, mientras que antes estos debían ser ejecutados y cargados en el MCU por separado. Dispondremos en la placa un pulsador extra para la selección del efecto.

Por otro lado, realizaremos una nueva tabla de ondas, incluyendo nuevas formas, algunas básicas y otras asociadas a determinados instrumentos.

Adicionalmente, contaremos con un control de volumen, mediante la incorporación de un potenciómetro logarítmico en la nueva tarjeta.

1.3 Etapas

Para llevar a cabo este trabajo se propone un **planteamiento conservador**: partiendo de la versión anterior del sintetizador, el Midisynth 1.0, **las mejoras se han introducido progresivamente**, verificando tras cada etapa el correcto funcionamiento del sistema completo.

Como primer paso, realizamos el **análisis del sintetizador de referencia**, lo cual ha supuesto un estudio detallado de los conceptos teóricos relacionados con su funcionamiento. Ello nos ha permitido interpretar el código que lo implementa, capacitándonos así para su futura reprogramación.

Seguidamente, se ha acometido la **programación del sintetizador**, una "traducción" del código inicial, específico para el microcontrolador *MSP430G2553*, a un nuevo programa compatible con el *MSP430FR2355*. Consultando los *datasheet* de sendos dispositivos, hemos ido cambiando los registros y variables propias, para lograr las mismas funcionalidades. Durante esta etapa, aunque así descrita pudiera parecer sencilla al lector, topamos con ciertas complicaciones que han requerido abundantes pruebas, paciencia y medidas con el osciloscopio hasta conseguir solventarlas.

Una vez puesto en marcha el programa de síntesis original en el nuevo microcontrolador, acometemos las modificaciones que se planteaban en el apartado previo: el método de modulación *PWM*+filtrado para la conversión digital-analógica de la onda se sustituye por el convertidor *DAC* y se construye la nueva tabla de ondas, adaptando el programa a la misma.

Finalizada la tarea de programación, y probado su éxito, procedemos a fabricar la tarjeta Midisynth 2.0, adaptada para la conexión del *MSP430FR2355*. Se han realizado ciertas modificaciones del hardware, como la integración del potenciómetro y la eliminación del filtro que precisábamos al modular por *PWM*.

1.4 Estructura del documento

El texto seguirá un orden paralelo a las etapas trabajadas, que referíamos en el apartado previo. Comenzamos con un capítulo teórico (*capítulo 2*, que sigue a esta introducción), en el que expondremos los conocimientos básicos necesarios para comprender el funcionamiento del sintetizador. Estos han sido adquiridos en una primera etapa de documentación bibliográfica detallada.

Seguidamente, en los *capítulos 3 y 4*, se explicarán las herramientas de software utilizadas y los elementos hardware que conforman nuestro dispositivo electrónico, así como las funciones del sistema que cumplen cada uno.

El capítulo 5, el más extenso, está dedicado a la programación del microcontrolador. Se ilustra el funcionamiento general del programa, la estructura del código, y los procedimientos de las subrutinas. Además se aclaran las diferencias significativas respecto a la versión anterior del sintetizador. En el primer anexo se

incluye el código completo. Las nuevas formas de onda incorporadas en el programa se ilustran en el anexo II, junto con el código implementado en *Matlab* para obtenerlas.

La última etapa del proyecto se desarrolla brevemente en el capítulo 7, en el que comentamos el diseño del circuito impreso y su fabricación. Los diagramas esquemáticos realizados se adjuntan en el Anexo III. Para terminar, los apartados 8 y 9 cerrarán el texto, exponiendo los resultados así como las conclusiones obtenidas.

2 FUNDAMENTOS TEÓRICOS

Computers and electronic music are not the opposite of the warm human music. It's exactly the same.

- Bill Laswell -

Trataremos en primer lugar de dar unas pinceladas acerca de la base teórica que sustenta a este proyecto, centrándonos en dos puntos: la síntesis digital del sonido y el protocolo MIDI. La asimilación de estos conceptos ha sido esencial para comprender la estructura del código de referencia, y emprender así su reprogramación para la introducción de mejoras.

Comenzamos repasando las propiedades básicas del sonido, que nos servirán de enlace para ilustrar las distintas técnicas de síntesis, principio de los instrumentos electrónicos. Por otro lado explicaremos el protocolo MIDI, estándar de comunicación entre equipos de sonido, tanto instrumentos electrónicos como módulos de síntesis ó computadoras. Al final de cada apartado se aclará la aplicación práctica a nuestro sintetizador de las ideas desarrolladas.

2.1 La síntesis del sonido

2.1.1 Descripción de la onda sonora

El sonido puede definirse como la *sensación producida en el órgano del oído por el movimiento vibratorio de los cuerpos, transmitido por un medio elástico, como el aire* [1]. Desde el punto de vista físico, denotamos como ondas sonoras, o sonido, a las ondas mecánicas generadas por dichas vibraciones, que originan una perturbación en el medio material y se propagan a través del mismo. El foco de la perturbación puede ser de naturaleza muy variada, desde el tintineo de un yunque golpeado con un martillo, hasta la vibración de las cuerdas vocales humanas, que originan nuestra voz, o el frotado de las cuerdas de un violín con su arco.

Figura 2-1. Ejemplo de onda sonora: foco de emisión, medio de propagación y receptor[2].

Las características que permiten distinguir los distintos sonidos son cuatro: **amplitud de la onda, frecuencia de vibración, timbre y duración**.

[2] Imagen tomada de https://eltelescopiodegalileo.wordpress.com/2015/09/24/20-hz/. Fecha de consulta: Noviembre de 2019.

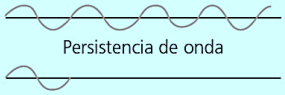
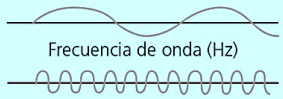
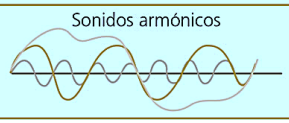
El intervalo de tiempo en el que persiste la onda, repitiéndose periódicamente, constituirá la **duración** del sonido producido.

La **frecuencia de vibración** de la onda sonora se relaciona directamente con lo que conocemos como *tono o altura* del sonido. Esto dará lugar al concepto de **nota musical**: cada nota se caracterizará por una frecuencia fundamental determinada.

La **amplitud** de la onda será el parámetro que defina la *intensidad* del sonido: cuanto más fuerte se golpee el yunque o se frote el violín, más intenso percibiremos su son.

Sin embargo, las ondas sonoras no suelen ser puras, sino que se componen de una onda predominante, de **frecuencia fundamental**, a la que se suman otras ondas denominadas **armónicos**, con frecuencias múltiplos de la fundamental. Las múltiples combinaciones de armónicos vienen a caracterizar el **timbre** del sonido. De esto deducimos que la composición de armónicos entre instrumentos del mismo tipo será similares, distinguiendo así nuestro oído el sonido de un violín de aquel producido por un piano, o un cello, o una voz… pero también existirán diferencias, más sutiles, entre los armónicos de elementos con naturaleza similar: no percibiremos igual la voz de todas las personas, ni todos los pianos suenan del mismo modo.

Tabla 2-1. Características de las ondas sonoras[3].

Cualidades	Distinguimos	Se produce por
Duración	Largo	
	Corto	
Altura	Grave	
	Agudo	
Intensidad	Fuerte	
	Suave	
Timbre	Voces	
	Instrumentos	

2.1.2 El Sintetizador

Se atribuye el adjetivo **"sintético"** a *aquellos productos que se obtienen por procedimientos industriales y que reproducen la composición y propiedades de uno natural* [1]. En el campo de estudio que nos ocupa aplicamos esta definición al **sonido sintético**, mediante el cual se trata de imitar las ondas sonoras reales, manipulando sus cuatro rasgos distintivos: frecuencia, amplitud, duración y armónicos.

Mientras el sonido natural se basa en ondas mecánicas, originadas por medios físicos (golpes, vibraciones, pulsado de cuerdas en la guitarra o frotado de las mismas en un violín…), **el sonido sintético se produce mediante ondas eléctricas**. Dichas ondas eléctricas podrán implementarse directamente con osciladores eléctricos, técnica conocida como **síntesis analógica,** ó bien mediante programación, procedimiento más típico, denominado **síntesis digital**. En este último caso, se genera información binaria que es traducida posteriormente a valores de voltaje a través de un convertidor digital-analógico *(DAC)*.

Para hacer llegar a nuestro oído la información producida por un sintetizador se emplean transductores, dispositivos capaces de transformar cierta energía de entrada en otra de diferente naturaleza a la salida. Un **transductor electro-acústico permite convertir las ondas eléctricas**, variaciones de voltaje, **en ondas acústicas**, u oscilaciones en la presión del aire [2]. Los altavoces o auriculares estándar son ejemplos comunes

[3] Imagen tomada de https://www.correodelmaestro.com/publico/html5022018/capitulo5/autorretrato_sonoro.html. Fecha de consulta: Diciembre de 2019.

de transductores electro-acústicos.

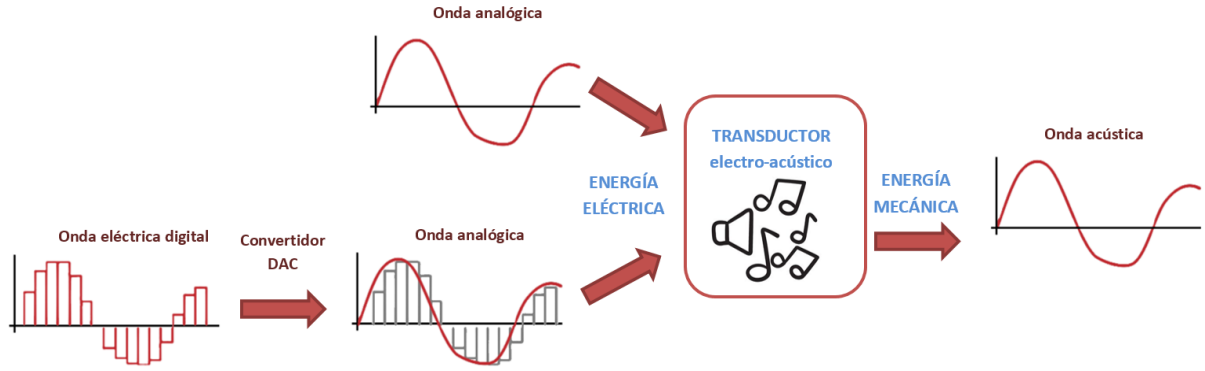


Figura 2-2. Conversión de las ondas sintetizadas en ondas sonoras.

En resumen, **el sintetizador será un instrumento musical en sí mismo**, que producirá sonidos audibles mediante circuítos eléctricos. Es un **error muy extendido el hecho de confundir los sintetizadores con "teclados"**: aunque estos dispositivos suelen estar accionados por las teclas de un piano electrónico, en sentido estricto *el sintetizador es el aparato que genera el sonido, independientemente del mecanismo utilizado para que un intérprete lo controle* [3]. Cualquier instrumento electrónico, no solamente el teclado, deberá incorporar un módulo de sintetizador para su funcionamiento. Además estos módulos podrán ejecutarse directamente mediante controladores de voltaje, en el caso de sintetizadores analógicos, ó a través de **controladores MIDI,** si se trata de sintetizadores digitales. Ilustraremos este tipo de control digital en el apartado siguiente del capítulo.

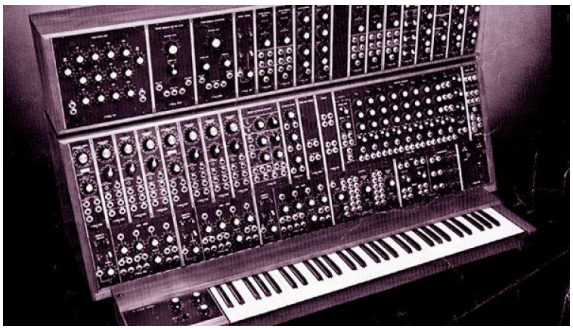


Figura 2-3. Sintetizador accionado por teclado[4].

2.1.3 Módulos del sintetizador

Los componentes electrónicos básicos de un sintetizador son cuatro, siendo posible disponer múltiples combinaciones de ellos en un mismo dispositivo para lograr una configuración más versátil del sonido:

- Oscilador principal. En los sintetizadores analógicos será un oscilador controlado por tensión, **VCO** (*"Voltage Controlled Oscillator"*), mientras que en los digitales encontraremos un **DCO** *("Digitally Controlled Oscillator").* La onda que genera se identifica como **portadora o *carrier***.
- Oscilador de baja frecuencia. Conocido como **LFO** *("Low Frequency Oscillator"),* genera la señal que llamamos **moduladora o envolvente.**
- Filtro. Su función es rebajar ó eliminar la amplitud de ciertas frecuencias, modificando los armónicos

[4] Imagen tomada de la fuente: https://www.radionica.rocks/noticias/el-sintetizador-la-transformacion-de-los-sonidos. Fecha de consulta: Octubre de 2019.

de la onda original del oscilador, para cambiar el timbre del sonido [3].

- ❖ Amplificador. Al igual que en el caso del oscilador, podemos distinguir entre el **VCA** *("Voltage Controlled Amplier")* y el **DCA** *("Digitally Controlled Amplifier"),* según trabajemos con valores analógicos ó digitales, respectivamente.

La relación entre estos elementos dependerá del efecto sonoro buscado. Generalmente se encadenan de este modo: Oscilador principal –> Filtro –> Amplificador. El LFO puede configurarse para alterar de forma cíclica los parámetros característicos de cualquiera de los 3 elementos anteriores.

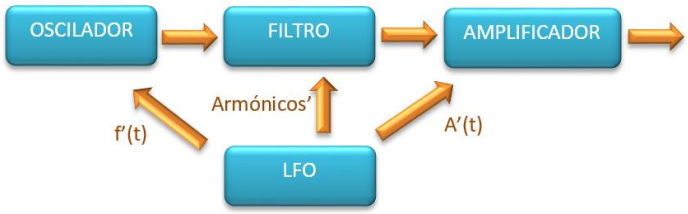


Figura 2-4. Encadenamiento de los elementos del sintetizador y posibles interacciones con el LFO.

2.1.4 Efectos de modulación

Las distintas configuraciones de los elementos anteriores en el sintetizador darán lugar a que se conoce como efectos de modulación:

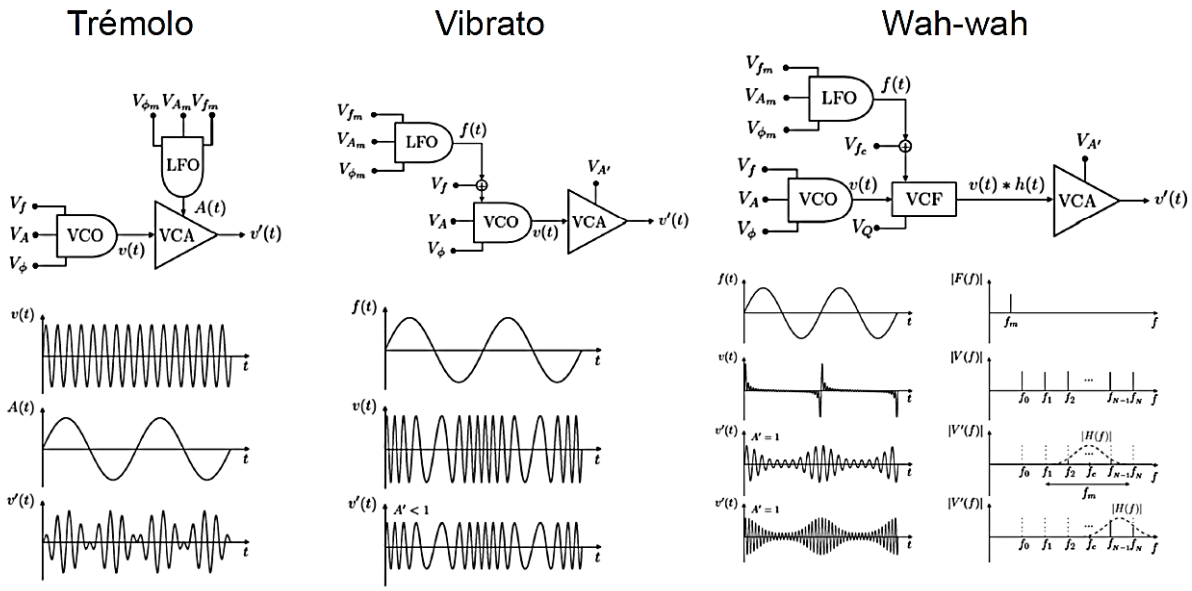


Figura 2-5. Efectos de modulación[5].

- o **Modulación en amplitud** (**AM,** *"Amplitude Modulation*"), también llamado **efecto trémolo**, cuando el LFO se dirige hacia el amplificador, para cambiar periódicamente la amplitud.
- o **Modulación en frecuencia** (**FM,** "*Frequency Modulation*") ó **efecto vibrato**, cuando el LFO afecta a la frecuencia del oscilador.

[5] Fuente: Kristian Alonso Stenberg, "Sintetizador FM para uso en tiempo real a partir de Csound" PFC. Univ. De Alicante

- **Efecto wah-wah**, en el caso de que el LFO modifique los parámetros del filtro.

Debemos aclarar que existe una diferencia significativa entre la modulación *FM*, propiamente dicha, y el efecto vibrato. Aunque el funcionamiento de ambos efectos es muy similar, una variación cíclica de la frecuencia, la denominación de **modulación FM** se aplica cuando el vibrato es muy rápido [3]: el modulador empleado ya no es un LFO (*Oscilador de baja frecuencia*) sino otro oscilador con frecuencia del orden de la onda *carrier*.

Para lograr el efecto vibrato se emplean ondas moduladoras con frecuencias bajas, alterando ligeramente la afinación (frecuencia) de la onda principal, sin llegar a modificar notablemente su amplitud. En cambio, la onda resultante de aplicar *FM* será completamente distinta a la original, dependiendo fuertemente de la frecuencia y amplitud de la moduladora. El ratio de la amplitud y la frecuencia moduladora se conoce como **índice de modulación**. El espectro de frecuencias de la onda sintetizada dependerá fuertemente de dicho índice: conforme aumente, irán apareciendo bandas laterales en el espectro de frecuencia [4] (variación de armónicos, timbre).

En la figura 2-6 puede observarse como, al aumentar la frecuencia de la onda moduladora por encima de la frecuencia de la portadora ó *carrier*, la onda resultante de la modulación se modifica radicalmente. La primera situación mostrada será lo que hemos definido como *vibrato*, que evoluciona hacia una *modulación FM* en el último caso, cuando la frecuencia de Sm supera a la de Sc.

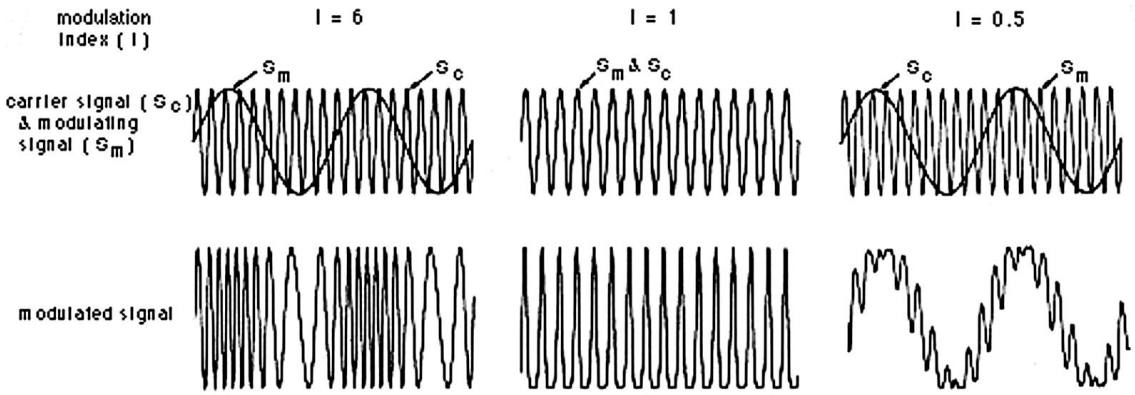


Figura 2-6. Modulación en frecuencia para distintos índices de modulación[6].

Otro efecto importante a considerar es la **modulación ADSR, variante de la modulación en amplitud**, que trata de imitar la dinámica musical de las notas al ser ejecutadas por un instrumento tradicional. Se basa en la aplicación una envolvente de 4 etapas a cada nota, modificando su intensidad de forma característica desde que es pulsada (*atacada*) hasta su liberación.

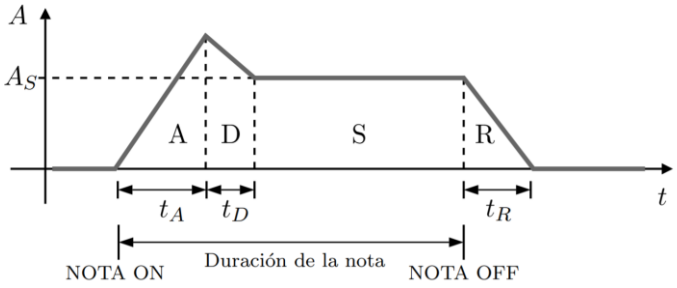


Figura 2-7. Forma característica de envolvente para modulación ADSR[7].

[6] Fuente: J. Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation," Journal of the Audio Engineering Society, vol. 21, pp. 526-34, 1973.

A diferencia del trémolo, en la modulación *ADSR* no se observa en la amplitud de onda la repetición de la envolvente de forma periódica, sino que se sigue un **patrón de intensidad aplicado una sola vez a cada sonido emitido**: al ejecutarse cada nota, la amplitud experimenta un aumento brusco, decae ligeramente y es sostenida hasta el instante en el que se libera, mitigándose progresivamente hasta el silencio. Se ilustra esta diferencia en la figura 2-8.

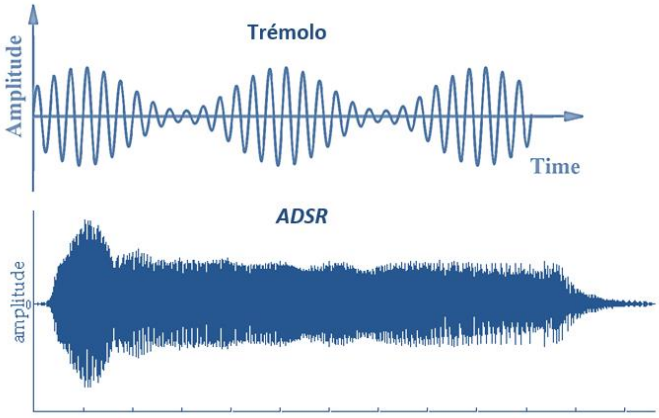


Figura 2-8. Comparación de modulaciones en amplitud.

2.1.5 Técnicas de síntesis

Entre los elementos constituyentes de un sintetizador, que fueron comentados previamente, la pieza fundamental será el oscilador principal, generador de la onda primitiva. La presencia del resto de elementos (filtro, LFO y amplificador) será opcional. Los componentes se combinarán para "retocar" la onda inicial, en configuraciones que dependerán del método de síntesis empleado. Las técnicas más comunes son las que siguen:

- **Síntesis substractiva:** se basa en el filtrado de la onda elemental, alterando sus armónicos, y con ello, su timbre. Además del **oscilador y el filtro**, básicos en esta técnica, podrán incluirse el amplificador, para variar la amplitud inicial, y el LFO para realizar modulaciones.
- **Síntesis aditiva:** en su configuración encontramos **dos ó más osciladores** cuyas ondas elementales se suman, generando una onda completamente distinta a las iniciales. La presencia del filtro, el amplificador y el LFO será opcional.
- **Síntesis por tabla de ondas:** las ondas son construidas por un programa, que lee punto a punto ciertas muestras digitalizadas de las ondas a reproducir, almacenadas en una tabla.
- **Síntesis de modelado físico:** consiste en el modelado matemático de la fuente del sonido a través de un conjunto de ecuaciones y algoritmos que simulan su comportamiento [3].

2.1.6 Aplicación práctica

El **sintetizador digital** que programaremos utilizará tanto el **método de síntesis aditiva** como el basado en **tabla de ondas**; producirá los sonidos sumando ondas generadas a partir de ciertas muestras de ondas digitalizadas, almacenadas en una tabla. En concreto, el programa podrá manejar hasta ocho ondas elementales. A cada una de ellas se asocia cierto desfase, en el intervalo [-63,+63], de modo que al sumarlas, podrán estar "desplazadas" unas respecto a otras, influyendo así los armónicos de la onda resultante y, por consiguiente, su timbre.

Midisynth 2.0 comprende **tres de los módulos mencionados antes: oscilador, amplificador y oscilador de baja frecuencia (*LFO*).** El filtro se omite, ya que no haremos uso del método substractivo. Aunque las ondas a sumar, producidas por el oscilador, se calculen en una misma rutina del programa *(calcula_onda + interrupción DCO)*, que veremos más adelante, queremos aclarar que este procedimiento emulará el

[7] Fuente: Kristian Alonso Stenberg, "Sintetizador FM para uso en tiempo real a partir de Csound" PFC. Univ. De Alicante.

comportamiento de varios osciladores físicos; si hemos seleccionado la suma de 4 ó 5 ondas, la función *calcula_onda* equivaldría a la configuración de 4 ó 5 osciladores principales, respectivamente.

Como dijimos, el LFO puede afectar a distintos módulos en función del efecto de modulación buscado. Nosotros implementaremos **cuatro efectos, a elegir: trémolo, vibrato, *ADSR* y modulación *FM***. Configuraremos una de las ocho ondas almacenadas en la tabla como onda moduladora ó envolvente, que será la que emitiría físicamente el oscilador de baja frecuencia. El caso de la modulación *ADSR,* difiere del resto, pues la onda moduladora toma la forma característica, de cuatro segmentos, que veíamos en la figura 2-7. Los segmentos se modelarán según el valor seleccionado para los parámetros *tA, tD, SUST y tR.*

Para la modulación en amplitud, *AM (trémolo* ó *ADSR)*, el LFO afectará al módulo amplificador. La onda principal, fruto de la síntesis aditiva, variará su amplitud, escalándola con la amplitud de la envolvente, para producir el efecto trémolo ó el *ADSR,* respectivamente.

Para el efecto vibrato, en cambio, la onda sumada modificará su frecuencia a cada instante como función de la frecuencia del *LFO*.

Si se selecciona la modulación en frecuencia, *FM,* no intervendrá el oscilador *LFO*, sino que tendremos un oscilador "ficticio", con frecuencia de orden mayor, calculada a partir de la frecuencia de la onda principal modificada ligeramente según el valor que demos al índice de modulación, en el rango [-12, +12]. El procedimiento es el siguiente: imaginemos que la nota a producir es la número 15 del vector de frecuencias, la nota RE, con frecuencia fundamentel 146.83Hz; si el índice de modulación es I = 3, sumaremos tres posiciones en el vector de frecuencias, tomando la posición número (15+3) = 18 del vector, correspondiente a la frecuencia 174.61Hz. Tal frecuencia será la asociada al oscilador *FM.*

2.2 Principios del protocolo MIDI

A lo largo de los años setenta, la síntesis musical experimentó un gran impulso, extendiéndose rápidamente el desarrollo de instrumentos electrónicos en la industria: fabricantes de instrumentos tradicionales, como las casas *Yamaha* o *Kawaii*, apostaron por la aplicación de esta nueva tecnología a los instrumentos convencionales, y nuevas compañías como *Korg, Roland* y *Sequential Circuits*, se abrieron camino en estos años, especializándose en la producción de sintetizadores.

A comienzos de los años 80, surgió la necesidad de crear una interfaz de comunicación entre los equipos de música electrónica, tanto para hacer posible el intercambio de información entre los instrumentos y las computadoras, como para lograr la interconexión de los sintetizadores entre sí. Con este objetivo fue concebido, en 1983, el protocolo estándar de comunicaciones MIDI *("Musical Instrument Digital Interface")*. La especificación fue creada como convenio entre fabricantes, que constituyeron la MMA (Midi Manufacturers Association) [4].

Figura 2-9. Logo MIDI de la MMA[8].

El estándar MIDI incluye especificaciones hardware de cables y conectores: se emplean conectores normalizados tipo *DIN 180º* de 5 pines. Como la comunicación es unidireccional, los equipos contarán con puertos de conexión IN y OUT. Además, una tercera conexión opcional, MIDI-THRU, emite una copia de todo lo que es recibido en el puerto de entrada, permitiendo enlazar unos equipos con otros en cadena, de manera que todos respondan a una fuente de datos MIDI inicial [5].

[8] Imagen tomada de https://es.wikipedia.org/wiki/MIDI#El_impacto_del_MIDI_en_la_industria_de_la_m%C3%BAsica. Fecha de consulta: Junio de 2019.

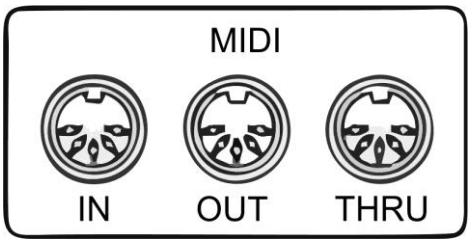


Figura 2-10. Puertos de conexión MIDI[9].

Los cables tendrán una longitud máxima de 15 metros, para evitar interferencias. Otra precaución que indica el estándar será disponer optoacopladores entre los conectores de distintos equipos para aislarlos eléctricamente.

2.2.1 Estructura del mensaje

Los mensajes transmitidos vía MIDI no sólo contienen información sobre la ejecución musical (notas, tono, dinámicas), sino también acerca de la sincronización entre los equipos, efectos de modulación, timbres de los instrumentos, etc.

El protocolo MIDI describe un tipo de **comunicación serie**, es decir, la información se transmitirá bit a bit, siguiendo las tramas de bits típicas de este tipo de transmisión digital. Los bits se agruparán en grupos de 10, denominados MIDI-bytes; un primer bit de *Start bit* o bit de comienzo, 8 bits de datos y un *Stop Bit* o bit de terminación. La velocidad de transmisión MIDI es de 31.25 kbps, es decir, en un segundo se enviarán 31 250 bits[10] ó, equivalentemente, 3125 MIDI-bytes [6].

Cada mensaje se compone de 2 ó 3 *MIDI-Bytes*: un byte de *status* y uno ó dos bytes de datos. El primero de ellos indica el tipo de dato que será enviado en los subsiguientes bytes de datos. Los bytes de status y datos se diferencian por el primer bit, el MSB *("Most Significant Bit").* En los bytes de *status* el MSB será igual a '1', mientras que en los de datos este primer bit tomará valor '0'.

Los 7 bits restantes del *Status-Byte* señalarán el tipo de mensaje que se envía, codificado en 3 bits, y el canal que debe atender el mensaje, mediante los 4 últimos bits. De este modo, **identificaremos hasta $2^3 = 8$ tipos de mensajes distintos, y podremos direccionar $2^4 = 16$ canales** mediante el protocolo MIDI.

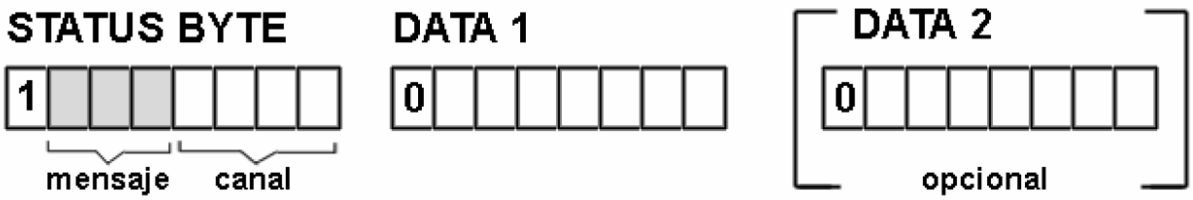


Figura 2-11. Estructura binaria de un mensaje MIDI [7].

2.2.2 Tipos de mensaje

Como mencionábamos en el anterior apartado, **son ocho los tipos de mensaje** que puede transmitir una línea de comunicación MIDI. Sin embargo, **no es frecuente que los dispositivos sean capaces de emitir todos estos mensajes, ni tampoco que puedan identificarlos.** En caso de que el equipo MIDI no comprenda un mensaje recibido lo ignorará, pero lo reenviará por la línea MIDI-THRU, si dispone de ella, para que la

[9] Imagen tomada de la fuente: https://thebassvalley.com/guia-de-midi-basico/. Fecha de consulta: Diciembre de 2019.
[10] Un baud equivale a 1 bit/segundo.

información pueda alcanzar al resto de módulos.

En la tabla siguiente se clasifican estos mensajes, junto a su estructura. Nótese que en algunos de ellos no se envía el segundo byte de datos.

Tabla 2-2. Tipos de mensajes MIDI.

Tipo de mensaje	Status-Byte	Data 1	Data 2
Note Off	"1 000 nnnn[11]"	Altura	Velocidad
Note On	"1 001 nnnn"	Altura	Velocidad
Postpulsación polifónica	"1 010 nnnn"	Altura	Presión
Cambio de control	"1 011 nnnn"	Tipo de control	Valor del control
Postpulsación de canal	"1 100 nnnn"	Presión	-
Pitch bend	"1 101 nnnn"	Byte más significativo	Byte menos significativo
Cambio de programa	"1 110 nnnn"	Nº de programa	-
Mensaje de sistema	"1 111 xxxx"	-	-

Reciben el nombre de **mensajes de canal** los siete primeros tipos que se enumeran en la tabla. Los llamados **mensajes de sistema**, en cambio, no se dirigen a ningún canal determinado; los 4 últimos bits del *Status-Byte* codificarán hasta 16 mensajes de sistema posibles.

Los mensajes de tipo **Note-ON/Note-OFF** son emitidos por el controlador MIDI al pulsar o liberar cierta nota. Los 7 bits de la altura representan la frecuencia fundamental de la nota, siendo posible generar $2^7 = 128$ notas distintas. De igual modo, la velocidad de ataque de la nota se escala en el intervalo [0,128]; constituye la fuerza de la nota, por lo que puede implementarse como la amplitud de la onda sonora.

Los mensajes de postpulsación o *aftertouch* permiten variar la velocidad ó fuerza de ataque de la nota, una vez pulsada, de ahí su nombre, de efecto **post**-pulsación; mientras la velocidad se mantiene constante durante todo el intervalo note_ON, la presión es variable para estos los efectos de *aftertouch*. En la **postpulsación de canal**, la presión indicada por el mensaje se aplicará a todos los canales, es decir, a todo el acorde de notas simultáneas. La **postpulsación polifónica** permite diferenciar presiones para las distintas notas activas, de ahí que se indique también la altura en el mensaje, para diferenciar los sonidos a los que se refiere.

En los mensajes de **cambio de control,** el primer byte indica el tipo de control, con el que se pueden configurar numerosos parámetros del sistema, como el volumen, la reverberación, la modulación, etc. El segundo byte codificará el valor de dicho control.

El conocido **pitch bend** es un control que se ejecuta mendiante una rueda que suelen incluir los teclados, como la mostrada en la figura 2-12, y sirve para desafinar la nota en ejecución hasta dos semitonos, siendo posible variar la nota -2, -1, +1 ó +2 semitonos. Los dos bytes de datos se combinan para dar un único valor con 14 bits de resolución, proporcional al ángulo de rotación, y comprendido entre -8192 y +8191. Este valor es nulo cuando la rueda está en su posición central [8].

[11] Las siglas nnnn indican los 4 bits para la codificación del canal. En los mensajes de sistema no se indica ningún canal, porque están dirigidos al sistema completo.

Figura 2-12. Ruedas de control en un teclado [8].

El único byte que comprenden los mensajes de **cambio de programa**, servirá para designar los posibles instrumentos ó efectos sonoros disponibles en el sintetizador. Se codificarán como números de programa, del 0 al 127.

Por último, podemos observar que los **mensajes de sistema**, no se dirigen a ningún canal en particular, sino al sistema completo. Estos mensajes codifican funciones como la grabación de secuencias de sonido, ó el llamado *active sensing*, mensajes de la forma "1111 1110", que se envían por parte de ciertos dispositivos cada 300 milisegundos, avisando al sistema de dicha conexión se encuentra activa.

2.2.3 Aplicación práctica

El sintetizador Midisynth 2.0 solo estará preparado para recibir información de tipo **note_ON/ note_OFF** por el canal 0. Sin embargo, se implementan algunas de las funcionalidades MIDI comentadas en la sección anterior, por medio de la interfaz de la tarjeta, para ofrecer un control más intuitivo al usuario. En concreto, el sintetizador atiende, a través de los periféricos del sistema, **órdenes MIDI de pitch bend, cambio de programa y cambio de control (modulación y volumen):** el pitch bend se controla con el desplazamiento vertical del joystick; los tres pulsadores superiores permitirán seleccionar los timbres de los sonidos a producir, emulando los distintos números de programa; la modulación deseada se configurará mediante el pulsador inferior, y el volumen será regulable con el girando la rueda del potenciómetro instalado.

Por otro lado, el programa incluye una modalidad que acepta mensajes de sistema *active sensing,* para hacerlo compatible con el piano electrónico *YAMAHA clavinova* de la serie *CLP320*, intrumento doméstico que empleamos para la realización de este proyecto. En el código actual, el sintetizador funciona con protocolo MIDI estándar, sin *active sensing*, pero dejamos comentadas las instrucciones del código anterior, para su consulta y caso de necesidad de uso en dispositivos de este tipo[12].

[12] Ver anexo I.

3 HERRAMIENTAS SOFTWARE

Se dice que las grandes disciplinas científicas son ejemplos de gigantes subidos a los hombros de otros gigantes.

También se dice que la industria del software es un ejemplo de enanos subidos a los dedos de los pies de otros enanos.

- Alan Cooper -

Si bien el grueso de este proyecto lo constituye la programación en lenguaje C del microcontrolador *MSP430FR2355* de Texas Instruments, basándonos en el software ofrecido por el propio fabricante; también nos hemos apoyado en otros programas de cálculo y diseño. A continuación repasamos brevemente cada una de las herramientas software utilizadas.

3.1 Code Composer Studio

Este entorno de programación constituye la base del desarrollo de aplicaciones para *MCU*, de Texas Instruments. Permite la edición de código fuente, compilación de programas en lenguaje C/C++ y la depuración y ejecución de los mismos. En nuestro proyecto hemos trabajado con la versión 9.0.1, que asegura el soporte para el microcontrolador *MSP430FR2355*[13].

3.2 Eagle

Para el diseño de la tarjeta Midisynth 2.0 nos apoyamos en ***Eagle*** ("**E**asily **A**pplicable **G**raphical **L**ayout **E**ditor"), software gratuito que incluye un editor de diagramas electrónicos y *PCB*s, además de bibliotecas de componentes predeterminadas.

3.3 Matlab

MATLAB (abreviatura de **MAT**rix **LAB**oratory, «laboratorio de matrices») es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M) [8]. Entre numerosas aplicaciones, Matlab constituye una herramienta muy eficaz para el tratamiento y análisis de datos, posibilitando la importación y exportación de los mismos en formatos de archivo estándar, como hojas de cálculo, imágenes, audio o vídeo.

Particularmente nos será de utilidad la posibilidad de importar archivos de audio en formato .wav, y convertirlos en vectores de puntos muestreados. Utilizaremos un repositorio de formas de onda, *AKWFs (Adventure Kid WaveForms),* que contiene grabaciones .wav de periodos de onda para numerosos instrumentos. Estos archivos de audio son descargables, de forma gratuita desde la página web *AdventureKid*[14]. De esta forma obtendremos la tabla de ondas digitalizadas, base del proceso de síntesis.

[13] En un principio, tratamos de trabajar con la versión 7 de CCS pero no era compatible con los dispositivos de la serie MSP430FRxxxx.
[14] https://www.adventurekid.se/akrt/waveforms/adventure-kid-waveforms/

4 DESCRIPCIÓN HARDWARE

It's hardware that makes a machine fast.
It's software that makes a fast machine slow.
- Craig Bruce -

En este capítulo describiremos detalladamente los componentes integrados en la tarjeta PCB que constituye el soporte para nuestro sintetizador: Midisynth 2.0. La principal novedad respecto a la primera versión, Midisynth 1.0, será la sustitución del microcontrolador MSP430G2553 por otro de la misma familia, MSP430xxxxx, pero mayores prestaciones: el MSP430FR2355. Incorporaremos además en esta nueva placa un potenciómetro logarítmico, que permitirá al usuario controlar el volumen del dispositivo.

4.1 El microcontrolador MSP430FR2355

El microcontrolador MSP430FR2355 pertenece a la familia de microcontroladores de bajo consumo MSP430 de *Texas Instruments*. Constituido por una arquitectura de tipo RISC ("Reduced Instruction Set Computer") con instrucciones de 16 bits, puede alcanzar velocidades de operación de hasta 24 MHz.

Este MCU[15] ("MicroController Unit") incluye una amplia variedad de periféricos que le confieren gran versatilidad de aplicaciones. Cabe destacar: el sistema de reloj con oscilador interno; **4 temporizadores** de 16 bits, con registros de captura/comparación; sistemas de **comunicación serie UART, IrDA, SPI e I²C**; convertidores analógicos **ADC y DAC** de 12 bits; **44 puertos de entrada/salida**; y capacidad de memoria de 32KB FRAM para programas, 512 bytes FRAM para datos y 4KB de memoria RAM [9]. Las funcionalidades mencionadas se recogen en el diagrama de bloques de la figura 4-1:

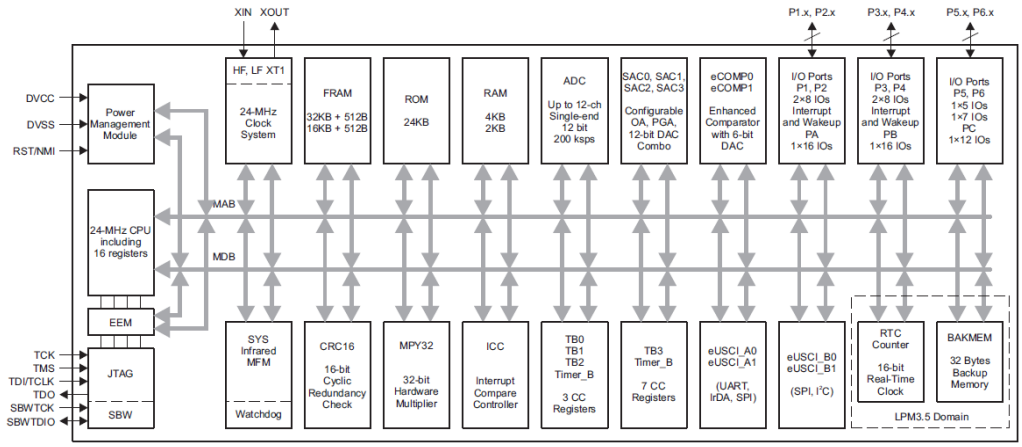


Figura 4-1. Diagrama de bloques funcional del MSP430FR2355[16]

[15] En lo sucesivo, denotaremos con frecuencia al microcontrolador como "MCU" por simplicidad.
[16] http://www.ti.com/lit/ds/symlink/msp430fr2355.pdf (MSP430FR2355 - datasheet). Fecha de consulta: Noviembre 2019

Todos los pines se encuentran multiplexados para lograr un diseño más compacto, de modo que cada uno de ellos puede tener hasta 5 funciones distintas.

El número de pines variará en función del encapsulado elegido; el fabricante TI ofrece 4 posibilidades para este dispositivo:

- 48-pin: LQFP
- 40-pin: VQFN
- 38-pin: TSSOP
- 32-pin: VQFN

En nuestro caso trabajaremos con el formato LQFP ("Low-profile Quad Flat Package") de 48 pines, que se muestra en la figura[17].

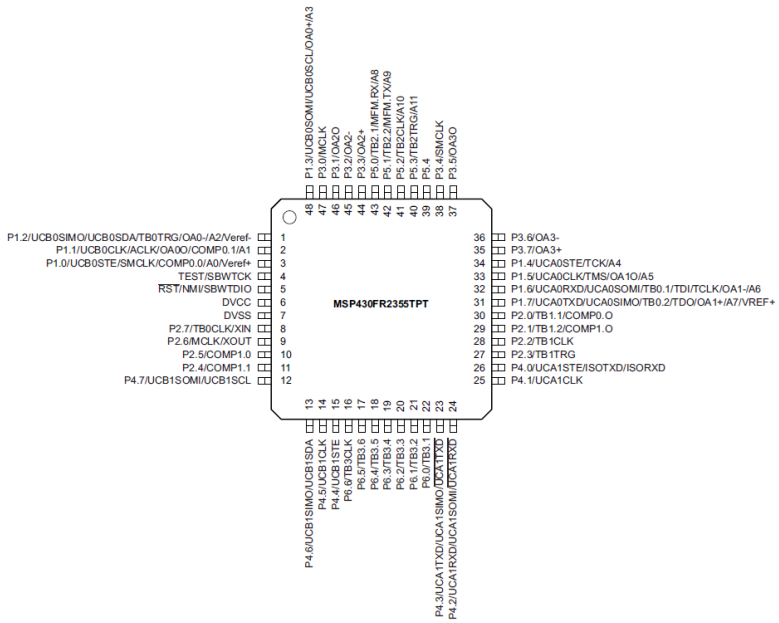


Figura 4-2. Disposición de los pines en el MSP430FR2355

4.1.1 MSP430G2553 vs MSP430FR2355

Son muchas las ventajas que observamos en el nuevo microcontrolador utilizado, no solo mayor disponibilidad de memoria sino funcionalidades adicionales, como el convertidor DAC, que vendrá a sustituir al PWM utilizado anteriormente como señal de salida del sintetizador. Además dispone de una librería de 20KB ROM que recoge funciones específicas del controlador, incluyendo transformadas FFT. Para no extendernos demasiado, nos centraremos en los cambios que afectan a nuestro sintetizador y las características aprovechadas para mejorarlo. En la tabla 5-1 resumimos dichas diferencias[18]:

Tabla 4-1. Comparativa de características entre los dos microcontroladores

	MSP430G2553	MSP430FR2355
Memoria No-volátil	FLASH 16kB	FRAM 16kB
Memoria volátil	RAM 512 Bytes	RAM 4kB
Convertidor ADC	Resolución: 10 bits	Resolución: 12 bits
Convertidor DAC	---	Resolución: 12 bits
I/O pins	24	48
Timers	2x 16-bit Timer_A	3x 16-bit Timer_B

[17] http://www.ti.com/lit/ds/symlink/msp430fr2355.pdf (MSP430FR2355 - datasheet). Fecha de consulta: Noviembre 2019
[18] Para mayor detalle, consultar los *datasheet* de ambos microncontroladores [13] [9].

En la tabla anterior, la referida memoria no-volátil comprende la memoria de código, los vectores de interrupción y las "signatures". La diferencia esencial entre ambas memorias es la tecnología de almacenamiento: la memoria FRAM, o RAM ferroeléctrica, mejora la velocidad de lectura/escritura de la FLASH, y precisa menor tensión de programación (~1,5V), disminuyendo así el consumo.

4.2 Tarjeta de desarrollo *Launchpad*

La placa MSP430FR2355 LaunchPad™ es una herramienta de desarrollo software enfocada al microcontrolador que utilizaremos. Se trata de una placa PCB que integra al MCU en el mencionado formato LQFP, y dispone de 48 pines, los cuales nos permitirán acceder a los distintos puertos del MCU. Además cuenta con un sensor de luminosidad, 2 pulsadores y 2 LEDs como interfaz sencilla para el usuario, un tercer pulsador con función de RESET, y un puerto de conexión USB para descargar programas en el dispositivo [10].

Para la programación del MCU, el Launchpad incorpora el emulador eZ-FET, que establece un puente de comunicación entre la interfaz USB y el puerto serie UART del microcontrolador. Esta herramienta permite la depuración del código en el propio hardware.

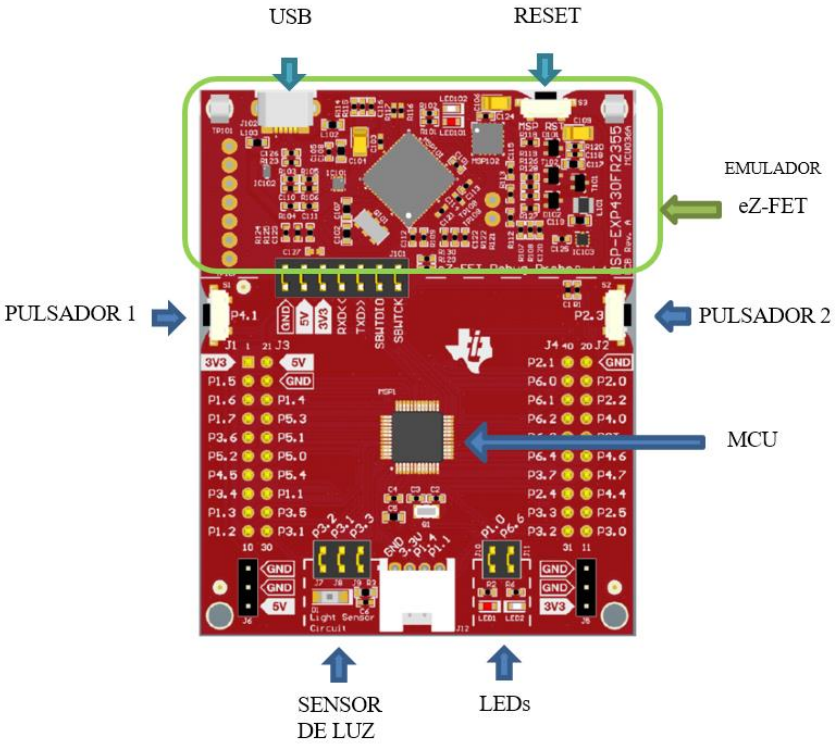


Figura 4-3. Tarjeta de desarrollo *MSP430FR2355 LaunchPad™*[19]

Los pines del MCU se asocian a los de la placa del siguiente modo: **40 pines se presentan en headers macho/hembra**, formando a cada lado de la placa 2 hileras de 10 pines, entre los que encontramos **36 pines I/O** (In/Out) de los 44 que tiene el microcontrolador. Los 4 pines restantes contituyen 2 **conexiones de alimentación a 3.3V y 5V** y **dos pines de tierra (GND)**. Se ilustran en la figura de la página siguiente las funciones asociadas a los 40 pines de los headers macho/hembra en la placa.

[19] http://www.ti.com/lit/ug/slau680/slau680.pdf (MSP430FR2355 Lunchpad Userguide). Fecha de consulta: Noviembre de 2019.

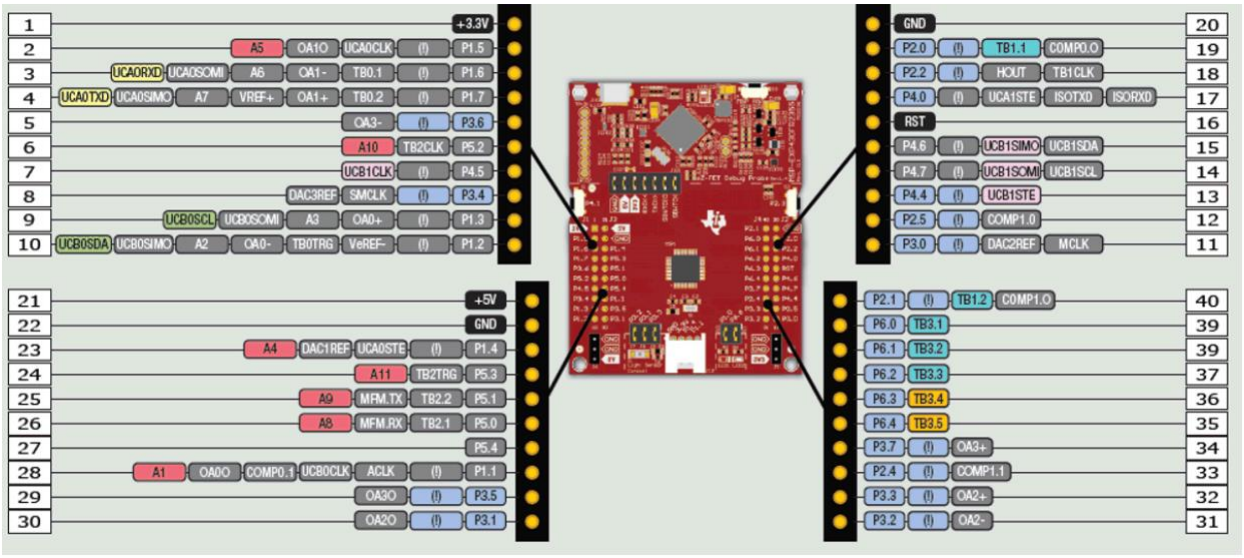


Figura 4-4. Pinout de los headers del Launchpad[20].

Como comentábamos en el párrafo anterior, a través de los headers solo podemos acceder a 36 puertos de los 44 del MCU. Los 8 pines restantes se encuentran "ocupados" por defecto para las funcionalidades del Launchpad especificadas anteriormente: 3 pines para los 3 pulsadores, 2 para los 2 LEDs, y 3 pines para el circuito del sensor de luz ambiente.

Excepto el pulsador de reset, estos elementos se encuentran en el circuito conectados a sus respectivos pines mediante *jumpers,* piezas de plástico que podremos quitar fácilmente en caso de necesitar acceder a sus puertos asociados. Los siete puertos en cuestión y sus funciones por defecto se citan a continuación:

- **P4.1** - Pulsador 1
- **P2.3** – Pulsador 2
- **P3.1, P3.2, P3.3** – Sensor de luz ambiente
- **P1.0** – LED 1
- **P6.6** – LED 2

4.3 Periféricos analógicos

Clasificamos los periféricos del dispositivo en función de la información que transmiten: analógica/ digital. Los periféricos analógicos operarán con valores de tensión entre 0 y su respectiva tensión de alimentación. La comunicación con el MCU se realizará gracias a los convertidores ADC y DAC integrados en el mismo.

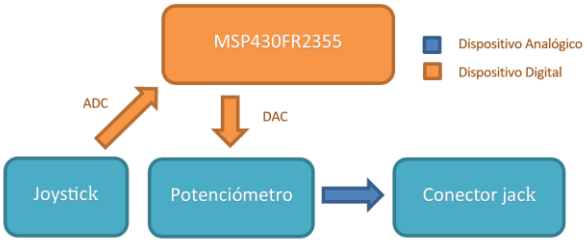


Figura 4-5. Flujo de información analógica.

[20] http://www.ti.com/lit/ug/slau680/slau680.pdf (MSP430FR2355 Lunchpad Userguide). Fecha de consulta: Noviembre de 2019.

4.3.1 Joystick de dos ejes

Un joystick analógico no es más que un **conjunto de resistencias variables** ensambladas, de manera que al mover la palanca en cada una de las direcciones posibles, varía el punto medio del potenciómetro asociado a su eje [11], que será la salida analógica. El joystick que utilizamos tiene **dos ejes, X e Y**, cuyos puntos extremos se conectan a alimentación, Vcc = 3.3V y a GND (0V), de modo que la **salida variará entre 0 y 3.3V**. Estos valores serán leídos por el convertidor ADC, cuya resolución es de 12 bits, esto es, un rango [0, 4095]. Por tanto, al desplazar el joystick el microcontrolador recibirá **valores escalados entre 0 y 4095**, para cada eje.

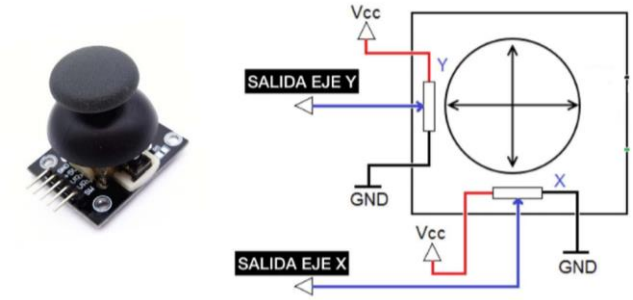


Figura 4-6. Esquema de conexión del joystick[21].

4.3.2 Potenciómetro logarítmico

El fundamento eléctrico de un potenciómetro es muy similar al del joystick: una resistencia, constituida por dos terminales fijos y un terminal variable, móvil, que se desplaza a través de la misma resistencia generando un divisor de tensión [12].

Para ajustar el volumen de un sistema de audio, como es nuestro propósito, los terminales fijos del potenciómetro se conectan a tierra y a la fuente de audio a regular, respectivamente, y el terminal variable se asocia a la salida, de manera que su valor se encontrará entre 0V (volumen mínimo) y el valor de tensión de la propia fuente *IN* (volumen máximo).

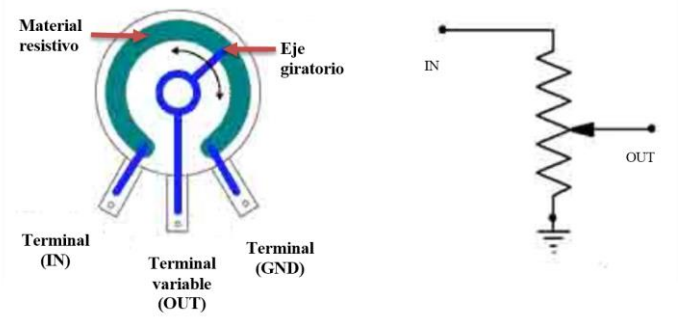


Figura 4-7. Esquema de un potenciómetro[22].

Recordemos que un potenciómetro puede ser lineal o logarítmico. En el lineal, la resistencia variará proporcionalmente al ángulo girado, mientras que, en el logarítmico, un pequeño giro del eje, corresponde una gran variación de la resistencia, que será mayor cuanto más giremos a la derecha el cursor del potenciómetro (función logarítmica). En sistemas de audio se suelen emplear estos últimos. El motivo es una "imperfección" de oído humano: cuando recibe determinada presión sonora por parte de un emisor, si queremos aumentar la

[21] https://www.robotica.school/curso/sensores-arduino-01/sensor-arduino-joystick-ky023/. Fecha de consulta: Noviembre de 2019.
[22] https://www.ecured.cu/Potenci%C3%B3metro. Fecha de consulta: Noviembre de 2019.

sensación auditiva al doble, por ejemplo, hay que aumentar el volumen mucho más del doble. Para traducir este efecto a un giro más o menos proporcional del eje del potenciómetro, se diseñaron estos dispositivos de respuesta logarítmica[23].

El valor del potenciómetro se expresa en ohmios (Ω), y representa el valor máximo de su resistencia variable. Para el sintetizador Midisynth, el potenciómetro elegido será de 10kΩ de tipo logarítmico.

Figura 4-8. Potenciómetro *log10k*

4.3.3 Conector de audio *jack*

Figura 4-9. Conectores tipo *jack*

La salida analógica regulada por el potenciómetro se conecta a un amplificador en modo seguidor de tensión (ganancia unidad). El amplificador utilizado será el TS482IDT, de ST Microelectronics. Esta etapa amplificadora nos permitirá conectar unos auriculares estándar al sintetizador, mediante un conector hembra ("*jack*") integrado en la PCB, como el mostrado en la figura 4-9.

4.4 Periféricos digitales

Los dispositivos digitales transfieren la información al microcontrolador directamente en forma binaria, es decir, como secuencias de pulsos de dos valores de tensión (nivel alto ó bajo), que se traducen en bits (valores lógicos, 0 ó 1).

4.4.1 Pantalla *Nokia 5110*

La información sobre el funcionamiento del sintetizador se mostrará mediante esta pantalla LCD, monocroma, de 48x84 píxeles. Se conectará al microcontrolador por puerto SPI, utilizando los dos pines necesarios la comunicación síncrona: el pin ***Din*** para la línea de transmisión de datos y el pin **clk** para la señal de sincronización.

Además tendremos dos pines de control, que se conectarán a puertos de propósito general (GPIO). **D/C** indicará el tipo de mensaje enviado, datos/comandos, con valores lógicos '0' ó '1'; y **CS** será la señal de *Chip Select,* que se activará cuando haya un mensaje en curso.

Figura 4-10. Pantalla LCD Nokia5110

El control del la pantalla es más complejo en comparación con resto de periféricos. Es por ello que se utiliza un *driver* específico, que será un conjunto de funciones predeterminadas que facilitan su funcionamiento, además de una tabla almacenada en memoria, que contiene los distintos caracteres ASCII codificados según el protocolo seguido para la escritura de los píxeles en la LCD.

Las 48 filas de píxeles se agruparán en 6 bancos de 8 píxeles (1Byte) cada uno, de modo que en el eje Y direccionaremos posiciones en el intervalo [0,5], y en el eje X tendremos direcciones [0,84]. Los mensajes de datos que recibirá el dispositivo serán de 1Byte, cada cual codificando los 8 bits de uno de los 6 bancos y determinada columna, de las 84 posibles. Dividiremos el eje X, por comodidad, en 14 columnas direccionables, de forma que cada carácter ocupe 6 píxeles a lo largo de dicho eje: 84/14 = 6. En el eje Y los caracteres ocupan un banco completo, 8 bits.

[23] https://www.forosdeelectronica.com/threads/diferencia-potenciometro-lineal-y-logaritmico.85869/. Fecha de consulta: Septiembre de 2019.

Para codificar un carácter, se enviarán 5 mensajes de 8 bits, correspondientes a las 5 columnas de un banco determinado, y la 6ª columna de espaciado. Dichos mensajes contendrán valores lógicos, '0' ó '1', según el píxel indicado se encuentre, respectivamente, "apagado" ó "encendido"(coloreado en negro). En la figura se señala el espacio ocupado por varios caracteres [14].

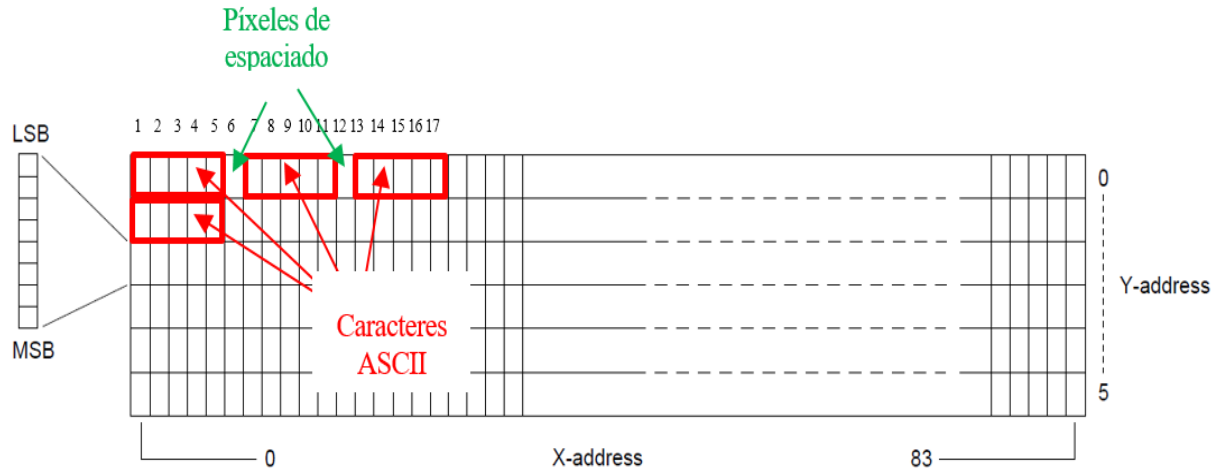


Figura 4-11. Distribución de los caracteres en la matriz de píxeles.

4.4.2 Pulsadores

Los botones para el manejo del sintetizador serán activos a nivel bajo, de modo que al pulsarlos enviarán un '0' lógico por el puerto asociado correspondiente. Se configuran, por tanto, con resistencias de pull-up, para que se encuentren a nivel alto por defecto.

Figura 4-12. Pulsadores utilizados.

4.4.3 Encoder rotativo

Figura 4-13. Encoder rotativo.

Este dispositivo se conectará al microcontrolador mediante dos pines de GPIO, que denotaremos como **canales A y B**. A través de estos canales se generarán dos **trenes de pulsos desfasados, que permitirán detectar el ángulo y el sentido de giro** [11]. Un tercer pin se conecta a tierra, GND.

Al producirse un giro se emitirán flancos por ambos canales, con una precisión de 24 pulsos por vuelta completa. En función del número de flancos, conoceremos los grados girados, por tanto cada flanco implicará un giro de 360º/24 = 15º.

El giro podrá ser en sentido de las agujas del reloj, que identificaremos como **CW**(*"Clock Wise"),* o en sentido contrario, **CCW**(*"Contrary Clock Wise").*

Tomando como referencia uno de los canales, por ejemplo el canal A, y **fijándonos en sus flancos, se observa que cuando el giro es CW, la señal A es siempr**e inversa a la B, mientras que si giramos CCW,

ambas señales lógicas (‘0’ ó ‘1’) serán idénticas [13]. Obsérvese la figura 4-14 de la página siguiente, que ilustra este comportamiento lógico.

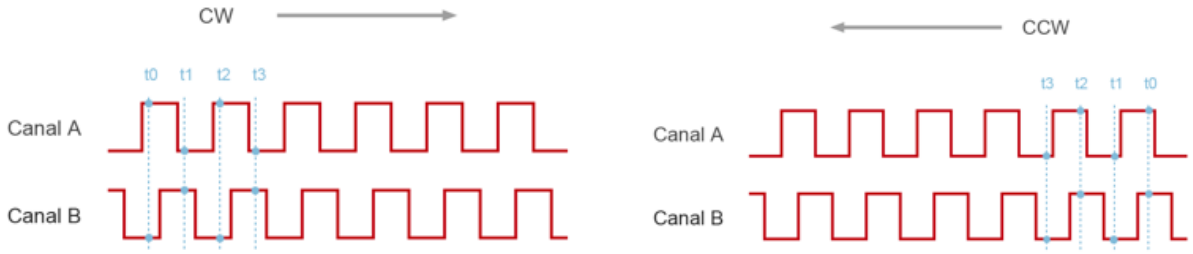


Figura 4-14. Señales generadas por el encoder, en función del sentido de giro [13].

4.4.4 Entrada MIDI

Como comentamos en el capítulo teórico, para comunicaciones MIDI se emplean conectores tipo DIN de 180º. Para la entrada de mensajes MIDI, que constituirán el control del sintetizador, utilizaremos el modelo de conector hembra *MAB5SH*[24]. Se dispondrá junto al conector un optoacoplador para aislar la entrada, protegiendo así el equipo ante posibles picos de voltaje: modelo *Isocom 6N139* de 8 pines.

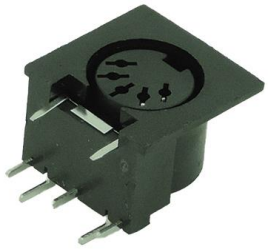


Figura 4-15. Conector MIDI hembra[25].

[24] Para más información consúltese datasheet: https://datasheet.octopart.com/MAB-5-SH-Hirschmann-datasheet-14707087.pdf. Última consulta: Agosto de 2019.
[25] Imagen tomada de: http://www.electronicajapon.com/product_info.php?products_id=271. Fecha de consulta: Diciembre de 2019.

5 PROGRAMACIÓN DEL MICROCONTROLADOR

To me,
the original VCS3 synthesizer is like a Stradivarius.
- Jean-Michel Jarre -

Este capítulo representa la etapa del proyecto que ha supuesto más horas de estudio, la programación del sintetizador en el entorno Code Composer, si bien procuraremos que la explicación de dicha tarea sea breve para no extendernos mucho en el texto.

En primer lugar, se explicará el **modo de operación** del sintetizador así como la interfaz para su control por parte del usuario. Esbozaremos a continuación el funcionamiento general del programa, esquematizando en distintas etapas las tareas a realizar por el bucle principal. Mediante un **pseudocódigo** sencillo, desglosaremos seguidamente tales etapas de funcionamiento.

Por otro lado, se expondrá la **estructura del código,** comentando brevemente los distintos módulos de los que se compone. Además, enunciaremos las funciones del programa, junto a la finalidad de las instrucciones que se ejecutan en cada una de ellas.

En una última sección se aclararán algunos detalles sobre las subrutinas del programa, necesarias para la comprensión del código completo, adjunto en el Anexo I. Se remarcan también en el mismo apartado las modificaciones más significativas que debimos realizar sobre el código de partida, *Midisynth.1.0.*

5.1 Especificaciones de funcionamiento

El sintetizador opera del siguiente modo. Accionando un teclado electrónico, que será el controlador de nuestro dispositivo, las notas pulsadas se envían codificadas en **mensajes MIDI** al sintetizador. Dichas notas se emitirán con cierto timbre, configurado mediante los **pulsadores**, como mezcla de los sonidos típicos de 8 instrumentos almacenados[26].

Cada instrumento, llevará asociado un desfase determinado, configurable mediante el giro del **encoder**. El desfase representará el desplazamiento en el eje X de la onda asociada a dicho instrumento, a la hora de sumar todas las ondas seleccionadas para la síntesis aditiva.

Contamos con 4 pulsadores: el pulsador superior aumenta el índice del instrumento mostrado por pantalla, el segundo de ellos añade dicho instrumento a la suma, el tercero decrementa el instrumento mostrado[27], y el cuarto, el inferior, permitirá seleccionar el efecto de modulación (trémolo, vibrato, *ADSR* ó *FM*). Por otro lado, la pulsación del encoder activará la onda asociada al instrumento *actual* como envolvente para el *LFO.*

Para que la modulación seleccionada se haga efectiva, debe ser activada mediante el **joystick**: si lo

[26] Ver aclaración sobre los "instrumentos" en la sección 5.2.3.
[27] En lo sucesivo, se denota como instrumento "actual", el que aparece mostrado en la LCD, como seleccionable para la modulación ó la suma.

desplazamos hacia la izquierda, "desconectaremos" la modulación, produciendo el sonido sin efecto alguno, mientras que si lo movemos al extremo derecho "conectaremos" la envolvente. Además, el desplazamiento del joystick en su eje vertical nos permitirá obtener un efecto *pitch,* como el que comentábamos en el capítulo 2.

Por último, el **potenciómetro** situado en la esquina inferior izquierda permitirá al usuario controlar el nivel de volumen.

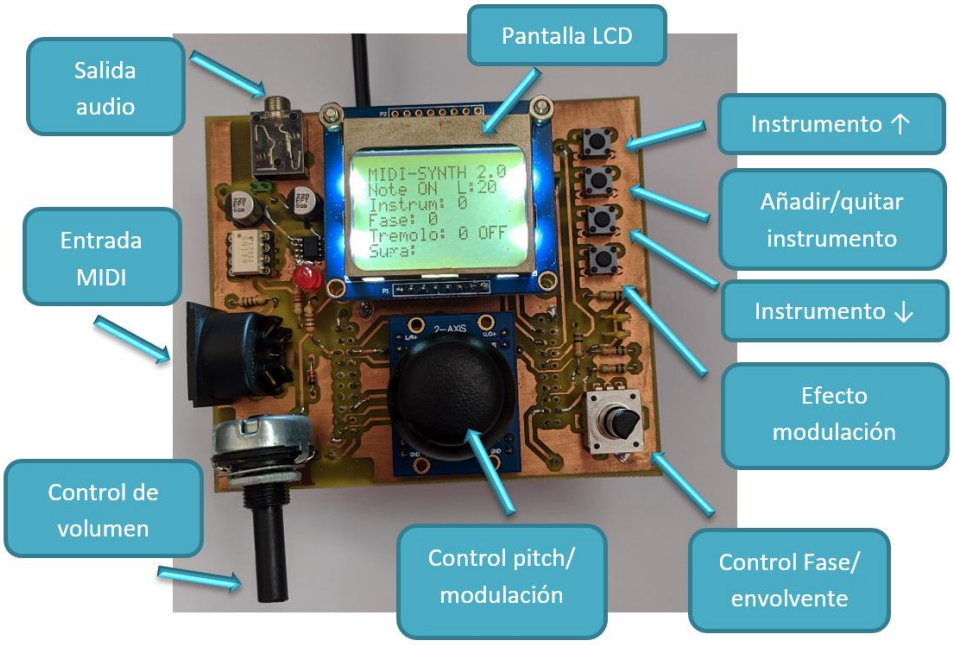


Figura 5-1. Controles del sintetizador.

En la pantalla se mostrarán los distintos parámetros configurables por el usuario, con este orden:

- Línea 1: Título **"MIDI-SYNTH 2.0"** y valor de la carga, **L**[28].
- Línea 2: índice del **instrumento actual.**
- Línea 3: **desfase** de la onda asociada al instrumento actual.
- Línea 4: **efecto** de modulación y **estado** de la modulación.
- Línea 5: **instrumentos sumados** para la síntesis de la onda.

5.2 Etapas del programa

Este apartado será meramente ilustrativo, para mejor comprensión por parte del lector del código del programa. Haremos una clasificación del conjunto de instrucciones del programa principal, *main*, atendiendo a las tareas que implementa. No obstante, aunque seguiremos el mismo orden de ejecución del programa, en el código fuente real no se distinguen claramente las secciones y bloques de instrucciones que comentamos.

Como puede observarse en la figura 5-1, tras una etapa de configuración inicial de los periféricos, el programa entra en un bucle infinito, *while (1)*, ejecutando 6 etapas de forma continua, señaladas en tono azul.

[28] La carga representa el tiempo empleado en la ejecución de la interrupción como porcentaje del intervalo de tiempo que transcurre entre cada llamada del programa a la interrupción (20kHz). Se considera un funcionamiento no deseado cuando la carga sobrepasa el 100%, en cuyo caso, se estarán solapando las interrupciones sucesivas del DCO.

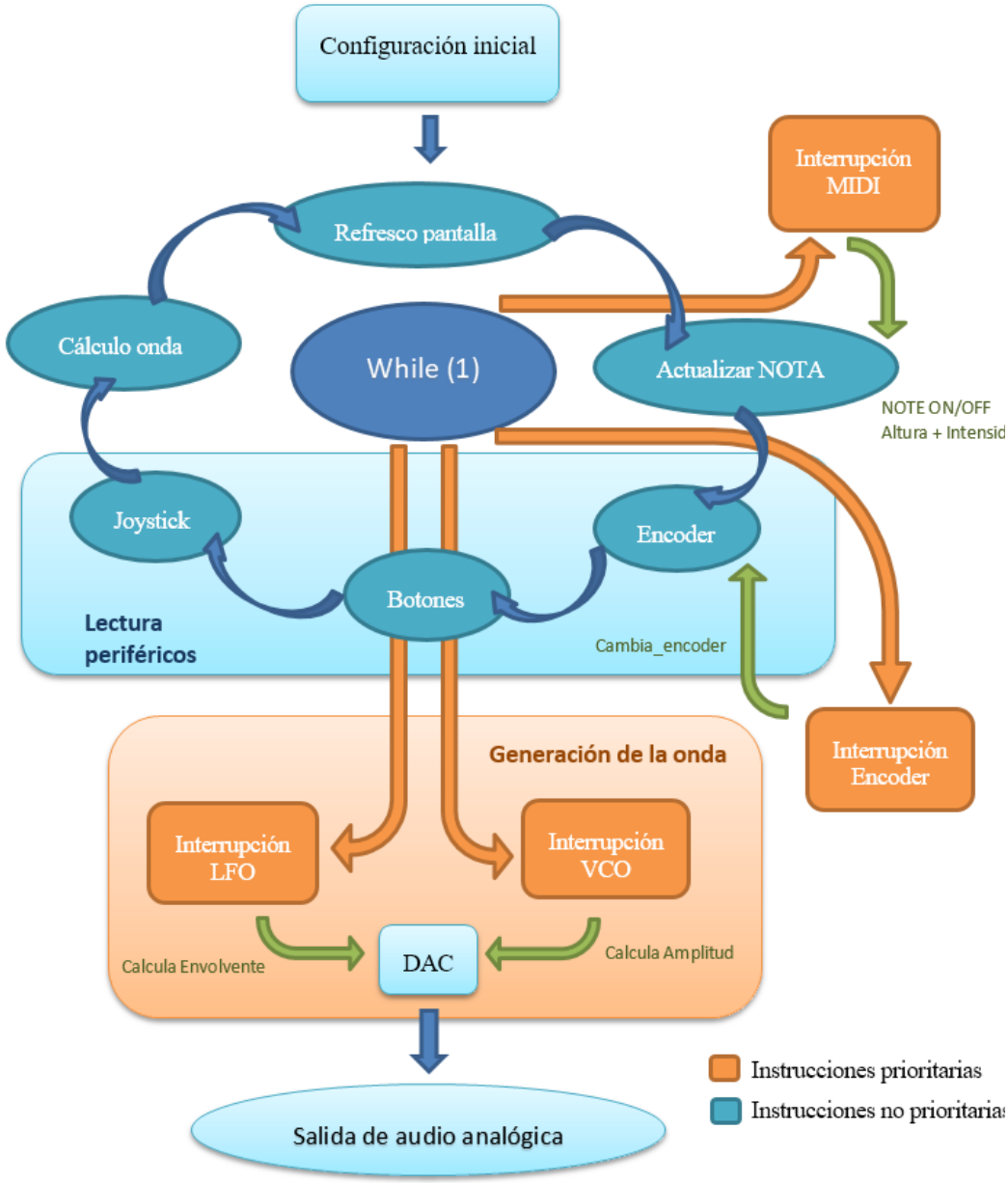


Figura 5-2. Esquema general del programa.

Los bloques naranjas representan las rutinas que interrumpen el funcionamiento de dicho bucle principal, dos de ellas ocurren de forma temporizada, son "esperadas", y otras dos dependen de la interacción con el usuario, pueden saltar en cualquier momento de la ejecución:

1. **Interrupción DCO:** rutina temporizada que se genera con frecuencia de 20kHz, calculando la amplitud, en el intervalo [0, 4095] de cada uno de los puntos que conforman la onda de salida. Dicha amplitud es modificada ligeramente por la envolvente que calcula la interrupción LFO, y es finalmente enviada al convertidor DAC.
2. **Interrupción LFO:** rutina programada a 100kHz, es decir, salta una vez por cada 5 interrupciones del DCO, generando los puntos de la onda envolvente.
3. **Interrupción de la entrada MIDI:** se ejecuta cuando se detecta la llegada de mensajes MIDI por el puerto de comunicación serie UART. En esta rutina se almacenan los mensajes *note ON/OFF* y la información de altura/velocidad para las notas, datos que son tratados posteriormente en la etapa de actualización de la nota.

4. **Interrupción del encoder:** "salta" únicamente cuando el usuario acciona este periférico. Se guarda la información recogida del giro del encoder, y se activa una bandera para avisar a la rutina de lectura de los cambios producidos, que deberá procesar en su momento de ejecución.

Cabe decir, que las flechas verdes asociadas a las interrupciones NO indican el orden de ejecución, sino el flujo de información con los bloques apuntados. El programa *saltará* a las interrupciones en cualquier punto del bucle principal (flechas naranjas), y al terminar volverá a la misma línea de ejecución en la que se encontraba.

En los subapartados siguientes se explica mediante pseudocódigo, las acciones ejecutadas en cada etapa de funcionamiento.

5.2.1 Configuración inicial

Las primeras líneas del programa contienen instrucciones para la configuración de los distintos periféricos: timers, convertidores, puertos de entrada/salida, comunicación serie… Estos se resumen a continuación, siguiendo su mismo orden de ejecución:

- Desactivar el *watchdog.*
- Configurar reloj a 16 MHz.
- Configurar puertos de E/S.
- Configurar comunicación serie asíncrona SPI para el control de la pantalla.
- Configurar el módulo USCI_A en modo UART para comunicación serie asíncrona con el periférico de entrada *MIDI.*
- Configurar los timers TB1 y TB2 para las interrupciones de generación de la onda de salida (*DCO* y *LFO*).
- Configurar convertidores *ADC* y *DAC.*
- Establecer frecuencia inicial de modulación por defecto: frecuencia para modulación *tremolo.*
- Asignar valores iniciales a los parámetros ADSR.
- Mostrar por pantalla la interfaz para el usuario, con los **valores de parámetros por defecto, en negrita:**
  - Línea 1: Título **"MIDI-SYNTH 2.0"** y valor de la carga, L.
  - Línea 2: índice del instrumento actual → **0.**
  - Línea 3: desfase de la onda asociada al instrumento actual[29] → **0.**
  - Línea 4: efecto de modulación → **Tremolo;** estado de la modulación → ON/**OFF.**
  - Línea 5: instrumentos sumados para la síntesis de la onda → **NINGUNO.**
- Activar las interrupciones.

5.2.2 Refresco de pantalla

- En la 1ª línea, actualizar el valor de la carga, L.
- En la 3ª línea, sobreescribir el desfase actual.
- En la 4ª línea, sobreescribir el efecto de modulación que se encuentre activo.

[29] Para la modulación FM, en la tercera línea de la LCD se indicará el índice de modulación en lugar del desfase.

5.2.3 Comprobación de la nota musical – Interrupción MIDI

Los mensajes recibidos por la interrupción *MIDI* llegarán en "paquetes" de 3, en el siguiente orden: **note on/off - altura - velocidad.** Esta información se almacena para ser procesada en la etapa que denotamos como "comprobación de la nota". En este paso, se comprueba si hay algún mensaje pendiente. En caso afirmativo, la bandera `Msg` se encontrará activa:

- Si el mensaje indica que se ha presionado una tecla:
  - Mostrar por pantalla "`NOTE_ON`".
  - Actualizar la nota actual almacenada: `nota_act = altura-OFFSET_NOTA`

    Donde la variable **nota_act será un entero que representa el índice del vector de frecuencias normalizadas**, donde encontraremos el valor de frecuencia correspondiente a dicha nota. El offset es una constante, igual a 36, valor de la primera "altura" contenida en nuestro vector de frecuencias normalizadas, de las 128 posibles enviadas por protocolo MIDI.
- Si la tecla se ha liberado:
  - Escribir "`NOTE-OFF`".

Por otro lado, la **rutina de interrupción MIDI** funciona de la forma siguiente: cada vez que llega un mensaje por el puerto serie de comunicación *UART,* se ordena la ejecución de la función **int_rx ().** Dicha función procede según esta casuística:

- Si la palabra leída por el búfer es **"10010000" (NOTE OFF),** deberemos "apagar" la nota que estaba sonando:
  - Si la modulación es ADSR, indicar el comienzo de la fase D *(decay),* mediante la bandera: `nota = 2;`
  - En caso contrario, simplemente avisar el fin de nota: `nota = 0;`
  - Activar bandera de mensaje completo: `Msg = 1`.
  - Resetear contador de mensajes: `n = 0.`
- Si la palabra es **"1000 0000" (NOTE ON),** mensaje 1/3 de cada nota:
  - Activar contador de mensajes: `n = 1.`
- Si el primer mensaje ya llegó, y estamos a la espera de los siguientes (`n=1` ó `n=2`):
  - Si llega el 2º mensaje, correspondiente a la **altur**a de la nota (`n=1`):
    - Copiar en la variable *altura* el valor del búfer de la línea de comunicación UART.
    - Incrementar el contador de mensajes: `n = 2.`
  - Si llega el 3er mensaje, que contiene la **velocidad** de ataque (`n = 2`)
    - Copiar en la variable *velocidad* el valor del búfer.
    - Resetear contador de mensajes: `n = 0.`
    - Activar bandera de mensaje completo y bandera de nota nueva: `nota = 1`

5.2.4 Lectura periféricos

5.2.4.1 Encoder – Interrupción encoder

La interrupción del encoder será generada, bien por detectarse un cambio a nivel bajo en la entrada del puerto P2.4, al que denotamos como *Canal A,* ó bien por una variación del mismo tipo en P2.5, que será el

Canal B. Tomamos como referencia de giro el *Canal A*. Cuando el programa "salta" a la **función de interrupción**, ejecuta las instrucciones que siguen:

- Si la interrupción ha sido ocasionada por el puerto P2.4 (P2.4 = '0'):
  - Si la señal A es inversa a la B (P2.5 = '1'), se detecta un giro a la derecha: incrementar el contador del encoder.
  - En caso contrario, si los niveles lógicos de A y B son coincidentes (P2.5 = '0'), tendremos un giro a la izquierda: restar 1 unidad al contador del encoder.
  - Si el contador del encoder alcanza valor igual a (+3):
    - Activar la bandera *cambia_encoder* de forma positiva: `cambia_encoder = 1;`
    - Resetear el contador del encoder.
    - Si la variable del encoder es menor que 30000, sumarle 1 unidad.
  - Si el contador toma valor igual a (-3):
    - Activar la bandera *cambia_encoder* de forma negativa: `cambia_encoder = -1;`
    - Resetear el contador del encoder.
    - Si la variable del encoder es mayor que (-30000), restarle 1 unidad.

En el bucle principal, durante la etapa dedicada a la lectura del encoder, se comprueba si está activa la bandera `cambia_encoder`. En dicho caso, variaremos el parámetro que corresponda:

- Si tenemos **modulación *FM***, variar el **índice de modulación**, dentro del rango [-12, +12].
- En cambio, si la **modulación actual es la *ADSR***, procedemos en función del ***parámetro a modificar***, seleccionable mediante la pulsación del encoder. Podrá modificarse:
  - La fase relativa, en el rango [-63, +63].
  - Los parámetros característicos ADSR: tiempo de ataque, ***tA;*** tiempo de decaimiento, ***tD;*** nivel de sostenido, ***SUST;*** y tiempo de relajación, o liberación de la nota, ***tR.*** Todas estas variables saturarán fuera del intervalo [1,100].

    Si se modifica alguna de las 4 variables anteriores, actualizar seguidamente los valores de las respectivas pendientes de la amplitud, durante las 4 fases de modulación[30]. Las divisiones se realizarán en punto flotante, para mayor exactitud:

    - Amplitud fase **A** *("Atack")* =

$$= \frac{Amplitud\ alcanzada\ al\ final\ de\ la\ fase\ A}{Duración\ de\ la\ fase\ A} = \frac{1000}{tA}$$

    - Amplitud fase **D** *("Decay")* =

$$= \frac{Amplitud\ final\ fase\ A - Amplitud\ final\ fase\ D}{Duración\ de\ la\ fase\ D} = \frac{1000 - 10 * SUST(\%)}{tD}$$

[30] Nótese que la amplitud de la fase de sostenido (S) es el propio valor del parámetro SUST, como porcentaje de la amplitud total, 1000.0, por lo que no precisa ser actualizarla en este punto.

- Amplitud fase **R** *("Release")* =

$$= \frac{Amplitud\ final\ fase\ S\ (constante) - Amplitud\ final\ fase\ R(0)}{Duración\ de\ la\ fase\ R} = \frac{10 * SUST(\%)}{tR}$$

- Por último, si la modulación elegida no es la FM ni la ADSR, lógicamente tendremos ó bien un **trémolo ó bien un efecto vibrato**, y procederemos variando la **fase relativa del *instrumento*** *actual,* siempre que se encuentre en el rango [-63,+63].
- Una vez contemplados los 4 posibles efectos de modulación, mostrar en la 4ª línea de la pantalla el valor del parámetro que se acaba de actualizar.

5.2.4.2 Pulsadores

Contamos con cinco botones de control: la hilera de cuatro pulsadores activos a nivel bajo, más el pulsador del encoder. Explicamos sus funciones en orden de su posición en la placa, de arriba abajo.

- Si se ha pulsado el **botón superior**, y el índice del instrumento actual, mostrado por pantalla, es menor que 8 (el último):
  - Incrementar el instrumento actual y escribir en la 2ª línea su valor actualizado.
- Al pulsarse el **segundo botón**:
  - Si el instrumento en cuestión no se encuentra ya sumado en la onda aditiva:
    - Añadirlo a la suma.
    - Si tenemos **modulación *ADSR***, establecer la **fase relativa** asociada al instrumento recién sumado como la fase relativa actual (la que aparece en la línea 3ª de la LCD).
    - De lo contrario, si la **modulación es del tipo *tremolo, vibrato o FM:***
      - Si la fase relativa actual es mayor ó igual que 0, dar dicho valor a la fase asociada al instrumento en cuestión.
      - Si la fase actual es negativa, asociar al instrumento la fase *(64 – fase relativa),* de forma que se compense el signo negativo, y tome un valor > 64. Como la onda muestreada tiene longitud de 64 puntos, esta operación será equivalente.
  - Si ocurre que el instrumento **ya formaba parte de la suma, eliminarlo de la onda aditiva.**
  - Activar la **bandera de cálculo de onda**, para regenerarla atendiendo a la suma de instrumentos actualizada.
- Si se detecta activo el **tercer botón**, actuar de modo similar que con el primer pulsador, disminuyendo en este caso el índice del instrumento y actualizar su valor en la 2ª línea de la LCD.
- De encontrarse pulsado el **cuarto botón**, proceder a cambiar el efecto de modulación:
  - Resetear la fase de modulación.
  - En función de la modulación actual, cambiar al efecto siguiente, siguiendo el orden: **TREMOLO – VIBRATO – ADSR - FM.**
- En último caso, si se detecta la **pulsación del encoder** (codificado como *botón 5):*
  - Si la envolvente actual es de tipo **ADSR**, cambiar el parámetro actual del encoder, siguiendo el orden **FASE-tA-tD-SUST-tR.**

- Para cualquier otro efecto de modulación, establecer como onda envolvente la asociada al instrumento actual, y actualizar este cambio en la 3ª línea de la pantalla.

5.2.4.3 Joystick

Desde el bucle principal se ordena la lectura de ambos potenciómetros del joystick, correspondientes a los ejes X e Y, cada uno de ellos asociado a un canal del convertidor ADC:

- Si se ha producido un evento relacionado con el **eje Y:**
  - Guardar en la variable *desplazamiento* la información leída por el canal ADC asociado al eje Y, escalada en el intervalo [0,64]. Este valor numérico representa realmente su posición, no un desplazamiento relativo.
  - Si el valor de la posición es mayor al del punto medio del eje ("31"), identificamos un **desplazamiento hacia arriba:**
    - Si el índice asociado a la **nota actual es menor ó igual que 54**[31], aún podemos subir 2 semitonos la nota actual: establecer como frecuencia aquella asociada a la nota actual más 1 ó 2 semitonos, en función del mayor ó menor desplazamiento del joystick hacia arriba.
  - Si la variable *desplazamiento* es menor que 31, el joystick se mueve **hacia abajo:**
    - Si el índice de **la nota actual es mayor ó igual a 2**, todavía será posible restarle hasta dos semitonos a dicha nota, procediendo igual que en el caso anterior.
- En caso de detectarse un cambio en la posición horizontal del joystick, **eje X:**
  - Si el valor de la posición, leído por el canal ADC asociado al eje X[32] es menor que 40, se habrá producido un desplazamiento al **extremo izquierdo: activar la modulación** e indicarlo por pantalla, mediante la escritura de la cadena **"ON"** junto al tipo de modulación, en la 4ª línea.
  - Si la posición horizontal es mayor que 4000 unidades, identificamos un movimiento hacia el **punto extremo derecho: desactivar el uso de la envolvente** para el cálculo de la amplitud en el DCO, y escribir en la 4ª línea la cadena **"OFF"** junto al efecto actual de modulación, que estará desactivado.

5.2.5 Cálculo forma de onda aditiva

Si los instrumentos que intervienen en la síntesis se modifican, será menester rehacer los cálculos de la onda aditiva:

- Resetear contador de instrumentos sumados (nº de ondas que intervienen en la síntesis aditiva).
  - Para cada uno de los 8 instrumentos posibles:
    - Si el instrumento en cuestión forma parte de la suma (esta información se lee del vector de instrumentos a sumar):
      - Escribir el valor de su índice en el campo "suma" de la pantalla e incrementar el contador de instrumentos sumados.
- La forma de la onda aditiva se almacena en un vector de 64 puntos. Cada uno de estos puntos se calcula ejecutando este bucle:
  - Resetear valor del punto.
  - Para cada uno de los instrumentos activos (1-8), tomar la altura de la onda correspondiente,

[31] Si tenemos una nota inferior a la nota más aguda de la tabla (última posición en el vector, "56") menos dos, es decir, si el índice actual es menor que 54, podemos aún desplazarnos dos posiciones adelante en el vector de frecuencias.
[32] Valor comprendido en el intervalo [0, 4095], por ser la resolución del ADC de 12 bits. En este caso no realizamos ningún escalado.

en el mismo punto del eje X contemplado y desplazado con el valor de su fase asociada, y sumarlo al total de la altura que calculamos.

- Escalar la altura total del punto dividiendo su valor entre el número de instrumentos que hemos sumado.

5.2.6 Generación de la onda analógica de salida

5.2.6.1 Interrupción LFO

El objetivo de esta función de interrupción será el cálculo de la onda envolvente que se emplea en los distintos efectos de modulación. Nótese que en el caso de la modulación en frecuencia, FM, no se realiza cálculo alguno; como comentamos en el capítulo teórico, el efecto FM se trata de un vibrato muy rápido, de ahí que las variaciones de la onda original se realizan a la frecuencia del DCO, en la función de interrupción correspondiente.

- Para el efecto trémolo:
  - La **envolvente será la onda asociada al instrumento modulador, dilatada en el eje X,** multiplicando por 256 la frecuencia de modulación. Además, **escalarla en altura,** dividiendo su valor entre 4 y sumándole un offset de 64.
  - **Incrementar la fase de modulació**n según la frecuencia de modulación.
- Si deseamos obtener un vibrato:
  - Calcular la **envolvente de igual modo que indicamos para la modulación trémolo**.
  - Obtener la **frecuencia de modulación** como el valor calculado de envolvente sumado a la frecuencia de la nota actual.
  - **Incrementar la fase de modulación** según la frecuencia de modulación.
- En el caso de tener modulación ADSR:
  - En función de la fase en la que nos encontremos**, incrementar la amplitud actual de la envolvente ADSR sumándole el valor de la pendiente asociada** (el cálculo de dichas pendientes se realizó en el **apartado 5.2.4.1**).
  - Escalar la amplitud calculada, de tipo flotante, y convertirla a tipo entero.

5.2.6.2 Interrupción DCO

- Resetear el valor de salida del DAC.
- Si hay una nota sonando (`nota != 0`):
  - Leer del vector de la onda aditiva el punto correspondiente, según la `fase` actual, y asociarlo a la variable de salida del DAC.
  - Para cualquier modulación distinta de la ADSR, escalar la altura del DAC en función de la `Fuerza` de ataque de la nota.
  - De tener modulación trémolo:
    - Incrementar la fase: nos desplazamos en el eje X un periodo de la frecuencia de la nota actual.
    - Si la modulación está activa, escalar el valor de salida con la onda envolvente.
  - Si el efecto seleccionado es el vibrato:
    - Si la modulación se encuentra activada, incrementar la fase en función de la frecuencia de modulación.
    - En caso contrario, aumentar la fase atendiendo a la frecuencia de la nota que se está

ejecutando.

- Cuando se selecciona la modulación en frecuencia, FM:
  - Si tenemos conectada la modulación:
    - La variación de frecuencia a cada paso, variable `FM`, se lee de la tabla de ondas; será la altura de la onda asociada al instrumento modulador, en el punto del eje X correspondiente a la fase de modulación.
    - Actualizar la frecuencia de modulación, que resultará de sumar a la frecuencia de la nota la variación `FM` calculada anteriormente.
    - Sumar a la fase actual la frecuencia de modulación.
    - Sumar a la fase de modulación la frecuencia asociada a la nota actual desplazada con el índice de modulación, en el vector de frecuencias normalizadas.
  - Para el caso sin modulación, simplemente incrementar la fase actual con la frecuencia de la nota.
- Por último, si contemplamos la modulación ADSR:
  - Con modulación activa, escalar la altura de la onda de salida con la amplitud ADSR calculada.
  - De lo contrario, sumar a la fase actual un periodo de la nota a producir.

- Sumar a la onda final de salida un offset de valor 400 y multiplicar este valor por 5. Esta operación se lleva a cabo **para escalar correctamente la onda, y que se adapte lo mejor posible al rango [0, 4095]. El ajuste realizado radica en que la resolución del DAC es de 12 bits**, mientras que la onda que generábamos mediante PWM contaba solo con 8 bits de resolución.
- Finalmente, calcular la carga, L, como porcentaje del tiempo consumido por la interrupción sobre 800, que es el tiempo transcurrido entre interrupciones sucesivas (timer B_2).

5.3 Estructura del código

El programa C *Midisynth 2.0*, se basa en los siguientes ficheros fuente, los cuales comentaremos brevemente en las próximas secciones:

1. midisynth.c
2. midisynth.h
3. formas.h
4. nokia5110.c
5. main.c

5.3.1 Midisynth.c

Es una librería con funciones para el manejo del sintetizador. Incluye las instrucciones de inicialización de los puertos de entrada/salida, así como las subrutinas de los periféricos y las funciones de interrupción de los mismos[33]. En la tabla se enumeran los prototipos de las funciones que define, junto a la finalidad de cada una de ellas.

[33] Solo el encoder y la entrada MIDI trabajan con interrupciones.

Tabla 5-1. Funciones incluidas en Midisynth.c

	Prototipo función	Finalidad
Funciones de Inicialización	void **conf_reloj** (char VEL)	Configuración reloj del MCU
	void **Software_Trim** (void)	Calibración del DCO[34]
	void **conf_puertos** (void)	Configuración puertos E/S
	void **conf_MIDI** (void)	Configuración del módulo USCI_A en modo UART
	void **conf_timers** (void)	Configuración Timer B2 -> DAC y Timer B1 -> moduladora
	void **conf_ADC** (void)	Configuración del ADC para entrada analógica del joystick
	void **conf_DAC_SAC**(void)	Configuración del DAC para salida analógica de audio
Subrutinas de los periféricos	void **lee_joystick** (void)	Ordena lectura canales ADC -> *leecanal (i)* Ordena filtrado de resultados -> *filtra()* Actualiza valor almacenado y banderas *cambio_x, cambio_y x*
	int **leecanal** (int ch)	Lectura de canal ADC
	void **filtra** (void)	Para cada canal, hace la media de las 4 últimas lecturas.
	void **lee_botones** (void)	Comprobación de los 5 pulsadores (incluido pulsador encoder)
	void **int_rx** (void)	Lectura entrada MIDI (activa por interrupciones)
Funciones de interrupción	__interrupt void **USCI0RX_ISR_HOOK**(void)	Ordena lectura búfer UART -> *int_rx ()* Se activa la interrupción cada vez que llega un mensaje
	__interrupt void **Encoder** (void)	Lectura del Encoder Se activa cada vez que se detecta un giro; activa bandera *cambio_encoder* y actualiza valor almacenado

5.3.2 Midisynth.h

Cabecera de la librería anterior. Se incluye en el fichero *main.c,* ya que recoge los prototipos de funciones que éste utiliza, además de variables **globales** y los *macros* del programa.

[34] DCO: "Digitally Controlled Oscillator".

5.3.3 Formas.h

Esta otra cabecera contiene la **tabla de ondas**, fundamento de la técnica de síntesis empleada. Además define ciertas constantes del sintetizador, como el vector de frecuencias normalizadas.

La tabla de ondas es una **matriz de 8 vectores**, por tanto, en función del vector leído en la tabla lograremos hasta 8 timbres distintos de sonido. Denotaremos estos timbres como **instrumentos 1-8**, si bien no se correponden a instrumentos físicos reales[35]. Cada vector almacena un periodo de cierta onda. Se toman **64 muestras por cada forma de onda**[36], es decir, los vectores contendrán 64 valores enteros, siendo la tabla una matriz 8x64.

5.3.4 Nokia5110.c

En este fichero encontramos las funciones empleadas para el manejo de la pantalla LCD: instrucciones de inicialización, de escritura de caracteres o de modificación de la dirección (x,y) del puntero de escritura. Se enumeran en la tabla siguiente:

Tabla 5-2. Funciones contenidas en el código fuente *nokia5110.c*

Prototipo función	Finalidad
void **init_hw** (void)	Configuración puertos de E/S asociados a la LCD Configuración del módulo USCI_B en modo SPI
void **initLCD** ()	Inicialización del periférico mediante comandos propios
void **writeToLCD** (unsigned char dataCommand, unsigned char data)	Envío de datos/commandos a través del búfer TX del puerto SPI.
void **writeCharToLCD** (char c)	Ordenar el envío de un carácter ASCII único, ejecutando 5 veces la función *writeToLCD (píxel i)*. Los 5 píxeles que conforman cada carácter ASCII se leen de la tabla *font[]*.
void **writeStringToLCD** (const char *string)	Ordenar envío de caracteres de una cadena ejecutando sucesivamente *writeCharToLCD(carácter i)*
void **clearLCD** ()	Borrar todos los caracteres de la pantalla ejecutando sucesivamente *writeToLCD (espacios)* Sitúar puntero de escritura en la posición (0,0) -> *setAddr (0, 0)*
void **clearBank** (unsigned char bank)	Borrar una de las 6 secciones de la pantalla (“bancos”).
void **setAddr** (unsigned char xAddr, unsigned char yAddr)	Establecer posición del puntero de escritura en la pantalla. Se señala el píxel exacto de los 84x48 posibles.
void **setChAddr** (unsigned char xAddr, unsigned char yAddr)	Establecer posición del puntero de escritura a partir de la posición del carácter que se pretende escribir. Se especifica la línea y columna de las 6x14 posibles: *setChAddr (columna, fila)*

[35] Alguno de estos timbres pueden recordarnos a los de un órgano, un clarinete, e incluso una voz humana, pero no todos ellos tienen semejanza con instrumentos reales, ya que el objetivo perseguido no es lograr un parecido con los mismos.
[36] Consúltese el Anexo II, en el que se ilustran las formas de onda.

5.3.5 Main.c

Es el programa principal. Como hemos comentado con anterioridad, implementa los cuatro efectos del sintetizador: *tremolo, vibrato, FM y ADSR.* Se trata de un **programa más completo** que la versión anterior pues **condensa los ficheros main_tremolo.c, main_vibrato.c, main_FM.c y main_ADSR.c** del Midisynth 1.0, que antes se debían compilar y cargar por separado en el microcontrolador. Además, contiene las rutinas de interrupción programadas del DCO y LFO, para la generación de la onda de salida.

5.4 Modificaciones relevantes

En primer lugar, realizamos una "traducción" del código original, Midisynth 1.0, compilado para el MSP430G2553, en un nuevo programa compatible con el MSP430FR2355. Trabajamos en paralelo con ambos datasheet [10] [16] para comparar los registros propios de cada MCU e ir modificándolos uno por uno. **Naturalmente, esto no bastó para lograr en el nuevo microcontrolador el mismo funcionamiento del programa de síntesis primigenio; pese a las similitudes entre los dos dispositivos, no todos los procedimientos de inicialización y configuración de los periféricos coincidían**. Todo esto supuso un posterior estudio en profundidad de los manuales de usuario de sendos MCUs [17] [18] y de sus tarjetas de desarrollo (*Launchpad*s) [11] [19].

No ahondaremos en las modificaciones de los registros, puesto que fue necesario cambiar aproximadamente 70 de ellos. Sí que comentamos en esta sección algunos puntos importantes en la programación, como la configuración de los puertos de entrada/salida y el funcionamiento del convertidor DAC para la generación de la onda de salida. Asimismo, la función de configuración del reloj fue fundamental para poner en marcha el sintetizador. Además comentaremos otros detalles que, aunque puedan parecer nimios, supusieron importantes baches en el proyecto.

5.4.1 Función de configuración de los pines I/O

Comenzamos reprogramando la función `void conf_puertos (void).` Los pines de entrada/salida deberán configurarse convenientemente, según la función de los periféricos a los que se conecten. En nuestro MCU disponemos de 7 puertos, con 7 pines cada uno. La nomenclatura de estos pines tomará la forma:

Px.y donde ***x*** es el número del puerto e ***y*** el bit que indica el pin

Por lo general, si queremos escribir un '0' ó '1' lógico en un registro, emplearemos máscaras de bits, del modo siguiente:

```
Px_nombre_registro &= ~BITy;   Escribe '0' lógico en registro asociado a Px.y
Px_nombre_registro |=  BITy;   Escribe '1' lógico en registro asociado a Px.y
```

En primer lugar se indicará al programa, para cada pin utilizado, si el periférico que conectaremos será de entrada ó de salida, mediante las instrucciones:

```
PxDIR &= ~BITy[37];    Si es una entrada (Configuración por defecto).
PxDIR |=  BITy;        Si se trata de una salida.
```

[37] Para valores de los registros por defecto las instrucciones pueden omitirse.

Para los pines de entrada, se establecerán además los valores lógicos por defecto, configurándolos bien a nivel alto '1', con resistencia de *pull-up*, ó nivel bajo '0', mediante resistencia de *pull-down*. Esto se programará como sigue:

```
PxREN |=  BITy;
PxREN &= ~BITy;
```

Resistencia de *pull-up*.
Resistencia de *pull-down*.

A continuación, se activará la función específica del pin, escribiendo en los registros P1SEL0 y P1SEL1*. Las 4 posibles combinaciones binarias de estos registros permitirán configurar distintas funciones, que se detallan en el *datasheet* del microcontrolador. En general, la combinación "00" (P1SEL0 = '0', P1SEL1 = '0') corresponderá a la función GPIO *("General Purpose In/Out"),* y será un valor por defecto para todos los pines[38]. Pongamos como ejemplo, la configuración de la línea de transmisión de datos, *UCB0SDA*, del puerto serie SPI. Consultando el *datasheet,* encontramos que dicha función se configura en el pin P1.2, dando valor "01" al registro PSEL:

```
P1SEL0 |=  BIT2;
(P1SEL1 &= ~BIT2;)
```

Bit menos significativo de "01"
Esta instrucción se obvia, porque ser '0' su valor por defecto.

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS[1]	
			P1DIR.x	P1SELx
P1.0/UCB0STE/SMCLK/ COMP0.0/A0/Veref+	0	P1.0 (I/O)	I: 0; O: 1	00
		UCB0STE	X	01
		SMCLK	1	10
		VSS	0	
		COMP0.0, A0/Veref+	X	11
P1.1/UCB0CLK/ACLK/ OA0O/COMP0.1/A1	1	P1.1 (I/O)	I: 0; O: 1	0
		UCB0CLK	X	01
		ACLK	1	10
		VSS	0	
		OA0O[2], COMP0.1, A1	X	11
P1.2/UCB0SIMO/ UCB0SDA/TB0TRG/ OA0-/A2/Veref-	2	P1.2 (I/O)	I: 0; O: 1	00
		UCB0SIMO/UCB0SDA	X	01
		TB0TRG	0	10
		OA0-[2], A2/Veref-	X	11
P1.3/UCB0SOMI/		P1.3 (I/O)	I: 0; O: 1	00

Tabla 5-3. Funciones asociadas a los pines E/S. Detalle [9].

Procedemos de igual modo para la configuración de todos los pines, puede consultarse la codificación de la función completa en el Anexo I.

Como último paso, pero no menos importante, será **necesario desactivar el modo de alta impedancia de los GPIOs para activar la configuración anterior de los puertos. De no ejecutarse esta instrucción, no se harán efectivas** todas las escrituras de los registros que hemos realizado:

```
PM5CTL0 &= ~LOCKLPM5;
```

Estudiando detenidamente los datasheet de ambos microncontroladores se observa que las funciones no se localizan en los mismos puertos, por lo que hemos tenido que modificar la mayoría de los pines E/S asociados a cada periférico, respecto a la versión Midisynth 1.0. Continuando con el ejemplo anterior, la línea de transmisión SPI no se asociaba al pin P1.2 en el MSP430Gxxxx, sino al P1.7.

[38] Ídem.

Especificamos en la tabla siguiente la nueva distribución de los pines. Los pines de GPIO se asignaron en último término, utilizando los pines que quedaban libres, una vez asociados los pines con funciones "fijas"; nótese que, en el ejemplo que comentábamos, la línea *UCB0SDA* sólo podrá ser configurada en el pin P1.2, con lo que le asignamos dicho pin al pin *Din* de la pantalla LCD, y buscamos un pin libre para el *Chip Select*, cuya función puede cumplir cualquier otro puerto E/S.

Tabla 5-4. Distribución de pines de E/S [10].

Periférico	Nombre pin	Función requerida	Puerto E/S	Configuración PSEL*	
				PSEL1	PSEL0
Entrada MIDI	MIDI-IN	*UCA0RXD:* línea de recepción de datos UART.	P1.6	0	1
Salida de Audio	Audio-OUT	*OA2O:* salida del convertidor DAC	P3.1	1	1
Pantalla	Clk	*UCB0CLK:* señal de sincronismo	P1.1	0	1
	Data in	*UCB0SDA:* línea de transmisión de datos SPI.	P1.2	0	1
	Data/Control	*GPIO*	P1.5	0	0
	Chip Select	*GPIO*	P1.7	0	0
Joystick	UD	*A3:* canal 3 del convertidor ADC	P1.3	1	1
	LR	*A4:* canal 4 del convertidor ADC	P1.4	1	1
Pulsadores	Botón 1	*GPIO*	P6.3	0	0
	Botón 2	*GPIO*	P6.2	0	0
	Botón 3	*GPIO*	P6.1	0	0
	Botón 4	*GPIO*	P6.0	0	0
Encoder	Canal A	*GPIO* (tren de pulsos)	P2.4	0	0
	Canal B	*GPIO* (tren de pulsos)	P2.5	0	0
	Botón 5	*GPIO*	P3.3	0	0

* En el MSP430G2553, registros P1SEL y P1SEL2.

5.4.2 Función de configuración del reloj

En el microcontrolador MSP430G2553, la función `void conf_reloj (char VEL)` se limitaba a configurar los registros de control propios del reloj, DCOCTL y BCSCTLx, de forma sencilla, utilizando ciertas máscaras de bits, del tipo CALBC1_xMHZ y CALDCO_xMHZ. No era necesaria la calibración del reloj sino simplemente escribir en los registros con dichas máscaras. Por ejemplo, para calibrar la frecuencia a 16MHz, como deseamos en nuestro programa, la función ejecuta las instrucciones:

```
BCSCTL1 = CALBC1_16MHZ;
DCOCTL = CALDCO_16MHZ;
```

El microcontrolador MSP430FR2355, en cambio, precisa un procedimiento más complejo para ajustar el oscilador *DCO*[39]. Por defecto, el dispositivo ejecuta un sistema de estabilización de la frecuencia denominado Frequency Locked Loop, FLL, mediante el cual se logra que la frecuencia tome valores en un rango próximo a la frecuencia *DCO* seleccionada. *Asegurarse de que el bucle FLL se puede bloquear para el rango de frecuencia seleccionado por su aplicación requiere el uso de una calibración por software y una rutina de bloqueo [18].*

[39] *DCO: "Digitally Controlled Oscillator".*

La calibración por software se implementa mediante la función `void Software_Trim (void)` en la que se ajusta el valor de los registros CSCTL0 y CSCTL1. El procedimiento paso a paso de ejecución se ilustra en el manual de usuario *[18].* Consultando los códigos de ejemplo disponibles en la página del fabricante [20], modificamos la función `void conf_reloj (char VEL)`, adaptándola al nuevo microcontrolador.

5.4.3 Convertidor Digital-Analógico vs. Modulación PWM

Como venimos diciendo, una de las mejoras fundamentales que se introducen en el sintetizador es la generación de la onda de salida mediante el convertidor DAC. Mediante la rutina de interrupción DCO (no confundir con el reloj del microcontrolador, *DCO, "Digitally Controlled Oscillator"*) generamos punto a punto las ondas sonoras. Comparamos brevemente ambos métodos de generación de la onda, para mejor comprensión de su programación.

Recordemos el funcionamiento de la modulación por anchura de pulsos, *PWM.* El porcentaje de tiempo a nivel lógico alto/bajo sobre el periodo se determina mediante el parámetro denominado Duty Cycle:

$$\textbf{Duty Cycle} = \frac{Ton}{T} = \frac{Ton}{Ton+Toff}$$

En nuestro caso, el nivel lógico alto corresponderá al valor de tensión con el que trabajamos, A = 3.3V.

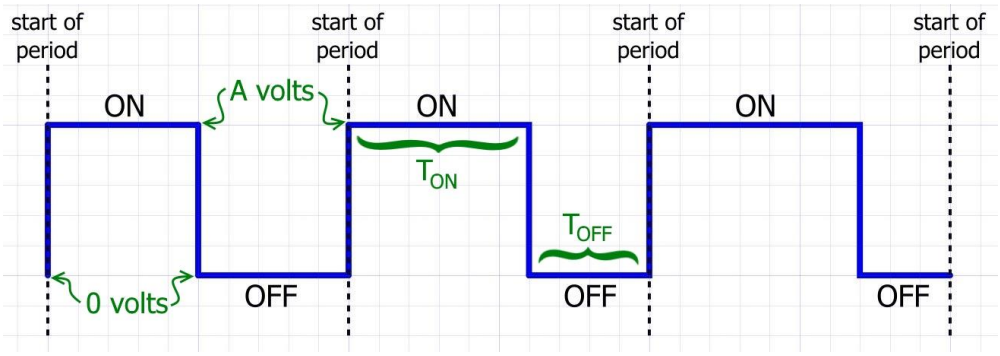


Figura 5-3. Modulación por anchura de pulsos, *PWM* [40].

Un convertidor digital-analógico, asociará los valores de la onda digitalizada, con cierta resolución, a distintos valores de tensión. El *DAC* con el que trabajamos hará corresponder las distintas alturas de la onda, en el rango [0, 4095][41] a valores de tensión en el intervalo [0, 3.3] V. En la modulación PWM los puntos de tensión se lograban de forma similar, simulando un convertidor *DAC,* de modo que el voltaje que éste produciría se calculaba así:

$$\textbf{Voltaje DAC} = \textbf{A} * \textbf{Duty Cycle} = 3.3V * \textbf{Duty Cycle}$$

Por ejemplo, para un duty cycle del 70%, obtendríamos un voltaje de salida de 0.7x3.3 V = 3.31 V.

[40] Imagen tomada de la fuente: https://www.allaboutcircuits.com/technical-articles/turn-your-pwm-into-a-dac/ Fecha de consulta: Septiembre de 2019.

[41] La resolución de nuestro convertidor DAC es de 12 bits: $2^{12} = 4095$.

La señal modulada era transmitida a un *filtro paso-bajo* para eliminar la portadora y reconstruir la onda analógica. Pero, ¿cuál sería la resolución del convertidor DAC emulado? Entendemos por resolución el número de diferentes voltajes que el *DAC* es capaz de generar, por tanto la resolución utilizando *PWM* dependerá del número de valores distintos que pueda tomar el parámetro Duty Cycle. Si el periodo que hemos elegido es T = 800, entonces T^{ON} podrá valer entre 0 y 799, y el Duty Cycle podrá tomar hasta 800 valores distintos. Esto equivaldría a tener un convertidor entre 8 y 9 bits, aproximadamente: $2^8 = 512$, $2^9 = 1024$.

El procedimiento seguido en el programa Midisynth 1.0 era escribir en el registro del trigger del *PWM*, al principio de la rutina de interrupción, la altura de la onda de salida, calculada en la interrupción anterior, resetear la variable en cuestión ("*dac*") y recalcularla para ser utilizada en el siguiente paso:

```
#pragma vector=TIMER0_A0_VECTOR
__interrupt void VCO (void)
{
    TA0CCR1 = dac;
    dac = 0;
    // cálculo del nuevo dac
    // (…)
```

La programación del DAC en el nuevo microcontrolador es similar pero más intuitiva: simplemente se escribe en el registro del convertidor el valor de altura calculado, en el mismo paso. Por tanto procedemos de igual modo, a cada interrupción se resetea la variable en cuestión, se recalcula y se asocia al convertidor de salida:

```
#pragma vector = SAC0_SAC2_VECTOR
__interrupt void SAC2_ISR(void)
{
  switch(__even_in_range(SAC2IV,SACIV_4))
  {
    case SACIV_0: break;
    case SACIV_2: break;
    case SACIV_4:

      DAC_data12 = 0;
      // cálculo del nuevo dac
      // (…)
      SAC2DAT = DAC_data12;
```

5.4.4 Otros detalles

Además de los cambios comentados, fueron necesarios otros pequeños ajustes del código para superar errores que fueron apareciendo debidos a las diferencias de funcionamiento de sendos microcontroladores. Entre otros, cabe destacar el aumento de los tiempos de espera tras la configuración inicial de la pantalla, periférico que en un principio no lográbamos que operara correctamente. Además se cambió la programación del joystick, puesto que el convertidor ADC ahora contaba con 12 bits de resolución frente a los 8 bits anteriores. De igual manera, tuvimos que escalar la onda de salida antes de transmitir sus puntos al convertidor *DAC*, de 12 bits de resolución, puesto que la onda generada anteriormente mediante *PWM* contaba con 800 puntos de resolución (9~10 bits).

6 FABRICACIÓN DE LA TARJETA MIDISYNTH 2.0

All the electronic devices are powered by white smoke.
When smoke goes out, device is dead.

- Milan Nikolice -

Como comentábamos al principio del texto, en este proyecto elegimos un planteamiento conservador: partiendo de la versión anterior del sintetizador, Midisynth 1.0, hemos ido introduciendo las mejoras de forma progresiva. Las **pruebas realizadas durante la fase de programación** del nuevo microcontrolador, MS430FR2355, se llevaron a cabo **sobre la placa primitiva, Midisynth 1.0.** Dicha placa de desarrollo fue diseñada para conectar el *Launchpad-MSP430G2553,* con dos headers de 10 pines de conexión directa a los 20 pines de dicho dispositivo. Por ello, para la implementación inicial del sintetizador, debimos adaptar la conexión de la primera versión de tarjeta al nuevo MCU, mediante conectores macho/hembra.

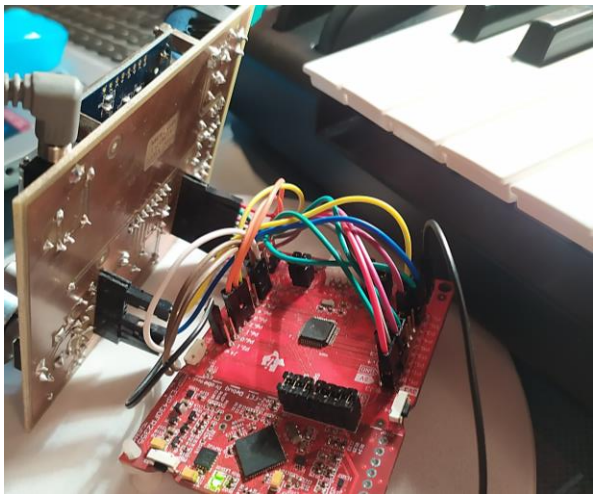


Figura 6-1. Conexión de la placa Midisynth 1.0 adaptada al nuevo MCU.

Una vez programado el sintetizador en el nuevo microcontrolador, **culminamos el proyecto fabricando la placa PCB adaptada** al mismo: la **tarjeta Midisynth 2.0**. En ella no solo variará la geometría de los headers de conexión (en el *Launchpad-MSP430FR2355* encontraremos 40 pines repartidos en 4 hileras), sino la distribución de los componentes electrónicos: debemos adaptarnos a la localización de los puertos en el nuevo MCU de forma óptima, minimizando las longitudes de pista para minimizar el efecto parásito resistivo. Otras novedades serán la introducción del potenciómetro logarítmico (control de volumen) y eliminación del filtro paso-bajo, presente en la versión anterior para suavizar la salida analógica PWM.

En la siguiente figura se comparan las geometrías de ambas versiones de la tarjeta Midisynth, en su cara

inferior. Nótese la variación en los conectores del Launchpad.

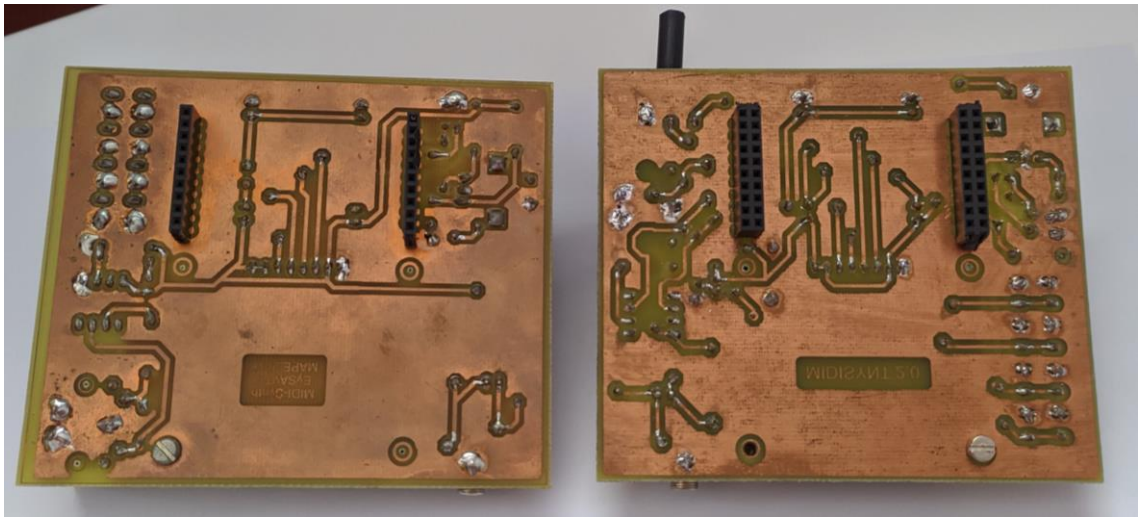


Figura 6-2. Tarjeta Midisynth 1.0 vs Midisynth 2.0

6.1 Diseño de la PCB

En primer lugar elaboramos el fotolito mediante el editor de circuitos impresos *Eagle 9.5*.

6.1.1 Edición del diagrama esquemático

Comenzamos creando un nuevo componente en la librería personalizada de *Eagle* para el microcontrolador *MSP430FR2355*. Se define su huella, atendiendo a la geometría del dispositivo, y se asocian los 20 pines a los puertos E/S correspondientes. De igual modo procedimos con el potenciómetro, que tampoco se encontraba en las librerías preexistentes.

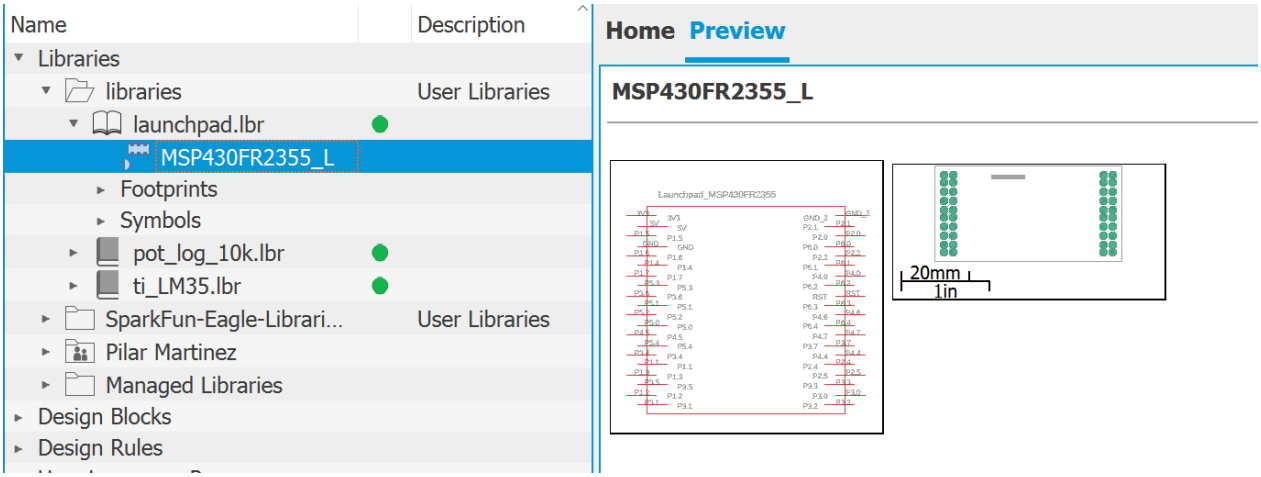


Figura 6-3. Edición de los nuevos componentes en las librerías Eagle.

Conectamos los periféricos a los puertos del MCU que habíamos asociado en el programa, obteniendo el diagrama que se muestra en la figura 6-3. Se identifican en la misma los distintos componentes grosso modo; para mayor detalle consúltese el Anexo III. Obsérvese que el potenciómetro se localiza entre el pin de salida

del DAC y la entrada del amplificador.

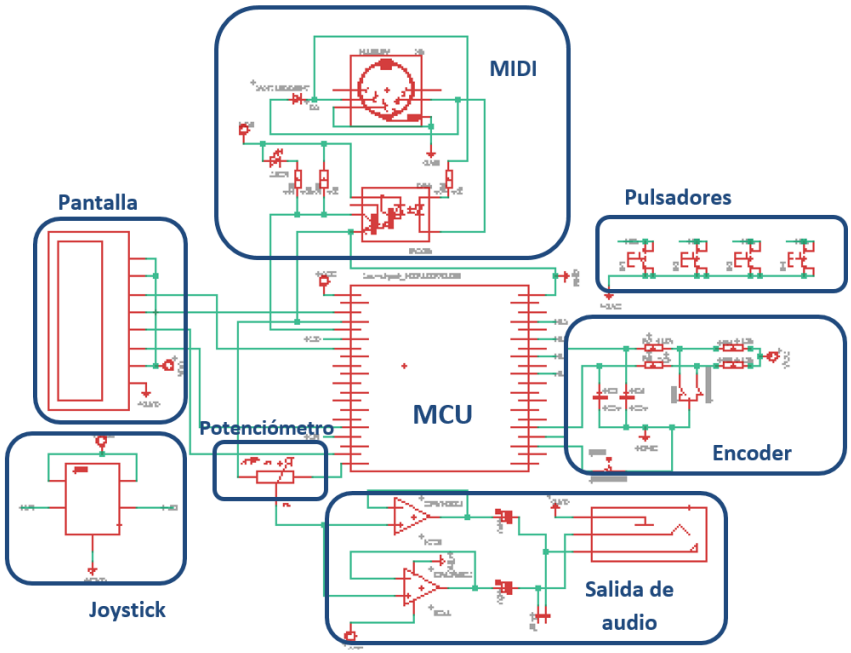


Figura 6-4. Diagrama esquemático del circuito.

6.1.2 Rutado

A partir del esquemático anterior, con el mismo programa, editamos la placa PCB. Se trató de distribuir los componentes de forma óptima y realizar el rutado de las pistas, por ambas caras de la placa. Entre otros detalles, debimos tener en cuenta los siguientes puntos:

- Las pistas debían alcanzar los pines en la cara de la PCB por la que serían soldados posteriormente. Por ejemplo, en el caso de los conectores del Launchpad, situados en la cara *bottom* y soldados en la cara *top,* las pistas tenían que llegar por la misma cara superior. Para el resto de componentes, situados encima de la placa, las pistas debían alcanzar sus pines de conexión en la cara inferior.
- Como excepción, los pines de conexión de las resistencias y los pulsadores pudieron ser rutados tanto por encima como por debajo de la placa, por ser más sencillos de soldar. Esto fue aprovechado para evitar el uso excesivo de vías de conexión entre ambas caras.
- Debimos tener cuidado con el paso de las pistas entre pines, para evitar posibles cortocircuitos. En especial, evitamos estos cruces en el conector del Launchpad, cuyos *pads* se hallaban muy próximos entre sí.

El fotolito del circuito impreso, resultado final del proceso de rutado, se adjunta también en el Anexo III.

6.2 Montaje de la placa

Finalizado el diseño de la placa, y revelado el fotolito, solo restaba el proceso de taladro y soldadura de los componentes en la PCB. Aunque esto se presentaba como una tarea sencilla, encontramos no pocos obstáculos en la misma, que comentamos de forma concisa para facilitar posibles mejoras futuras del sintetizador.

Aunque advertimos la necesidad de eludir el paso de pistas cerca de los pads de conexión del Launchpad, los numerosos cruces entre pistas nos llevaron a adoptar esta solución en ciertos puntos de la placa. La dificultad encontrada posteriormente al soldar puntos tan delicados nos lleva a concluir que lo óptimo hubiera sido

incluir mayor número de vías para evitar dicha situación. En la figura se observan dichos cruces de las pistas en detalle:

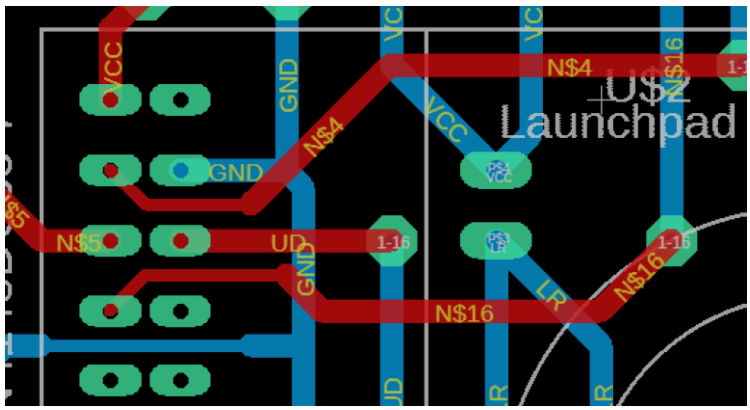


Figura 6-5. Cruces delicados de pistas entre los PADs.

Por otro lado, la soldadura de determinados componentes con conexiones muy finas fue también dificultosa. En el caso del amplificador de audio, por ejemplo, se produjeron cortocircuitos y fue necesario repetir y comprobar las soldaduras. La complicación aumentaba debido a la proximidad del resto de componentes, que podría haberse evitado aumentando el tamaño total de la tarjeta como solución de compromiso.

Por último, debemos aclarar para futura fabricación de la tarjeta, que erramos en la edición de la huella del Launchpad, resultando la separación entre sendos headers menor de la que debiera. Concretamente, esto debería ser corregido aumentando en 2.54mm la distancia entre los headers interiores. Nosotros adaptamos el enlace del MCU a la PCB mediante headers macho/hembra intermedios, que hicieron posible lograr un correcto funcionamiento de la tarjeta, si bien es cierto que se trataba de una conexión más inestable.

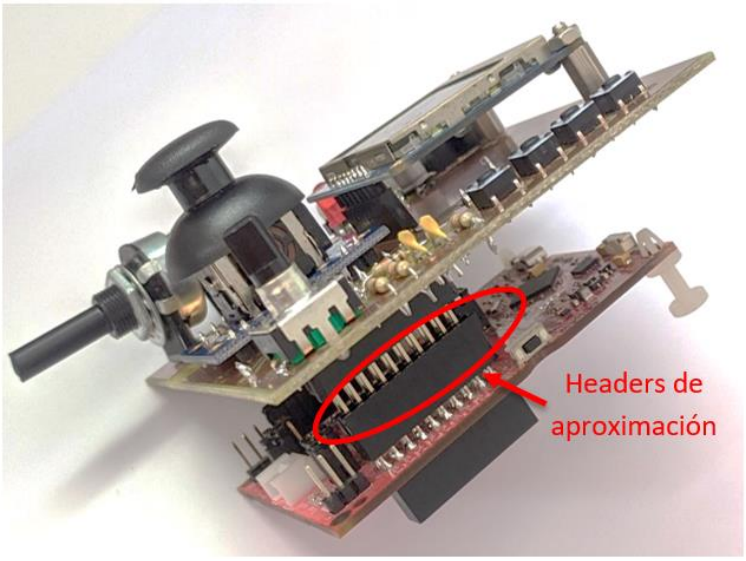


Figura 6-6. Error de conexión solventado.

7 RESULTADOS

Faith is like electricity.
You can't see it, but you can see the light.

- Gregory Dickow -

Los frutos más vistosos del proyecto, por su naturaleza sonora, no podrán ser presentados en este documento. No obstante, aunque no se demuestre aquí la amplia variedad de sonidos generados por el sintetizador implementado, representaremos gráficamente algunas de las ondas sonoras producidas, medidas mediante un osciloscopio en el conector de salida de la tarjeta Midisynth 2.0.

Estas gráficas nos servirán para ilustrar los distintos efectos de audio comentados en el capítulo teórico, confirmando que el dispositivo construido es capaz de crear las formas de onda que referíamos. Asimismo, comparamos las gráficas generadas mediante el Midisynth 2.0 con aquellas que resultaban de la modulación *PWM* en su versión anterior.

Comenzamos mostrando los 4 efectos de audio obtenidos. El primero de ellos, el trémolo, resulta de utilizar la onda senoidal (instrumento 0) como señal portadora, y una señal de naturaleza cuadrada (instrumento 1) como moduladora:

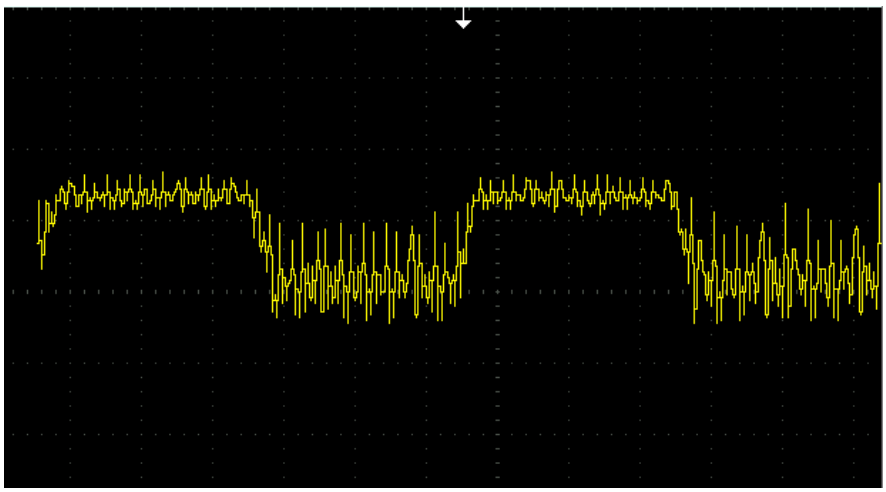


Figura 7-1. Efecto trémolo obtenido mediante Midisynth 2.0.

El siguiente efecto buscado era el vibrato. Para visualizar dicho efecto de forma más clara, se ha tomado la señal senoidal como portadora y también moduladora. De igual modo se ha procedido para obtener la gráfica de la modulación en frecuencia, FM. Nótese que, como comentamos en el capítulo 2, la modulación FM no es más que un vibrato muy rápido. De ahí que aunque ambas gráficas hayan sido tomadas en la misma escala temporal, en la primera de ellas (vibrato) no llega a distinguirse un periodo completo, cosa que sí logramos capturar para la modulación FM.

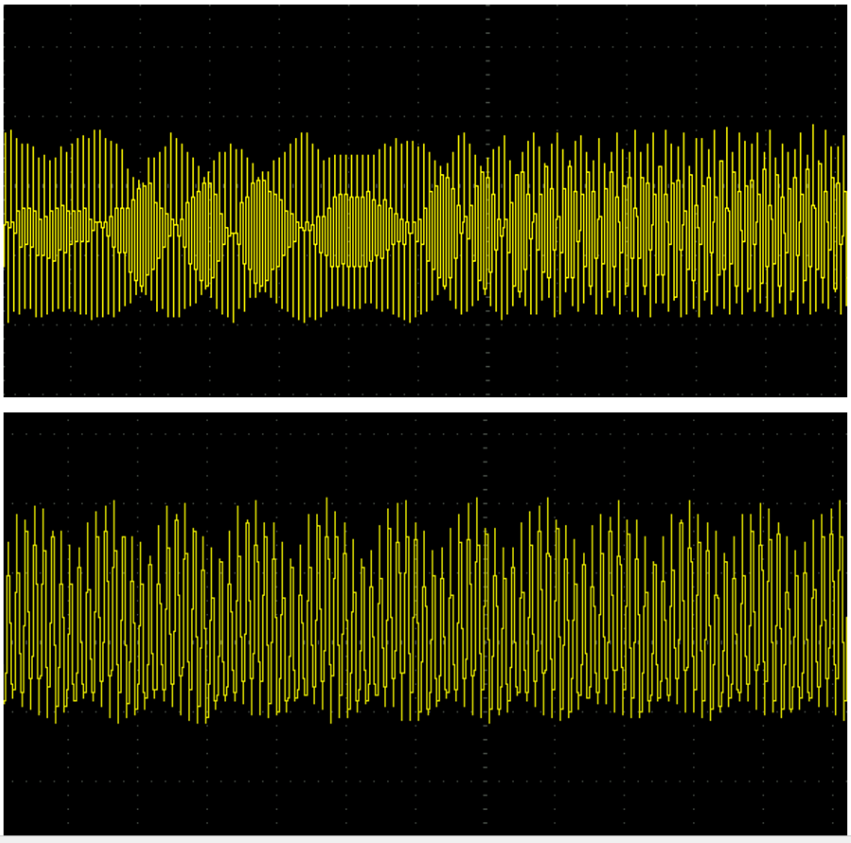


Figura 7-2. Efectos de modulación en frecuencia: vibrato y FM.

Finalmente, tratamos de graficar la envolvente de la modulación ADSR, utilizando de nuevo la senoidal como portadora. Sin embargo, se hace difícil la captura de dicha onda por la rapidez con la que ocurren las fases de ataque, decaimiento y relajación (del orden de milisegundos). Con suerte, hemos logrado congelar la imagen del osciloscopio al inicio de la onda:

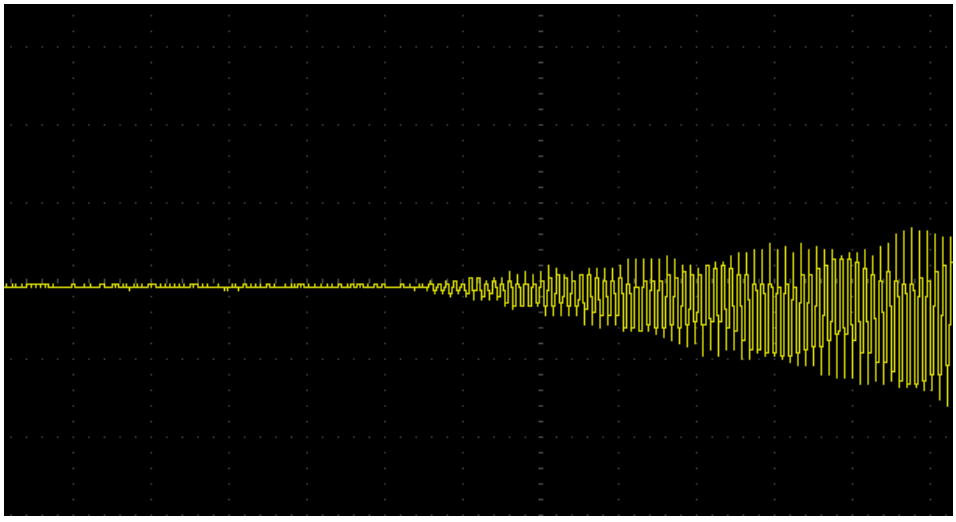


Figura 7-3. Inicio de la envolvente ADSR.

Igualmente, resultan llamativas las mismas ondas sonoras, sin aplicarles efecto de modulación alguno, al

compararlas con las que se obtenían con el sintetizador Midisynth 1.0. La gráfica superior corresponde a la onda senoidal generada mediante *PWM*, y la inferior, a la producida por el convertidor *DAC* de la nueva versión Midisynth. Nótese que el trazo de la onda se hace mucho más fino.

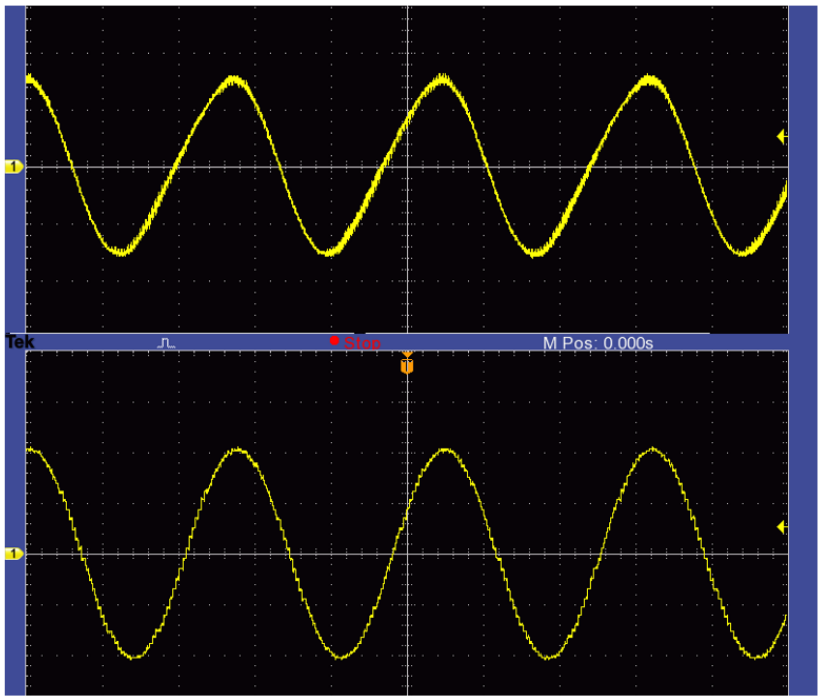


Figura 7-4. Comparativa onda senoidal: PWM vs DAC

En la siguiente figura puede observarse el rizado en detalle:

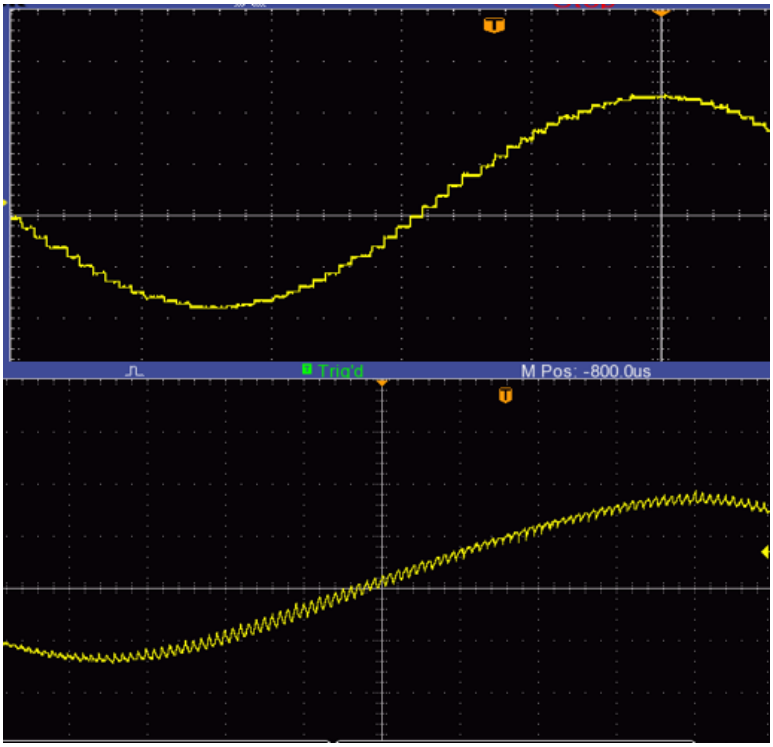


Figura 7-5. Detalle rizado PWM vs DAC

Se adjuntan asimismo sendas gráficas comparando la onda número 2 de la tabla, en ambos

sintetizadores, con su rizado correspondiente.

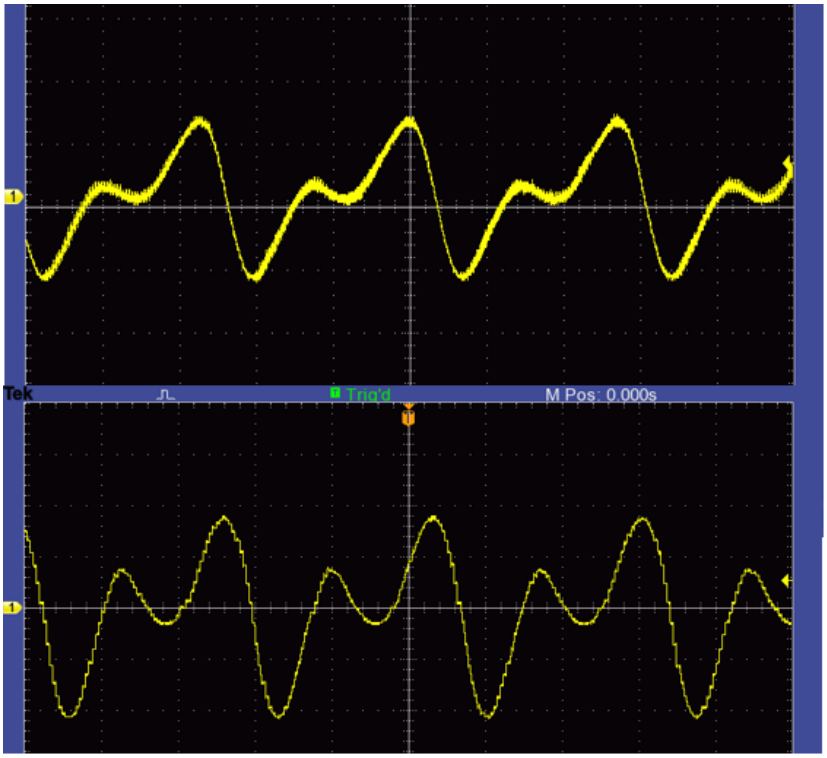


Figura 7-6. Onda nº 2: *PWM vs DAC*

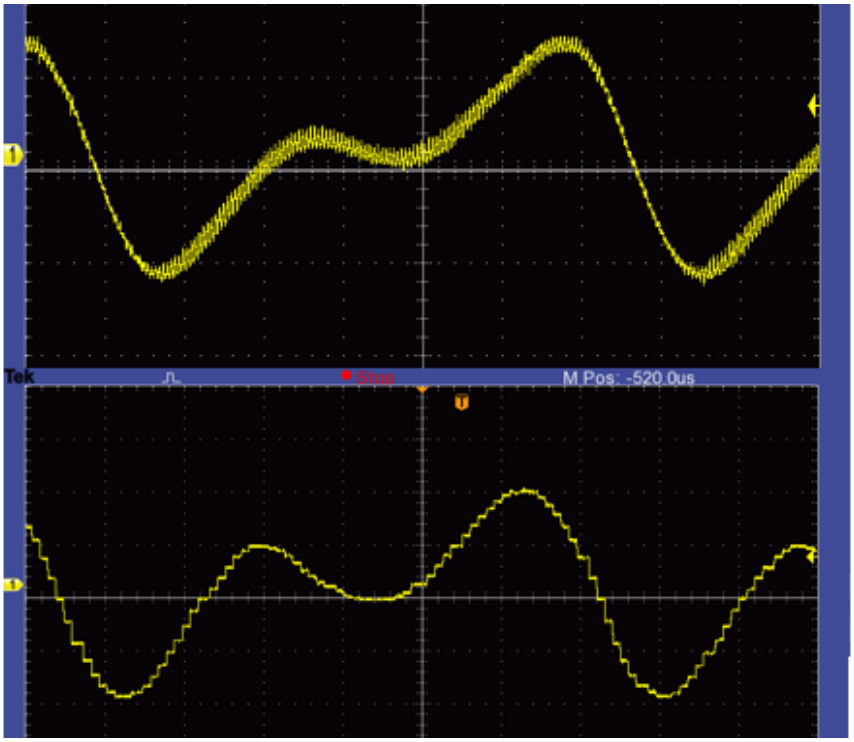


Figura 7-7. Detalle del rizado para la onda nº 2.

En definitiva, resultados más tangibles pueden observarse (ó más bien, escucharse) al ejecutar el programa

microcontrolador conectado a la tarjeta Midisynth que hemos fabricado. Para ello se adjunta el código completo en el Anexo I.

Nos gustaría aclarar que aunque dicho código sigue el protocolo MIDI estándar, se incluyen ciertas líneas de instrucciones, debidamente comentadas, para su uso en pianos electrónicos del modelo YAMAHA Clavinova clp-320. Esto se debe a que en un principio nosotros trabajamos con dicho modelo de teclado, con supuesta comunicación MIDI según el fabricante [21]. Sin embargo encontramos numerosos inconvenientes en la programación, ya que el dispositivo no seguía el estándar, sino que empleaba un procedimiento propio (esto sucede con ciertos instrumentos electrónicos) por el cual solo mandaba mensajes de note-ON, que se distinguían de los de note-OFF por tener estos últimos velocidad nula. Además cada 300 ms el teclado enviaba mensajes de sistema propios, del tipo “active sensing” con formato binario “1111 1110”.

8 CONCLUSIONES

The simpler the insight,
the more profound the conclusion

- Janna Levin -

A la vista de los satisfactorios resultados que acabamos de presentar, podría justificarse la implementación de una nueva versión de la tarjeta en el futuro. **Bajamos el telón de este proyecto proponiendo una serie de líneas de mejora** para nuestro sintetizador digital Midisynth 2.0.

En primer lugar, la disponibilidad de memoria posibilitaría la **ampliación de la tabla de ondas**, tanto incrementando el número de las mismas como mejorando su resolución aumentando la frecuencia de muestreo, es decir, almacenando más de 64 muestras por cada una de las ondas.

Cabe decir que en los **sintetizadores comerciales e instrumentos electrónicos, no solo se almacena una onda por cada tipo de timbre, sino al menos una onda por cada cierto conjunto de notas** e incluso una onda por cada nota. Esto es debido a la variación de los armónicos para un mismo instrumento entre sonidos de distinta frecuencia. De ahí que en el repositorio de ondas que utilizamos encontráramos gran cantidad de muestras por cada instrumento, entre las cuales elegimos una sola onda para implementar cada uno. Se propone aquí como trabajo futuro posibilitar al programa la selección de distintas ondas en función de la nota a producir.

Por otro lado, si se buscara un **sintetizador de tamaño más reducido ó la capacidad de controlarlo únicamente vía MIDI**, podría prescindirse de alguno de los periféricos de control (joystick, potenciómetro, pulsadores), **ampliando el conjunto de mensajes MIDI comprensibles por el programa**. Además de los mensajes note ON/OFF que atiende actualmente, sería interesante programar el pitch bend y alguno de los mensajes de control del protocolo.

Adicionalmente, podría contemplarse la **idea de sustituir el amplificador de audio que se integra en la placa, por el amplificador interno del que dispone el MSP430FR2355**, de ganancia programable. Se deberían estudiar las corrientes máximas para verificar si sería adecuado para la operación del sintetizador.

Finalmente, debe tenerse en cuenta para la fabricación futura de la tarjeta PCB los **puntos de mejora en el circuito impreso que propusimos en el capítulo 6**. Recuérdese en primer término solventar el error en la separación de los headers de conexión del Launchpad, modificando la huella del mismo como se especificaba en dicho apartado. Además se recomienda **rediseñar el rutado del circuito impreso, aumentando la superficie de la placa como solución de compromiso, para evitar la proximidad entre las pistas y los pads**. De este modo se facilitará el proceso de soldadura y se evitarán cortocircuitos inesperados.

Terminamos remarcando que este trabajo no solo nos ha servido como oportunidad para aplicar los conocimientos sobre Electrónica adquiridos durante el Grado y el aprendizaje de muchos otros, sino que ha supuesto un acercamiento atrayente al mundo de la síntesis digital. **Aunque las propuestas de trabajo que señalábamos previamente ya escapan del alcance de nuestro proyecto, no descartamos ahondar algún día en este nuevo arte:** la idea de cómo unas simples ondas, "retocadas" debidamente, pueden llegar a conformar ricas melodías realmente ha logrado despertar nuestro interés y esperamos que también el del lector.

REFERENCIAS

[1] Real Academia Española, «RAE,» 2019. [En línea]. Available: https://dle.rae.es/. [Último acceso: Septiembre 2019].

[2] WIKIPEDIA, «Transductor,» Octubre 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Transductor. [Último acceso: Diciembre 2019].

[3] D. Martínez-Zorrilla, «ResearchGate,» 2008. [En línea]. Available: https://www.researchgate.net/publication/260403799_LOS_SINTETIZADORES_Una_breve_introduccion. [Último acceso: Julio 2019].

[4] Simon Fraser University, «Frequency Modulation,» [En línea]. Available: https://www.sfu.ca/sonic-studio-webdav/handbook/Frequency_Modulation.html. [Último acceso: Diciembre 2019].

[5] DISCA, Departamento de Informática de Sistemas y Computadores, «Tema 5, MIDI,» Universitat Politècnica de València, UPV, [En línea]. Available: http://www.disca.upv.es/adomenec/IASPA/tema5/Midi.html. [Último acceso: Agosto 2019].

[6] Soyuz, «El protocolo MIDI,» HISPASONIC, Enero 2002. [En línea]. Available: https://www.hispasonic.com/reportajes/protocolo-midi/13#section2. [Último acceso: Junio 2019].

[7] L. C. C., «U-Cursos. MIDI: tipos de mensajes y circuitos.,» [En línea]. Available: https://www.u-cursos.cl/artes/2011/1/ESON361-204/1/material_docente/bajar?id_material=584523. [Último acceso: Mayo 2019].

[8] S. J. Puig, «La especificación MIDI a fondo,» Guías Monográficas Anaya Multimedia, Madrid, 1997.

[9] WIKIPEDIA, «MATLAB,» [En línea]. Available: https://es.m.wikipedia.org/wiki/MATLAB. [Último acceso: Agosto 2019].

[10] Texas Instruments, «MSP430FR235x, MSP430FR215x mixed-signal microcontrollers datasheet (Rev. C),» Marzo 2019. [En línea]. Available: http://www.ti.com/lit/ds/symlink/msp430fr2355.pdf. [Último acceso: 20 Octubre 2019].

[11] Texas Instruments, «MSP430FR2355 LaunchPad™ Development Kit (MSP-EXP430FR2355) User's Guide,» Mayo 2018. [En línea]. Available: http://www.ti.com/lit/ug/slau680/slau680.pdf. [Último acceso: 19 Noviembre 2019].

[12] M. Á. P. Esteve, «Apuntes Práctica 3: Equipos y Sistemas de Audio, Vídeo y TV,» Universidad de Sevilla, Sevilla, 2018.

[13] L. Díaz, «Descubre el potenciómetro,» DIWO Mundo Reader, S.L, 27 Noviembre 2015. [En línea]. Available: http://diwo.bq.com/descubre-el-potenciometro/. [Último acceso: Noviembre 2019].

[14] Integrated Circuits, «PCD8544, 48x84 pixels matrix LCD datasheet,» Abril 1999. [En línea]. Available: https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf. [Último acceso: Septiembre 2019].

[15] L. Llamas, «Medir el ángulo y el sentido de giro con Arduino y encoder rotativo,» Luis Llamas, Noviembre 2016. [En línea]. Available: https://www.luisllamas.es/arduino-encoder-rotativo/. [Último acceso: Agosto 2019].

[16] Texas Instruments, «MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller datasheet (Rev. J),» Mayo 2013. [En línea]. Available: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf. [Último acceso: 18 Octubre 2019].

[17] Texas Instruments, «MSP430x5xx and MSP430x6xx Family User's Guide,» Marzo 2018. [En línea]. Available: http://www.ti.com/lit/ug/slau208q/slau208q.pdf. [Último acceso: Febrero 2019].

[18] Texas Instruments, «MSP430FR4xx and MSP430FR2xx family User's Guide,» Marzo 2019. [En línea]. Available: http://www.ti.com/lit/ug/slau445i/slau445i.pdf. [Último acceso: Mayo 2019].

[19] Texas Instruments, «MSP-EXP430G2 LaunchPad™ Development Kit,» Marzo 2016. [En línea]. Available: http://www.ti.com/lit/ug/slau318g/slau318g.pdf. [Último acceso: Marzo 2019].

[20] Texas Instruments, «MSP430FR235x, MSP430FR215x Code Examples (Rev. B),» 2018.

[21] YAMAHA, «CLP-320 Data List,» 2008. [En línea]. Available: https://jp.yamaha.com/files/download/other_assets/9/335659/clp320_en_dl_a0.pdf. [Último acceso: Octubre 2019].

[22] Adventure Kid, «Single cycle waveforms, AKWF free samples,» [En línea]. Available: https://www.adventurekid.se/akrt/waveforms/adventure-kid-waveforms/individual-folder-downloads-of-the-akwf-pack/. [Último acceso: 2019].

[23] M. Á. P. Esteve, «Midisynth 1.0, ficheros fuente,» Sevilla, 2018.

[24] N. Petkov, Artist, *Electronic analog Moog Synthesizer.* [Art]. Creative Commons - Attribution ShareAlike International, 2017.

[25] R. Keim, «Turn Your PWM into a DAC,» Abril 2016. [En línea]. Available: https://www.allaboutcircuits.com/technical-articles/turn-your-pwm-into-a-dac/. [Último acceso: Septiembre 2019].

GLOSARIO

ADC: *Analog to Digital Converter* 12

DAC: *Digital to Analog Converter* 12

DCO: *Digitally Controlled Oscilator* 30

FM: *Frequency Modulation* 9

GPIO: *General-Purpose Input/Output* 17

LFO: *Low frequency oscillation* 7

MCU: *MicroController Unit* 12

PCB: *Printed Circuit Board* 1

PWM: *Pulse-Width Modulation* xi

TI: *Texas Intruments* 13

ANEXO I

Se adjuntan en este apartado los **ficheros fuente del programa completo**. En este orden: main.c, midisynth.h, Midisynth.c, Nokia5110.c y formas2.h.

❖ main.c

```
*****************************************/
/* Sintetizador aditivo N formas            */
/*****************************************/

#include <msp430.h>
#include "midisynth.h"
#include "stdio.h"
#include "formas2.h"

#define  HzM  *64L*256/100              // sizeof(sin)*256 / F(tmr2)

// VARIABLES:
unsigned int fase = 0, frec = 0;        // Fase (w·t)
char Env_on = 0;                        // Flag modulación activa
int Env = 0;                            // Valor modulación
int InstMod = 0;                        // Inst. Referencia modulación

int frecMod_ini_tremolo = (int) (1 HzM);
int frecMod_ini_vibrato = (int) (5 HzM);
int faseMod = 0, frecMod = 0;
int Tremolo_ON = 1, Vibrato_ON = 0, FM_ON = 0, ADSR_ON = 0, modulacion = 0;

volatile char nota = 0 ;            // Indica nota ON/OFF
char nota_act = 0, Fuerza = 0;
char inst[NUMINST];                 // Vector de 0/1: indica instrumento activos
char instrum = 1;                   // Instrumento seleccionado [1,8]
char Inst_act;                      // No. de instrumentos activos
char fase_ins[NUMINST];             // Vector fases relativas de instrumentos

int desplaz = 0;                    // Desplazamiento del joystick [0, 64]
char enc[16];                       // Búffer para escritura sprintf

char calcula_onda = 0;              // Flag para recalcular onda
int forma[64];                      // Vector alturas:64 puntos de onda calculada

int t_nota, t_nota_off, tA, tD, tR, SUST, Amp_int;   // Parámetros ADSR
float Amp, Amp_i, Amp_d, Amp_r;
char indice_param = 0;              // Parámetro actual (ENCODER)

int I_MOD = 0, FM = 0;
int carga = 0;
int DAC_data12 = 0;


/* PROGRAMA PRINCIPAL*/

int main(void)
{
    char i,j;                           // Índices para bucles
    WDTCTL = WDTPW + WDTHOLD;           // Disable WDT
    conf_reloj(16);                     // Reloj a 16MHz
    WDTCTL = WDTPW | WDTHOLD;           // stop watchdog timer
```

```
init_hw();                  // Configuración HARDWARE LCD - puertos I/O

_delay_cycles(500000);

conf_puertos();             // Configuración puertos E/S del micro

/* Disable the GPIO power-on default high-impedance mode to activate previously
configured port settings*/
PM5CTL0 &= ~LOCKLPM5;

conf_MIDI();
conf_timers();
conf_ADC();
conf_DAC_SAC();

initLCD();                  // Inicialización de la pantalla NOKIA
clearLCD();

_delay_cycles(80000);

frecMod = frecMod_ini_tremolo;         // Por defecto: efecto Tremolo
setAddr(0, 0);                         // Puntero de la pantalla a dirección inicial
writeStringToLCD("MIDI-SYNTH 2.0");

_delay_cycles(8000);

// Inicialización ADSR
tA=10;                  // 100ms
tD=10;                  // 100ms
tR=10;                  // 100ms
SUST=80;                // 80% del total (80*128/100)

Amp_i = 1000.0/(float)tA;
Amp_d = (float)(10*(100-SUST))/(float)tD;
Amp_r = (float)(10*SUST)/(float)tR;

// Escribir valores iniciales pantalla:

setChAddr(0, 2);                       // Escribir en la 3a línea de la LCD el instrumento
writeStringToLCD("Instrum:");
setChAddr(9, 2);
writeCharToLCD(instrum+'0');

setChAddr(0, 3);                       // Escribir en la 4a línea de la LCD la FASE
sprintf(enc,"Fase: %d   ",relat);
writeStringToLCD(enc);

setChAddr(0, 4);
writeStringToLCD("Tremolo: ");
writeCharToLCD(instrum+'0');

setChAddr(11, 4);
writeStringToLCD("OFF");

setChAddr(0,5);                        // Escribir instrumentos sumados en 4a línea
writeStringToLCD("Suma:");

__bis_SR_register(GIE);                // Habilitar las interrupciones

while(1)  // Bucle principal
{
   setChAddr(0,4);

   if(Tremolo_ON)
   {
```

```
            writeStringToLCD("Tremolo:");
            setChAddr(0, 3);                        // Escribir en la 4a línea la FASE
            sprintf(enc,"Fase: %d   ",relat);
            writeStringToLCD(enc);
        }
        else if (Vibrato_ON)
        {
            writeStringToLCD("Vibrato: ");
        }
        else if (ADSR_ON)
        {
            writeStringToLCD("Mod_ADSR:  ");
        }
        else // if (FM_ON)
        {
               writeStringToLCD("Mod_FM:  ");
               setChAddr(0, 3);
               sprintf(enc,"I_MOD: %d   ",relat);
               writeStringToLCD(enc);
        }
        setChAddr(9,1);                         // Posición puntero en la LCD para la carga L
        sprintf(enc,"L:%d",carga);              // Almacenar cadena "carga" en variable "enc"
        writeStringToLCD(enc);

        if (Msg)                                // Msg indica flanco subida/bajada en "nota"
        {
               Msg = 0;
               setAddr(0, 1);

               if(nota == 1) // Note ON
               {
                      writeStringToLCD("Note ON ");
                      nota_act = altura-OFFSET_NOTA;
                      frec = Freq[nota_act];
                      Fuerza = velocidad >> 2;
                      fase = 0;
                      if (ADSR_ON)
                      {
                             Amp = 0;
                             t_nota = 0;
                      }
         }
         else   // Si nota = 0 ó nota = 2 (ADSR)
                      {
                             writeStringToLCD("Note OFF");
                             if (ADSR_ON)
                             {
                                    t_nota_off = 0;
                             }
                      }
        }

      if(cambia_encoder)                // desfase (-63..63)
      {
             setChAddr(0, 3);           // Escribir en la 4a línea de la LCD

             if (FM_ON)
             {
               if(relat > 12) relat=0;
               if(relat <-12) relat=0;
               sprintf(enc,"I_MOD: %d   ",relat);
               I_MOD = relat;
             }
             else if (ADSR_ON)
             {
                 switch(indice_param)
                 {
```

```
            case 0:
                if(relat > 63 || relat < -63)
                  relat = 0;
                sprintf(enc,"Fase: %d   ",relat);
                break;

            case 1:
                tA += cambia_encoder;
                if(tA>100)
                  tA=100;
                if(tA<1)
                  tA=1;
                sprintf(enc,"tA: %dms", tA*10);
                break;

            case 2:
                tD += cambia_encoder;
                if(tD>100)
                  tD=100;
                if(tD<1)
                  tD=1;
                sprintf(enc,"tD: %dms", tD*10);
                break;

            case 3:
                SUST += cambia_encoder;
                if(SUST>100)
                  SUST=100;
                if(SUST<1)
                  SUST=1;
                sprintf(enc,"SUST: %d%%      ", SUST);
                break;

            case 4:
                tR += cambia_encoder;
                if(tR>100)
                  tR=100;
                if(tR<1)
                  tR=1;
                sprintf(enc,"tR: %dms  ", tR*10);
                break;

            }

            Amp_i = 1000.0/(float)tA;
            Amp_d = (float)(10*(100-SUST))/(float)tD;
            Amp_r = (float)(10*SUST)/(float)tR;

        }
        else      // Vibrato/Tremolo
        {
          if(relat > 63)relat = 0;
          if(relat < -63)       relat = 0;
          sprintf(enc,"Fase: %d   ",relat);
        }
        writeStringToLCD(enc);
        cambia_encoder = 0;            // Bajar bandera
    }
    lee_botones();

    if(BOT[3] == 1)                     // Incrementar instrumento mostrado
    {
        if (instrum < (NUMINST-1))
        {
            instrum++;                          // Efectuar incremento
            setChAddr(0, 2);                    // Escribir en 3a línea instr.considerado
```

```
            writeStringToLCD("Instrum:");
            setChAddr(9, 2);
            writeCharToLCD(instrum+'0');       // Dato de entrada en formato ASCII
        }
    }
    if(BOT[1] == 1)                    // Decrementa instrumento mostrado
    {
        if (instrum > 0)
        {
            instrum--;                 // Efectuar el decremento
            setChAddr(0, 2);
            writeStringToLCD("Instrum:");
            setChAddr(9, 2);
            writeCharToLCD(instrum+'0');
        }
    }
    if(BOT[2] == 1)            // Añadir el instrumento seleccionado (suma)
    {
          if (inst[instrum] == 0)
          {
                inst[instrum] = 1;        // Activar instrumento en vector suma
                if (ADSR_ON)
                      fase_ins[instrum] = relat;
                else
                {
                      if(relat >= 0)
                            fase_ins[instrum] = relat;
                      else
                            fase_ins[instrum] = (64-relat);
                }
          }
        else
        {
           inst[instrum] = 0;
        }
          calcula_onda = 1;
    }
    if (BOT[0] == 1)            // Pulsador cambio de efecto
    {
          faseMod = 0;

          switch (modulacion)
                {
                case 0:                              // Efecto Tremolo->Vibrato
                            modulacion = 1;
                            Tremolo_ON = 0;
                            Vibrato_ON = 1;
                            frecMod = frecMod_ini_vibrato;
                            break;
                case 1:                             // Efecto Vibrato->ADSR
                            modulacion = 2;
                            Vibrato_ON = 0;
                            ADSR_ON = 1;
                            break;
                case 2:                              // Efecto ADSR->FM
                            modulacion = 3;
                            ADSR_ON = 0;
                            FM_ON = 1;
                            break;
                case 3:                             // Efecto FM->Tremolo
                            modulacion = 0;
                            FM_ON = 0;
                            Tremolo_ON = 1;
                            frecMod = frecMod_ini_tremolo;

                            break;
                }
```

```
    }
    if(BOT[4] == 1)      // Asignar onda a modulación
    {
          if (ADSR_ON)
          {
                indice_param++;
                if(indice_param == 5)
                   indice_param = 0;

                cambia_encoder = 1;
          }
          else
          {
                InstMod = instrum;
                setChAddr(9, 4);
                writeCharToLCD(instrum+'0');
          }
    }

    lee_joystick();       // Leer potenciómetros y filtrar. Resultado: CH_F[0..3]

    if(Cambio_y)         // Función del joystick -> subir/bajar 2 semitonos
    {
          desplaz = (Joy_y >> 6);             // Escalar joystick: 4095/64 = 64

        if(desplaz > 31) // 64/2 = 32 -> Punto medio del joystick
   {
     if(nota_act <= 54)  // Max. notas en la tabla – aún podemos incrementar nota
     {
      frec = …
     …Freq[nota_act]+(((Freq[(nota_act+2)]-Freq[nota_act])*(desplaz-32))>>5);
           // Desplazamiento >> 5 -> dividir por 2^5=32 -> “desplaz” será [0,1]
     }
   }
   if(desplaz < 31)
   {
     if(nota_act >= 2)          // Aún podemos restar 2 hasta semitonos
     {
      frec = …
     …Freq[(nota_act)]-(((Freq[(nota_act)]-Freq[(nota_act-2)])*(32-desplaz))>>5);
      }
    }
}
if (Cambio_x)
{
          if (Joy_x < 40)                    // Activar efecto en cuestión
          {
                Env_on = 1;
                setChAddr(11, 4);           // Escribir en línea 5 (Modulación)
                writeStringToLCD("ON ");
          }
          if (Joy_x > 4000)                  // Desconectar efecto
          {
                Env_on = 0;
                setChAddr(11, 4);           // Escribir en línea 5 (Modulación)
                writeStringToLCD("OFF");
          }
   }
   if(calcula_onda)                          // Calcula la forma de onda (aditiva)
   {
          calcula_onda = 0;                  // Bajar bandera
          Inst_act = 0;
         setChAddr(0,5);                     // Escribir instrumento actual en 4a línea
          writeStringToLCD("Suma:         ");

          setChAddr(6,5);
```

```
            for(i=0; i<NUMINST; i++)      // Para cada uno de los 8 instrumentos...
             {
                    if(inst[i]==1)
                    {
                           writeCharToLCD(i+'0');       // Escribir instrumento actual
                           Inst_act ++;                 // = No. de instrumentos sumados
                    }
                    else
                    {
                           inst[i] = 0;
                    }
             }
             for(i=0; i<64; i++)          // 64 muestras almacenadas por instrumento
             {
                    forma[i] = 0;         // Resetear ALTURA del punto correspondiente
                    for(j=0; j<NUMINST; j++)
                    {
                           if(inst[j]==1)          // Para cada instrumento ACTIVO
                           {
                             forma[i] += SAMPLES[j][(64+i+fase_ins[j]) & 0x3F];
                           // fase+64 para compensar fases negativas
                           // Producto (& 0x3F (=63)) para no tomar valores > 63
                           }
                    }
                  if(Inst_act)                            // If Inst_act !=0
                  {
                    forma[i] = forma[i]/Inst_act;         // Escalar la onda
                  }
             }

      }
   }
}
#pragma vector = TIMER0_B0_VECTOR
__interrupt void LFO (void)
{
       If (Tremolo_ON)
       {
        Env = 64 + ((SAMPLES[InstMod][(faseMod>>8)&0x3F])/4);    // Env-> [32, 96]
        faseMod += frecMod;
       }
       else if (Vibrato_ON)
       {
        Env = 64+((SAMPLES[InstMod][(faseMod>>8)&0x3F]) / 4);    // Env-> [32, 96]
        frecMod = frec + Env;
        faseMod += frecMod;
       }
       else if (FM_ON)
       {

       }
       else // ADSR_ON
       {
        if(nota == 1)
        {
              t_nota++;
              if(t_nota <= tA)             // Fase "Attack"
              {
                     Amp += Amp_i;
                     if(Amp>1000)
                           Amp=1000;
              }
             else
             {
                     if(t_nota < (tA+tD))  // Fase "Decay"
                     {
                           Amp -= Amp_d;
```

```
                        if(Amp<(10*SUST))
                                Amp = 10*SUST;
                    }
                    else                        // Fase "Sustain"
                    {
                        t_nota = tA+tD;
                        Amp = 10*SUST;
                    }
                }
            }
            if(nota == 2)
            {
                 t_nota_off++;
                   if(t_nota_off < tR)
                   {
                       Amp -= Amp_r;
                       if(Amp<0)
                         Amp = 0;
                   }
                   else
                   {
                       nota = 0;
                   }
               }
            Amp_int = (int)(Amp/10.0);          // Amp_int es, como máximo 100
            }
}
#pragma vector = SAC0_SAC2_VECTOR
__interrupt void SAC2_ISR(void)
{
  switch(__even_in_range(SAC2IV,SACIV_4))
  {
    case SACIV_0: break;
    case SACIV_2: break;
    case SACIV_4:

    DAC_data12 = 0;                             // Resetear

    if(nota)                                    // Hay NOTE-ON
        {
           DAC_data12 = forma[(fase >> 8) & 0x3f];              // Leer onda
                // >> 8 equivale a dividir fase /2^8 = 256
               // Limitar el índice a 63 como máximo (0x3F = 63)

           if (!ADSR_ON)     // Variar intensidad  en todos efectos menos ADSR
           {
                 DAC_data12 = (DAC_data12 * Fuerza) >> 5;     // Modular con la fuerza
                 // Si fuerza es [0,64] y desplazamos 5 (/32) entonces fuerza [0,2]
           }
           if (Tremolo_ON)
           {
               fase += frec;
              if(Env_on)            // Modular con LFO - varía altura
              {DAC_data12 = (DAC_data12 * Env) >> 6;}
                      // Dividir /64: envolvente-> [0,5;1,5]
            }
            else if (Vibrato_ON)
            {
              if(Env_on)
              {
                 fase += frecMod;       // Modular con LFO - varía frecuencia
              }
              else
              {
                  fase += frec;
              }
```

```
            }
            else if (FM_ON)
            {
                  if(Env_on)
                    {
                     FM = SAMPLES[InstMod][(faseMod>>8) & 0x3F];       // faseMod/64;
                     // FM = oscilador 2
                     frecMod = frec + FM;
                     fase += frecMod;
                     faseMod += Freq[nota_act+I_MOD];
              }
              else
              {
                     fase += frec;
              }
              }
             else //if (ADSR_ON)
             {
                     if(Env_on)
              {
              DAC_data12 = (DAC_data12*Amp_int)/100;      // Amp_int será como máximo 1
              }
                     fase += frec;
            }
          }
        DAC_data12 = (DAC_data12+400)*5;
        DAC_data12 = 0xFFF & DAC_data12;  // Limitar a 12 bits para que no desborde!
        SAC2DAT = DAC_data12;
        carga = TB2R >> 3;

        break;
      default: break;
    }
}
```

❖ Midisynth.h

```
#ifndef MIDISYNTH_H_
#define MIDISYNTH_H_

// Funciones contenidas en Midisynth.c:

int leecanal(int ch);
void lee_joystick(void);
void lee_botones(void);

void conf_puertos(void);
void conf_MIDI(void);
void conf_ADC(void);
void conf_timers(void);
void conf_DAC_SAC(void);
void conf_reloj(char VEL);

extern char BOT[5];
extern int Joy_x, Joy_y;
extern char Cambio_x,Cambio_y;
extern char altura, velocidad;
extern char Msg;
extern int DAC_data12;
extern volatile char nota;
extern int ADSR_ON;
extern int increm;
extern int relat;
extern signed char cambia_encoder;
```

```
/************************************************************
 * Cabeceras relativas a la pantalla NOKIA5100
 * Basado en el ejemplo disponible en 43oh.com
/************************************************
 * Esquema de conexión de la pantalla al MSP430:
 *
 * Pines de la USCIB:
 *      DIN: MOSI - P1.7 (antes) -> AHORA P1.2
 *      CLK: P1.5 (antes) -> AHORA P1.1
 * Pines de control:
 *      CS:  P1.0 (antes) -> AHORA P1.5
 *      D/C: P1.2 (antes) -> AHORA P1.7
 * Si se quiere cambiar la configuración de los pines, se
 * deben tocar los defines a continuación
 *
 ************************************************/
// CLK input
#define LCD5110_SCLK_PIN BIT1
// DATA input
#define LCD5110_DN_PIN BIT2
// CHIP ENABLE - activo a nivel BAJO
#define LCD5110_SCE_PIN BIT7
// DATA (1)/COMMAND(0)
#define LCD5110_DC_PIN BIT5

#define LCD5110_SELECT P1OUT &= ~LCD5110_SCE_PIN            // Activar periférico (0)
#define LCD5110_DESELECT P1OUT |= LCD5110_SCE_PIN           // Desactivar periférico
#define LCD5110_SET_COMMAND P1OUT &= ~LCD5110_DC_PIN        // DATA mode - DC = 1
#define LCD5110_SET_DATA P1OUT |= LCD5110_DC_PIN            // COMMAND mode - DC = 0
#define LCD5110_COMMAND 0
#define LCD5110_DATA 1

#define SPI_MSB_FIRST UCB0CTL0 |= UCMSB
// or UCA0CTL0 |= UCMSB (USCIA) or USICTL0 &= ~USILSB (USI)
#define SPI_LSB_FIRST UCB0CTL0 &= ~UCMSB
// or UCA0CTL0 &= ~UCMSB or USICTL0 |= USILSB (USI)

void init_hw(void);
void writeToLCD(unsigned char dataCommand, unsigned char data);
void initLCD(void);

#define PCD8544_POWERDOWN 0x04
#define PCD8544_ENTRYMODE 0x02
#define PCD8544_EXTENDEDINSTRUCTION 0x01
#define PCD8544_DISPLAYBLANK 0x0
#define PCD8544_DISPLAYNORMAL 0x4
#define PCD8544_DISPLAYALLON 0x1
#define PCD8544_DISPLAYINVERTED 0x5// H = 0
#define PCD8544_FUNCTIONSET 0x20
#define PCD8544_DISPLAYCONTROL 0x08
#define PCD8544_SETYADDR 0x40
#define PCD8544_SETXADDR 0x80
#define PCD8544_HPIXELS 84
#define PCD8544_VBANKS 6
#define PCD8544_MAXBYTES 504 // PCD8544_HPIXELS * PCD8544_VBANKS// H = 1
#define PCD8544_SETTEMP 0x04
#define PCD8544_SETBIAS 0x10
#define PCD8544_SETVOP 0x80  //transform
#define NONE 0x00
#define FLIP_H 0x01
#define FLIP_V 0x02
#define ROTATE 0x04                 // 90 deg CW
#define ROTATE_90_CW ROTATE
#define ROTATE_90_CCW (FLIP_H | FLIP_V | ROTATE)
#define ROTATE_180 (FLIP_H | FLIP_V)
```

```
static const char font [][5] =            // Los caracteres ocupan 5x8 píxeles
    {
         {0x00, 0x00, 0x00, 0x00, 0x00}// (space)
        ,{0x00, 0x00, 0x5F, 0x00, 0x00}// !
        ,{0x00, 0x07, 0x00, 0x07, 0x00}// "
        ,{0x14, 0x7F, 0x14, 0x7F, 0x14}// #
        ,{0x24, 0x2A, 0x7F, 0x2A, 0x12}// $
        ,{0x23, 0x13, 0x08, 0x64, 0x62}// %
        ,{0x36, 0x49, 0x55, 0x22, 0x50}// &
        ,{0x00, 0x05, 0x03, 0x00, 0x00}// '
        ,{0x00, 0x1C, 0x22, 0x41, 0x00}// (
        ,{0x00, 0x41, 0x22, 0x1C, 0x00}// )
        ,{0x08, 0x2A, 0x1C, 0x2A, 0x08}// *
        ,{0x08, 0x08, 0x3E, 0x08, 0x08}// +
        ,{0x00, 0x50, 0x30, 0x00, 0x00}// ,
        ,{0x08, 0x08, 0x08, 0x08, 0x08}// -
        ,{0x00, 0x60, 0x60, 0x00, 0x00}// .
        ,{0x20, 0x10, 0x08, 0x04, 0x02}// /
        ,{0x3E, 0x51, 0x49, 0x45, 0x3E}// 0
        ,{0x00, 0x42, 0x7F, 0x40, 0x00}// 1
        ,{0x42, 0x61, 0x51, 0x49, 0x46}// 2
        ,{0x21, 0x41, 0x45, 0x4B, 0x31}// 3
        ,{0x18, 0x14, 0x12, 0x7F, 0x10}// 4
        ,{0x27, 0x45, 0x45, 0x45, 0x39}// 5
        ,{0x3C, 0x4A, 0x49, 0x49, 0x30}// 6
        ,{0x01, 0x71, 0x09, 0x05, 0x03}// 7
        ,{0x36, 0x49, 0x49, 0x49, 0x36}// 8
        ,{0x06, 0x49, 0x49, 0x29, 0x1E}// 9
        ,{0x00, 0x36, 0x36, 0x00, 0x00}// :
        ,{0x00, 0x56, 0x36, 0x00, 0x00}// ;
        ,{0x00, 0x08, 0x14, 0x22, 0x41}// <
        ,{0x14, 0x14, 0x14, 0x14, 0x14}// =
        ,{0x41, 0x22, 0x14, 0x08, 0x00}// >
        ,{0x02, 0x01, 0x51, 0x09, 0x06}// ?
        ,{0x32, 0x49, 0x79, 0x41, 0x3E}// @
        ,{0x7E, 0x11, 0x11, 0x11, 0x7E}// A
        ,{0x7F, 0x49, 0x49, 0x49, 0x36}// B
        ,{0x3E, 0x41, 0x41, 0x41, 0x22}// C
        ,{0x7F, 0x41, 0x41, 0x22, 0x1C}// D
        ,{0x7F, 0x49, 0x49, 0x49, 0x41}// E
        ,{0x7F, 0x09, 0x09, 0x01, 0x01}// F
        ,{0x3E, 0x41, 0x41, 0x51, 0x32}// G
        ,{0x7F, 0x08, 0x08, 0x08, 0x7F}// H
        ,{0x00, 0x41, 0x7F, 0x41, 0x00}// I
        ,{0x20, 0x40, 0x41, 0x3F, 0x01}// J
        ,{0x7F, 0x08, 0x14, 0x22, 0x41}// K
        ,{0x7F, 0x40, 0x40, 0x40, 0x40}// L
        ,{0x7F, 0x02, 0x04, 0x02, 0x7F}// M
        ,{0x7F, 0x04, 0x08, 0x10, 0x7F}// N
        ,{0x3E, 0x41, 0x41, 0x41, 0x3E}// O
        ,{0x7F, 0x09, 0x09, 0x09, 0x06}// P
        ,{0x3E, 0x41, 0x51, 0x21, 0x5E}// Q
        ,{0x7F, 0x09, 0x19, 0x29, 0x46}// R
        ,{0x46, 0x49, 0x49, 0x49, 0x31}// S
        ,{0x01, 0x01, 0x7F, 0x01, 0x01}// T
        ,{0x3F, 0x40, 0x40, 0x40, 0x3F}// U
        ,{0x1F, 0x20, 0x40, 0x20, 0x1F}// V
        ,{0x7F, 0x20, 0x18, 0x20, 0x7F}// W
        ,{0x63, 0x14, 0x08, 0x14, 0x63}// X
        ,{0x03, 0x04, 0x78, 0x04, 0x03}// Y
        ,{0x61, 0x51, 0x49, 0x45, 0x43}// Z
        ,{0x00, 0x00, 0x7F, 0x41, 0x41}// [
        ,{0x02, 0x04, 0x08, 0x10, 0x20}// "\"
        ,{0x41, 0x41, 0x7F, 0x00, 0x00}// ]
        ,{0x04, 0x02, 0x01, 0x02, 0x04}// ^
        ,{0x40, 0x40, 0x40, 0x40, 0x40}// _
        ,{0x00, 0x01, 0x02, 0x04, 0x00}// `
```

```
        ,{0x20, 0x54, 0x54, 0x54, 0x78}// a
        ,{0x7F, 0x48, 0x44, 0x44, 0x38}// b
        ,{0x38, 0x44, 0x44, 0x44, 0x20}// c
        ,{0x38, 0x44, 0x44, 0x48, 0x7F}// d
        ,{0x38, 0x54, 0x54, 0x54, 0x18}// e
        ,{0x08, 0x7E, 0x09, 0x01, 0x02}// f
        ,{0x08, 0x14, 0x54, 0x54, 0x3C}// g
        ,{0x7F, 0x08, 0x04, 0x04, 0x78}// h
        ,{0x00, 0x44, 0x7D, 0x40, 0x00}// i
        ,{0x20, 0x40, 0x44, 0x3D, 0x00}// j
        ,{0x00, 0x7F, 0x10, 0x28, 0x44}// k
        ,{0x00, 0x41, 0x7F, 0x40, 0x00}// l
        ,{0x7C, 0x04, 0x18, 0x04, 0x78}// m
        ,{0x7C, 0x08, 0x04, 0x04, 0x78}// n
        ,{0x38, 0x44, 0x44, 0x44, 0x38}// o
        ,{0x7C, 0x14, 0x14, 0x14, 0x08}// p
        ,{0x08, 0x14, 0x14, 0x18, 0x7C}// q
        ,{0x7C, 0x08, 0x04, 0x04, 0x08}// r
        ,{0x48, 0x54, 0x54, 0x54, 0x20}// s
        ,{0x04, 0x3F, 0x44, 0x40, 0x20}// t
        ,{0x3C, 0x40, 0x40, 0x20, 0x7C}// u
        ,{0x1C, 0x20, 0x40, 0x20, 0x1C}// v
        ,{0x3C, 0x40, 0x30, 0x40, 0x3C}// w
        ,{0x44, 0x28, 0x10, 0x28, 0x44}// x
        ,{0x0C, 0x50, 0x50, 0x50, 0x3C}// y
        ,{0x44, 0x64, 0x54, 0x4C, 0x44}// z
        ,{0x00, 0x08, 0x36, 0x41, 0x00}// {
        ,{0x00, 0x00, 0x7F, 0x00, 0x00}// |
        ,{0x00, 0x41, 0x36, 0x08, 0x00}// }
        ,{0x08, 0x08, 0x2A, 0x1C, 0x08}// ->
        ,{0x08, 0x1C, 0x2A, 0x08, 0x08} // <-
    };
void writeStringToLCD(const char *string);
void writeCharToLCD(char c);
void clearLCD();
void clearBank(unsigned char bank);
void setAddr(unsigned char xAddr, unsigned char yAddr);
void setChAddr(unsigned char xAddr, unsigned char yAddr);

#endif /* MIDISYNTH_H_ */
```

❖ Midisynth.c

```
/*********************************************/
/*     Funciones Midisynth                   */
/*********************************************/

// LIBRERÍAS:
#include <msp430.h>
#include "midisynth.h"

unsigned int ADC_result = 0;

int leecanal(int ch)
{
        // Por el canal ADC leemos el joystick; función empleada por leejoystick()

        ADCCTL0 &= !(0x0002);               // Deshabilitar ADC - ADCENC = 0
        ADCCTL0 |= ADCON;
        ADCMCTL0 &= 0xFFF0;                 // Borrar canal anterior - poner a 0 los 4 LSB
        ADCMCTL0 |= (ch & 0xf);             // Apuntar a nuevo canal - asignar ch a 4 LSB
        ADCCTL0 |= ADCENC | ADCSC;
        while(ADCCTL1 & ADCBUSY);           // Espera a terminar transmisión
        ADC_result = ADCMEM0;
```

```
        return ADC_result;                  // Devuelve valor leído - precisión de
12bits
}
int n;                                      // Contador
char altura, velocidad;
char Msg = 0;                           // Flag que indica note ON/OFF indistintamente

void int_rx(void)                       // Lee entrada midi (UART)- a cada ejecución lee un byte
{
        char Buf_RX;                    // Variable para almacenar lectura
        Buf_RX= UCA0RXBUF;              // Copiar buffer recepción UART (16BITS - ANTES 8 BITS!)

        // MIDI estándar
        if (Buf_RX == 0x80)             // LLega aviso note OFF
        {
                n = 0;                   // Resetear contador

                if (ADSR_ON)
                {
                        nota = 2;
                }
                else
                        nota = 0;

                Msg = 1;                        // Mensaje de NOTE-OFF completo
        }
        else if (Buf_RX == 0x90)                // Llega  note ON mensaje de Altura/Velocidad
        {
                n = 1;
        }
        else
        {
                if (n == 1)              // Altura
                {
                        altura = Buf_RX;
                        n++;
                }
                else if (n == 2)        // Velocidad
                {
                        n = 0;
                        velocidad = Buf_RX;
                        nota = 1;
                        Msg = 1;          // Mensaje  note_ON completo
                }
        }
        /* YAMAHA CLAVINOVA
        if (Buf_RX != 0xFE)     // No es "active sensing"
                {
                        if (Buf_RX == 0x90)
                         // LLega aviso note ON/OFF (puede llegar altura directamente...)
                                n = 0;
                        else                            // Altura/velocidad
                        {
                                if (n == 0)    // Altura
                                {
                                        altura = Buf_RX;
                                        n++;
                                }
                                else if (n == 1)      // Velocidad
                                {
                                        n = 0;
                                        velocidad = Buf_RX;

                                        if (velocidad == 0)
                                                nota = 0;
                                        else
                                                nota = 1;
```

```
                                        Msg = 1;              // Se ha producido el flanco
                                }
                        }
                }
                else
                        n = 0;          // Resetear contador
                */
}
int CH[2][5], CH_F[2];
void filtra(void)                       // Filtra info leída por canales A3 y A4 del ADC
{
        int i,j;
        for (i=0; i<2; i++)                                 // Para los 2 canales...
                {
                        CH_F[i] = 0;                        // Reseteo variable de filtro

                        for(j=3; j>=0; j--)                 // Repetimos 4 veces para cada canal
                        {
                                CH_F[i] += CH[i][j];
                                CH[i][j+1] = CH[i][j];
                        }
                        CH_F[i] = CH_F[i] >> 2;
                        // Dividir /4! Media de las 4 últimas lecturas! :)
                }
}
int CHIN[2] = {3,4};                            // Canales a leer: serán el A3 y el A4
int Cambio_analog[2], ch_f_ant[2];
int Joy_x, Joy_y;
char Cambio_x, Cambio_y;
void lee_joystick(void)
{
        int i;
        for(i=0; i<2; i++)                    // Leer los 2 canales ADC ( A3 y A4 )
        {
                CH[i][0] = (leecanal(CHIN[i]));
        }
        filtra();

        for(i=0; i<2; i++)                  // Para cada canal... (x2)
        {
                if(CH_F[i] >= ch_f_ant[i])
                {
                        Cambio_analog[i] = (CH_F[i]-ch_f_ant[i]);
                        // Variable cambio valdrá 0 ó >0 -> se considerará como 1 lógico
                }
                else
                {
                        Cambio_analog[i] = (ch_f_ant[i]-CH_F[i]);
                }
                ch_f_ant[i] = CH_F[i];      // Guardar valor actual para siguiente vuelta
        }
        Cambio_x = Cambio_analog[0];
        Joy_x = CH_F[0];

        Cambio_y = Cambio_analog[1];
        Joy_y = CH_F[1];                    // Valor que variará entre 0 y 4095
}
char BOT[5] = {0,0,0,0,0};
void lee_botones(void)              // ACTIVOS a nivel BAJO
{
    char i;                         // Contador

    for(i=0; i<4; i++)              // Recorre botones del 1 al 4! (encoder APARTE)
    {
        if(!(P6IN & (1 << i)))      // Lectura entrada - si el botón i esta activo...
```

```
        {
            if(BOT[i] < 2)   // Para que el botón SOLO valga 0 ó 1!!
              BOT[i]++;
        }
        else
        {
            BOT[i]=0;               // Resetear
        }
    }
    // Pulsación del encoder - se corresponde con el botón 5 (BOT[4])
    if (!(P3IN & BIT3))
    {
        if(BOT[4]<2)                // Ídem q botones 1-4
              BOT[4]++;
    }
    else
       BOT[4]=0;                    // Resetear
}
void conf_puertos(void)
{
    // Pines de E/S

       // MIDI - ENTRADA!

        P1SEL0 |= BIT6;             // P1SELx = 01
        P1SEL1 |= 0;

    // PANTALLA - SALIDA!: Ya configurado en nokia.c

    // ENCODER - ENTRADA!

        // Botón 5: (Pulsación encoder = forma de onda para la modulación)

         P3OUT |= BIT3;
        P3REN |= BIT3;                      // Habilitación resistencia PULL-UP


    // BOTONES - ENTRADA!

        P6OUT |= BIT0 | BIT1 | BIT2;        // Botones 1, 2, 3 y 4
        P6REN |= BIT0 | BIT1 | BIT2;        // Habilitación resistencia PULL-UP

        // Botón selección efectos:

        P6OUT |= BIT3;                      // P4.1 - cambiar más tarde
        P6REN |= BIT3;

    // JOYSTICK - ENTRADA! - P1.3 Y P1.4

        // ADC modo simple, canales 3,4 activos
        P1DIR &= ~ (BIT3 + BIT4);                // Por defecto a 0 -> ENTRADA

        // Configurar ADC A3 y A4 pin
        P1SEL0 |= BIT3 | BIT4;
        P1SEL1 |= BIT3 | BIT4;

     // SALIDA DE AUDIO - DAC!! (NOVEDAD)

        P3SEL0 |= BIT1;                       // Select P3.1 como función OA2O
        P3SEL1 |= BIT1;                       // OA como buffer para DAC

     // Interrupciones puerto 2 - para ENCODER rotativo

        P2IES = 0;          // 0b = P2IFG flag activa con transición low-to-high
        P2IFG = 0;          // 0b = No hay interrupción pendiente
        P2IE |= BIT4; // P2.4 -> 1b = interrupt enabled
}
```

```
void conf_MIDI(void)
{
       // MIDI! : USCI como UART a 31250
       UCA0CTLW0 |= UCSWRST;
       UCA0CTLW0 = UCSSEL__SMCLK | UCSWRST;
       UCA0BRW = 512;
       UCA0CTLW0 &= ~UCSWRST;
       UCA0IFG &= ~(UCRXIFG);               // Poner a cero la bandera
       UCA0IE |= UCRXIE;                    // Activar interrupciones
}
void conf_ADC(void)
{
       ADCCTL0 &= ~ADCENC;                 // Disable ADC
       ADCCTL0 = ADCON  | ADCSHT_3 ;       // Habilitar (ADC10ON),tiempo s/h: 32*clk
       ADCCTL1 = ADCSSEL_3 | ADCSHP;
       ADCCTL2 &= ~ADCRES;                 // clear ADCRES in ADCCTL
       ADCCTL2 |= ADCRES_2;                // 12-bit conversion results
       ADCMCTL0 = ADCSREF_0;
       //ADCIE |= ADCIE0;                   // No! entra en "trampa" interrupt
}
void conf_DAC_SAC(void)
{
         SAC2DAC = DACSREF_0 + DACLSEL_2 + DACIE;
              // Primary ref (VCC = 3.3V), device specific 0: TimerB2.1
         SAC2DAT = DAC_data12;                  // Initial DAC data
         SAC2DAC |= DACEN;                      // Enable DAC
         SAC2OA = NMUXEN + PMUXEN + PSEL_1 + NSEL_1;
          // Select positive and negative pin input
         SAC2OA |= OAPM;                        // Select low speed and low power mode
         SAC2PGA = MSEL_1;                      // Set OA as buffer mode
         SAC2OA |= SACEN + OAEN;                // Enable SAC and OA
}
void conf_timers(void)
{
           // SALIDA DE AUDIO: - configuración del timer como DAC

               // Use TB2.1 as DAC hardware trigger
                 TB2CCR0 = 799;                             // PWM Period/2
                 TB2CCTL1 = OUTMOD_6;                       // TBCCR1 toggle/set
                 TB2CCR1 = 400;                             // TBCCR1 PWM duty cycle
                 TB2CTL = TBSSEL__SMCLK | MC_1 | TBCLR;     // SMCLK,up mode,clear TBR

           /*******************************************************/
           // TB1CCR0 como moduladora (LFO), a 100Hz de muestreo
           /*******************************************************/

           // TB1CCR0 para marcar paso de tiempo de nota: Se fija un periodo de 10ms

                TB0CCTL0 = CCIE;
                TB0CCTL1 = 0;
                TB0CTL = TBSSEL_2 + MC_1 + ID_3;  // 16/8= 2M
                TB0CCR0 = 19999;                  // 2M/20000 = 0.1kHz
}
// Función de configuración del reloj:

void Software_Trim(void);             // Software Trim to get the best DCOFTRIM value
#define MCLK_FREQ_MHZ 16              // MCLK = 1MHz

void conf_reloj(char VEL)
{
   __bis_SR_register(SCG0);  // Disable FLL to enable writing the registers.
   CSCTL3 |= SELREF__REFOCLK;   // Set REFO as FLL reference source

    switch(VEL){
    case 1:
```

```
        CSCTL1 = DCOFTRIMEN_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL_0;
        // DCOFTRIM = 3, DCO Range = 1MHz
        CSCTL2 = FLLD_0 + 30;
        // DCODIV = 1MHz

       break;

    case 8:

        CSCTL1 = DCOFTRIMEN_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL_3;
        // DCOFTRIM=3, DCO Range = 8MHz
        CSCTL2 = FLLD_0 + 243;
        // DCODIV = 8MHz
         break;

    case 12:

        CSCTL1 = DCOFTRIMEN_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL_4;
        // DCOFTRIM=3, DCO Range = 12MHz
         break;

    case 16:

        CSCTL1 = DCOFTRIMEN_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL_5;
        // DCOFTRIM=3, DCO Range = 16MHz
        CSCTL2 = FLLD_0 + 487;
        // DCOCLKDIV = 16MHz
         break;

    default:          // 1MHz por defecto

       CSCTL1 = DCOFTRIMEN_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL_0;
        // DCOFTRIM=3, DCO Range = 1MHz
        CSCTL2 = FLLD_0 + 30;
        // DCODIV = 1MHz
         break;
    }
    __delay_cycles(3);
    __bic_SR_register(SCG0);        // Enable FLL
    Software_Trim();                // Software Trim to get the best DCOFTRIM value

    CSCTL4 = SELMS__DCOCLKDIV;
    CSCTL5 = DIVM_0 | DIVS_0;
}
void Software_Trim(void)
{
    unsigned int oldDcoTap = 0xffff;
    unsigned int newDcoTap = 0xffff;
    unsigned int newDcoDelta = 0xffff;
    unsigned int bestDcoDelta = 0xffff;
    unsigned int csCtl0Copy = 0;
    unsigned int csCtl1Copy = 0;
    unsigned int csCtl0Read = 0;
    unsigned int csCtl1Read = 0;
    unsigned int dcoFreqTrim = 3;
    unsigned char endLoop = 0;

    do
    {
        CSCTL0 = 0x100;                         // DCO Tap = 256
        do
        {
            CSCTL7 &= ~DCOFFG;                  // Clear DCO fault flag
        }while (CSCTL7 & DCOFFG);               // Test DCO fault flag

        __delay_cycles((unsigned int)3000 * MCLK_FREQ_MHZ);
         // Wait FLL lock status (FLLUNLOCK) to be stable
```

```
        // Suggest to wait 24 cycles of divided FLL reference clock
        while((CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)) && ((CSCTL7 & DCOFFG) == 0));

        csCtl0Read = CSCTL0;                    // Read CSCTL0
        csCtl1Read = CSCTL1;                    // Read CSCTL1

        oldDcoTap = newDcoTap;                   // Record DCOTAP value of last time
        newDcoTap = csCtl0Read & 0x01ff;         // Get DCOTAP value of this time
        dcoFreqTrim = (csCtl1Read & 0x0070)>>4; // Get DCOFTRIM value

        if(newDcoTap < 256)                     // DCOTAP < 256
        {
            newDcoDelta = 256 - newDcoTap;      // Delta value between DCPTAP and 256
            if((oldDcoTap != 0xffff) && (oldDcoTap >= 256)) // DCOTAP cross 256
                endLoop = 1;                    // Stop while loop
            else
            {
                dcoFreqTrim--;
                CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);
            }
        }
        else                                    // DCOTAP >= 256
        {
            newDcoDelta = newDcoTap - 256;      // Delta value between DCPTAP and 256
            if(oldDcoTap < 256)                 // DCOTAP cross 256
                endLoop = 1;                    // Stop while loop
            else
            {
                dcoFreqTrim++;
                CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);
            }
        }
        if(newDcoDelta < bestDcoDelta)          // Record DCOTAP closest to 256
        {
            csCtl0Copy = csCtl0Read;
            csCtl1Copy = csCtl1Read;
            bestDcoDelta = newDcoDelta;
        }
    }while(endLoop == 0);                       // Poll until endLoop == 1

    CSCTL0 = csCtl0Copy;                        // Reload locked DCOTAP
    CSCTL1 = csCtl1Copy;                        // Reload locked DCOFTRIM
    while(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)); // Poll until FLL is locked
}
// Función de interrupción de la USCI - UART de la entrada midi
#pragma vector = EUSCI_A0_VECTOR
__interrupt void USCI0RX_ISR_HOOK(void)
{
    // USER CODE START (section: USCI0RX_ISR_HOOK)
    // replace this comment with your code
        int_rx();
    // USER CODE END (section: USCI0RX_ISR_HOOK)
}
int increm = 0, relat = 0;
signed char cambia_encoder = 0;

// Función de interrupción del encoder rotativo
#pragma vector = PORT2_VECTOR
__interrupt void Encoder (void)
{
    if(P2IFG & BIT4)  // Interrupción por cambio P2.4 a nivel ALTO - Canal A
    {
        P2IFG = 0;          // Bajar la bandera
        if(P2IN & BIT5)     // Si la señal_A = señal_B (giro a la derecha - CW)
        {
            increm++;
```

```
        }
         else                // Si la señal_A != señal_B (giro a la izquierda - CCW)
              {
                    increm--;
              }

        if(increm == 3)
        {
            cambia_encoder = 1;     // Activar bandera para aumento de fase relativa
            increm = 0;             // Resetear contador
            if(relat < 30000)       // Saturación del encoder
                relat++;
         }
         if(increm == -3)
         {
            increm = 0;              // Resetear contador
            cambia_encoder = -1;     // Activar bandera para decremento de fase relativa
            if(relat > -30000)       // Saturación del encoder
                relat--;
         }
     }
}
```

❖ Nokia5110.c

```
/*********************************************/
/*  nokia5110.c                              */
/*********************************************/

#include "msp430fr2355.h"
#include "midisynth.h"

void init_hw(void)             // Inicialización HARDWARE - PUERTOS I/O
{
    P1OUT |= LCD5110_SCE_PIN + LCD5110_DC_PIN;
       // Puertos P1.5 (DC) y P1.7 (SCE) salida a nivel alto
    P1DIR |= LCD5110_SCE_PIN + LCD5110_DC_PIN;
       // Puertos P1.5 (DC) y P1.7 (SCE) como OUTPUT

    // setup USCIB
    P1SEL0 |= LCD5110_SCLK_PIN | LCD5110_DN_PIN;

    UCB0CTLW0 |= UCSWRST;     // **Put state machine in reset**
    UCB0CTLW0 |= UCCKPH + UCMSB + UCMST + UCSYNC;    // 3-pin, 8-bit SPI master
    UCB0CTLW0 |= UCSSEL_2;                           // SMCLK
    UCB0BR0 = 1;              // 1:1   OJO: si velocidad a 16MHz, poner 16
    UCB0BR1 = 0;
    UCB0CTLW0 &= ~UCSWRST;    // clear SW
}
void initLCD()
{
    writeToLCD(LCD5110_COMMAND, PCD8544_FUNCTIONSET | PCD8544_EXTENDEDINSTRUCTION);
    writeToLCD(LCD5110_COMMAND, PCD8544_SETVOP | 0x3F);
    writeToLCD(LCD5110_COMMAND, PCD8544_SETTEMP | 0x02);
    writeToLCD(LCD5110_COMMAND, PCD8544_SETBIAS | 0x03);
    writeToLCD(LCD5110_COMMAND, PCD8544_FUNCTIONSET);
    writeToLCD(LCD5110_COMMAND, PCD8544_DISPLAYCONTROL | PCD8544_DISPLAYNORMAL);
}
void writeToLCD(unsigned char dataCommand, unsigned char data)
{
    if(dataCommand)                    // #define LCD5110_DATA 1
        {
            LCD5110_SET_DATA;       // DATA mode - pin DC = 1
        }
```

```
    else                                // #define LCD5110_COMMAND 0
        {
            LCD5110_SET_COMMAND;    // COMMAND mode - pin DC = 0
        }

    LCD5110_SELECT;                 // #define LCD5110_SELECT P1OUT &= ~LCD5110_SCE_PIN
         // #define LCD5110_SCE_PIN BIT0
         // Equivalente~ :  P1OUT &= ~BIT0 -> CHIP select activo, a nivel BAJO

    UCB0TXBUF = data;           // Copiar dato en el bus de transmisión de la UART

    while (!(UCB0IFG&UCTXIFG));
         LCD5110_DESELECT;           // Desactivar el periférico
}
void writeStringToLCD(const char *string)       // Escribir cadena de caracteres
{
    while(*string)
    {
        writeCharToLCD(*string++);
    }
}
void writeCharToLCD(char c)     // Escribir carácter único
{
    unsigned char i;            // Contador de filas en la pantalla LCD

    for(i = 0; i < 5; i++)      // A cada vuelta del bucle manda un byte (8bits)
    {
        writeToLCD(LCD5110_DATA, font[c - 0x20][i]);
         // Font es una matriz que contiene todos los caracteres ASCII
    }
    writeToLCD(LCD5110_DATA, 0);   // Después de mandar un caracter, mando un espacio
}
void clearLCD()
{
    int c = 0;
    setAddr(0, 0);

    while(c < PCD8544_MAXBYTES)   // No. máx de bytes = 48x84 píxeles / 8
    {
        writeToLCD(LCD5110_DATA, 0);
        c++;
    }
    setAddr(0, 0);               // Vuelvo a poner el puntero en el píxel (0,0)
}

void clearBank(unsigned char bank)  // Borrar banco de bytes (6 bancos de 84 bytes)
{
    int c = 0;
    setAddr(0, bank);

    while(c < PCD8544_HPIXELS)         // No. píxeles horizontales = 84
    {
        writeToLCD(LCD5110_DATA, 0);
        c++;
    }
    setAddr(0, bank);
}
void setAddr(unsigned char xAddr, unsigned char yAddr)
{
    writeToLCD(LCD5110_COMMAND, PCD8544_SETXADDR | xAddr);
       // #define PCD8544_SETXADDR 0x80
    writeToLCD(LCD5110_COMMAND, PCD8544_SETYADDR | yAddr);
  // #define PCD8544_SETYADDR 0x40
}
void setChAddr(unsigned char xAddr, unsigned char yAddr)
// Establecer posición del carácter
```

```
{
    if(xAddr < 14)
       // Como máximo cabrán 84(píxeles)/6(píxeles/caracter) = 14 caracteres
    {
        xAddr = xAddr * 6;
    }
    else
    {
        xAddr = 0;
    }
    writeToLCD(LCD5110_COMMAND, PCD8544_SETYADDR | yAddr);
    writeToLCD(LCD5110_COMMAND, PCD8544_SETXADDR |xAddr);
}
```

❖ Formas2.h

```
#ifndef FORMAS_H_
#define FORMAS_H_

#define NUMINST 8
const int SAMPLES[NUMINST][64] = {

// senoidal

{0,13,25,37,49,60,71,81,91,99,106,113,118,122,126,127,128,127,126,122,118,113,106,99,91,81,71,
60,49,37,25,13,0,-13,-25,-37,-49,-60,-71,-81,-91,-99,-106,-113,-118,-122,-126,-127,-128,-127,-
126,-122,-118,-113,-106,-99,-91,-81,-71,-60,-49,-37,-25,-13},

// square 1

{-128,-64,0,64,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,
127,127,127,127,127,127,127,127,64,0,-64,-128,-128,-128,-128,-128,-128,-128,-128,-128,
-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,
-128,-128},

// square 2

{0,55,100,118,127,127,127,127,127,127,127,127,127,127,127,100,63,27,0,0,0,0,0,0,0,0,0,0,-8,
-27,-19,0,18,27,9,0,0,0,0,0,0,0,0,0,0,-26,-63,-100,-128,-128,-128,-128,-128,-128,-128,-128,
-128,-128,-128,-119,-101,-57},

// Electric_guitar
{11,123,122,121,121,120,120,39,21,55,106,117,116,115,115,106,38,-101,-127,-127,-126,-126,
-125,-91,11,70,72,-27,-124,-123,-123,-121,-120,-87,120,120,120,119,118,63,-67,-123,-122,-121,
-120,-118,119,120,119,118,118,-124,-122,-122,-122,-120,-79,72,95,30,-63,-120,-119,-9},

// E_organ
{7,120,128,126,121,114,107,100,93,86,79,72,64,57,50,43,36,29,21,14,7,-0,-7,-14,-21,-29,-36,
-43,-50,-57,-62,-55,54,62,57,50,43,36,29,22,15,7,0,-7,-14,-21,-28,-35,-43,-50,-57,-64,-71,
-78,-86,-93,-100,-107,-114,-120,-126,-128,-120,-14},

// oboe 5
{1,10,22,34,46,59,71,82,92,101,108,113,115,114,108,100,89,75,59,38,14,-10,-36,-62,-83,-101,
-114,-122,-127,-128,-125,-118,-107,-94,-79,-65,-49,-32,-18,
-6,7,19,31,39,45,45,43,40,34,26,18,10,4,-1,-7,-13,-16,-18,-18,-18,-17,-14,-9, 0},

// hdrawns 05
{0,43,63,68,78,83,90,94,100,106,110,114,116,118,119,120,121,121,121,121,121,121,120,119,117,11
3,110,105,90,73,38,-7,-42,-64,-80,-87,-96,-105,-111,-120,-123,-125,-128,-128,-128,-128,-127,
-127,-126,-125,-125,-122,-118,-115,-112,-106,-102,-91,-63,-39,-20,-9,-5,-1},

// vgame 03
{0,8,15,22,30,37,44,51,59,66,73,81,88,95,102,110,117,124,126,120,105,88,73,61,53,47,41,34,27,
19,12,4,-3,-11,-18,-25,-33,-40,-47,-55,-62,-69,-76,-84,-91,-98,-105,-113,-120,-127,-125,-114,
-97,-81,-67,-58,-50,-44,-38,-31,-24,-16,-8,-1},
```

```
};

#define     Hz  *64L*256/20000  // Normalizado de frecuencias
#define OFFSET_NOTA 36          // Primera nota en la tabla: MIDI 36

const int Freq[] = {
                    (int)( 65.4064 Hz), //DO0
                    (int)( 69.2957 Hz),
                    (int)( 73.4162 Hz),
                    (int)( 77.7817 Hz),
                    (int)( 82.4069 Hz),
                    (int)( 87.3071 Hz),
                    (int)( 92.4986 Hz),
                    (int)( 97.9989 Hz),
                    (int)(103.8262 Hz),
                    (int)(110.0000 Hz),
                    (int)(116.5409 Hz),
                    (int)(123.4708 Hz),
                    (int)(130.81278 Hz),  //DO1
                    (int)(138.59132 Hz),
                    (int)(146.83238 Hz),
                    (int)(155.56349 Hz),
                    (int)(164.81378 Hz),
                    (int)(174.61412 Hz),
                    (int)(184.99721 Hz),
                    (int)(195.99772 Hz),
                    (int)(207.65235 Hz),
                    (int)(220.00000 Hz),
                    (int)(233.08188 Hz),
                    (int)(246.94165 Hz),
                    (int)(261.62557 Hz),  //DO2
                    (int)(277.18263 Hz),
                    (int)(293.66477 Hz),
                    (int)(311.12698 Hz),
                    (int)(329.62756 Hz),
                    (int)(349.22823 Hz),
                    (int)(369.99442 Hz),
                    (int)(391.99544 Hz),
                    (int)(415.30470 Hz),
                    (int)(440.00000 Hz),
                    (int)(466.16376 Hz),
                    (int)(493.88330 Hz),
                    (int)(523.25113 Hz),  //DO3
                    (int)(554.36526 Hz),
                    (int)(587.32954 Hz),
                    (int)(622.25397 Hz),
                    (int)(659.25511 Hz),
                    (int)(698.45646 Hz),
                    (int)(739.98885 Hz),
                    (int)(783.99087 Hz),
                    (int)(830.60940 Hz),
                    (int)(880.00000 Hz),
                    (int)(932.32752 Hz),
                    (int)(987.76660 Hz),
                    (int)(1046.50226 Hz), //DO4
                    (int)(1108.73052 Hz),
                    (int)(1174.65907 Hz),
                    (int)(1244.50793 Hz),
                    (int)(1318.51023 Hz),
                    (int)(1396.91293 Hz),
                    (int)(1479.97769 Hz),
                    (int)(1567.98174 Hz),
                    (int)(1661.21879 Hz)  //LA4
};

#endif /* FORMAS_H_ */
```

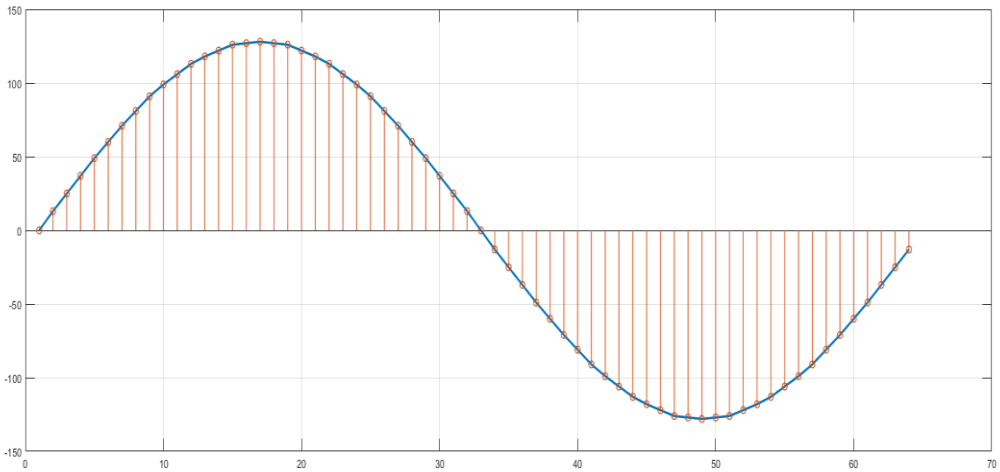
ANEXO II

Se incluye en este apartado el código Matlab implementado para la representación gráfica y vectorial de las formas de onda, a partir de distintas muestras descargadas del repositorio AdventureKid [21], en formato .wav.

```
% Código

filename = 'AKWF_piano_0015.wav';
% Escribir aquí el nombre del archivo .wav e incluirlo en la misma carpeta
info = audioinfo(filename);
[y, Fs] = audioread(filename);

t = 0:seconds(1/Fs):seconds(info.Duration);
t= t(1:end-1);
t_muest = linspace(0,t(end),64);
t_muest_doub = seconds(t_muest);
y = y*128;                          % Escalamos en el intervalo [-128,+128]

y_muest = round (interp1(seconds(t), y, t_muest_doub, 'linear'));
allOneString = sprintf('%.0f,' , y_muest)  % Obtener el vector

figure
plot(t,y)
grid on
hold;
stem(t_muest, y_muest);
```

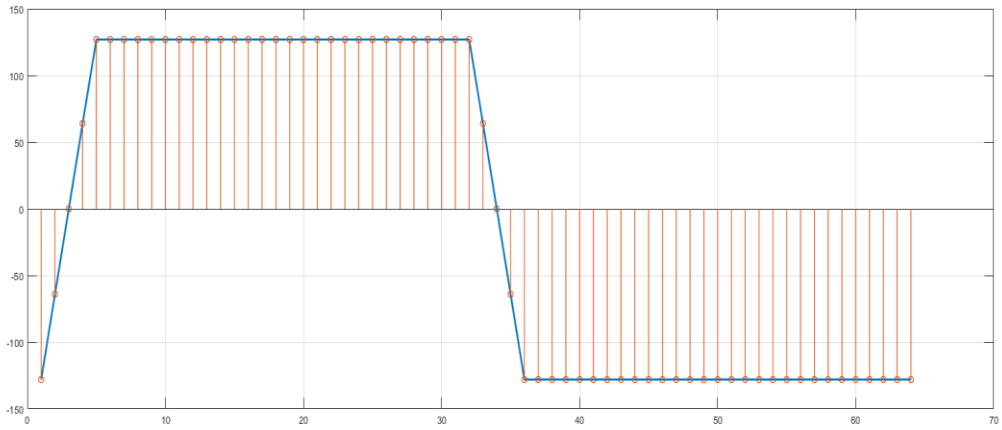
Las formas de onda asociadas a los 8 instrumentos[42], numerados del 0 al 7 son las que se muestran a continuación:

1. Senoidal.

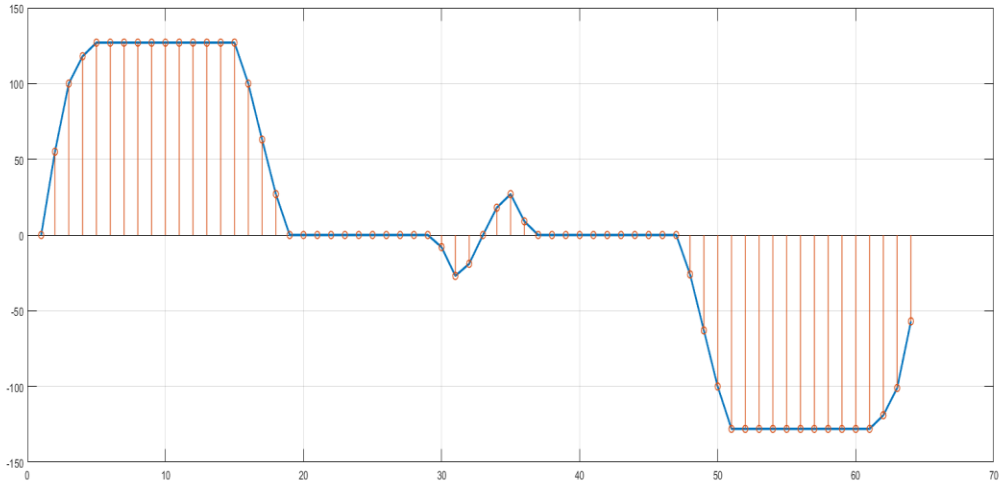


[42] Recordar que geerábamos 8 timbres distintos a partir de estas ondas, algunas de ellas asociadas a instrumentos reales y otras de forma genérica (senoidal, cuadrada) o aleatoria.

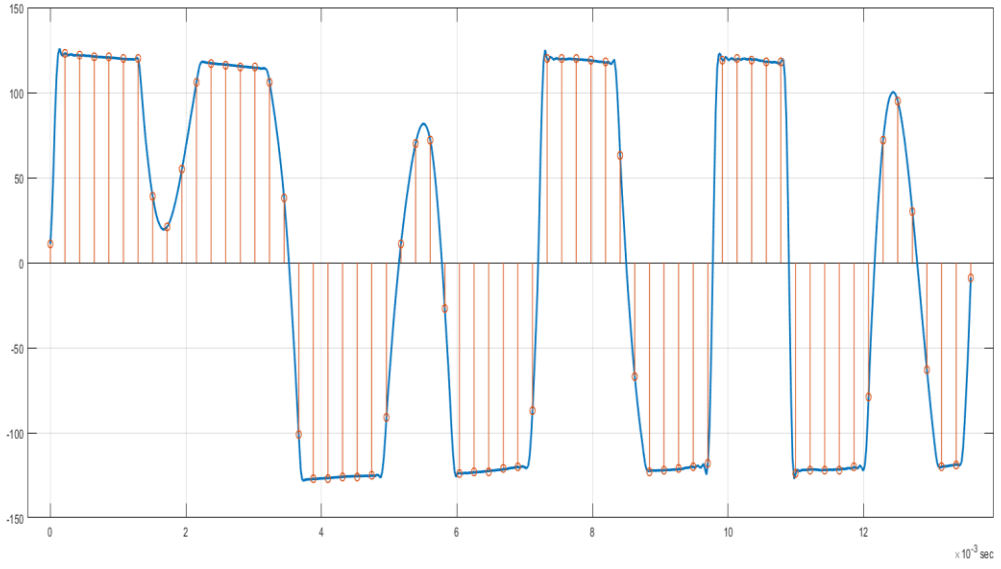
2. “Square” 1



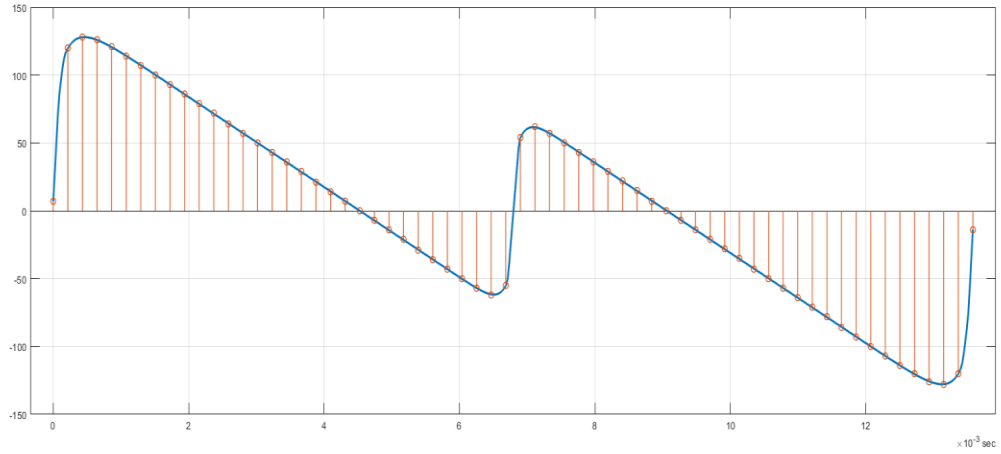
3. “Square” 2



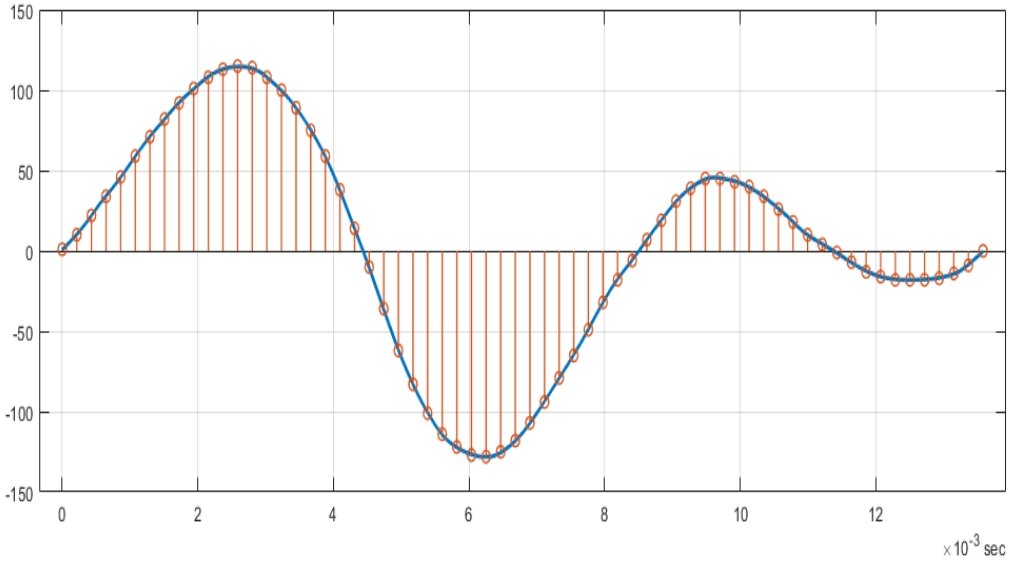
4. “Electric Guitar”



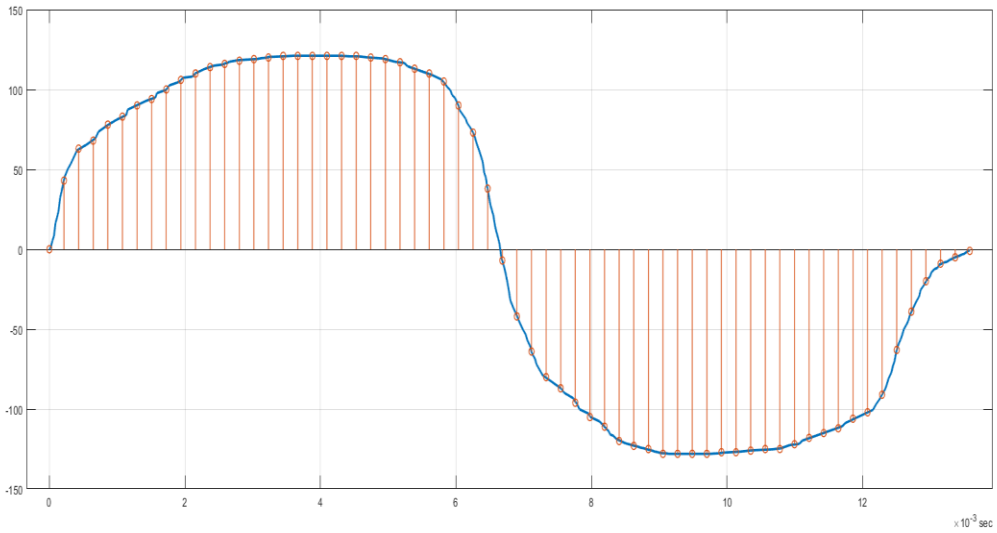
5. “Electric Organ”



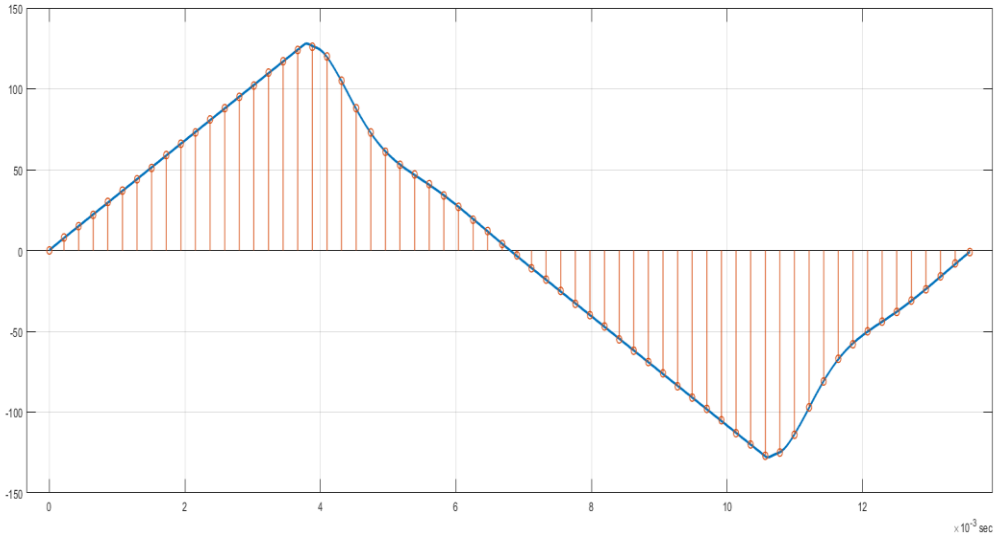
6. “Oboe”



7. “Hand Drawn”

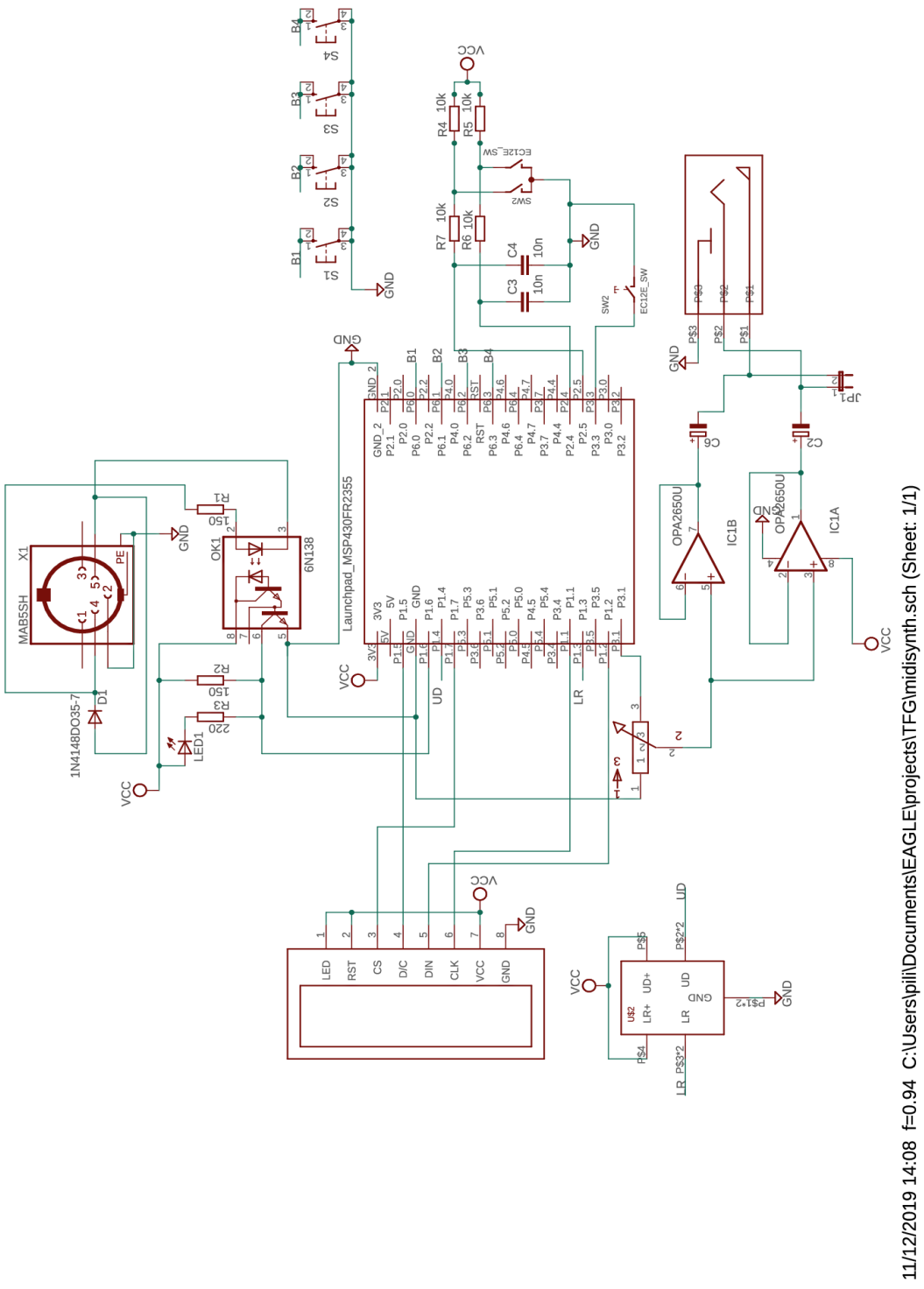


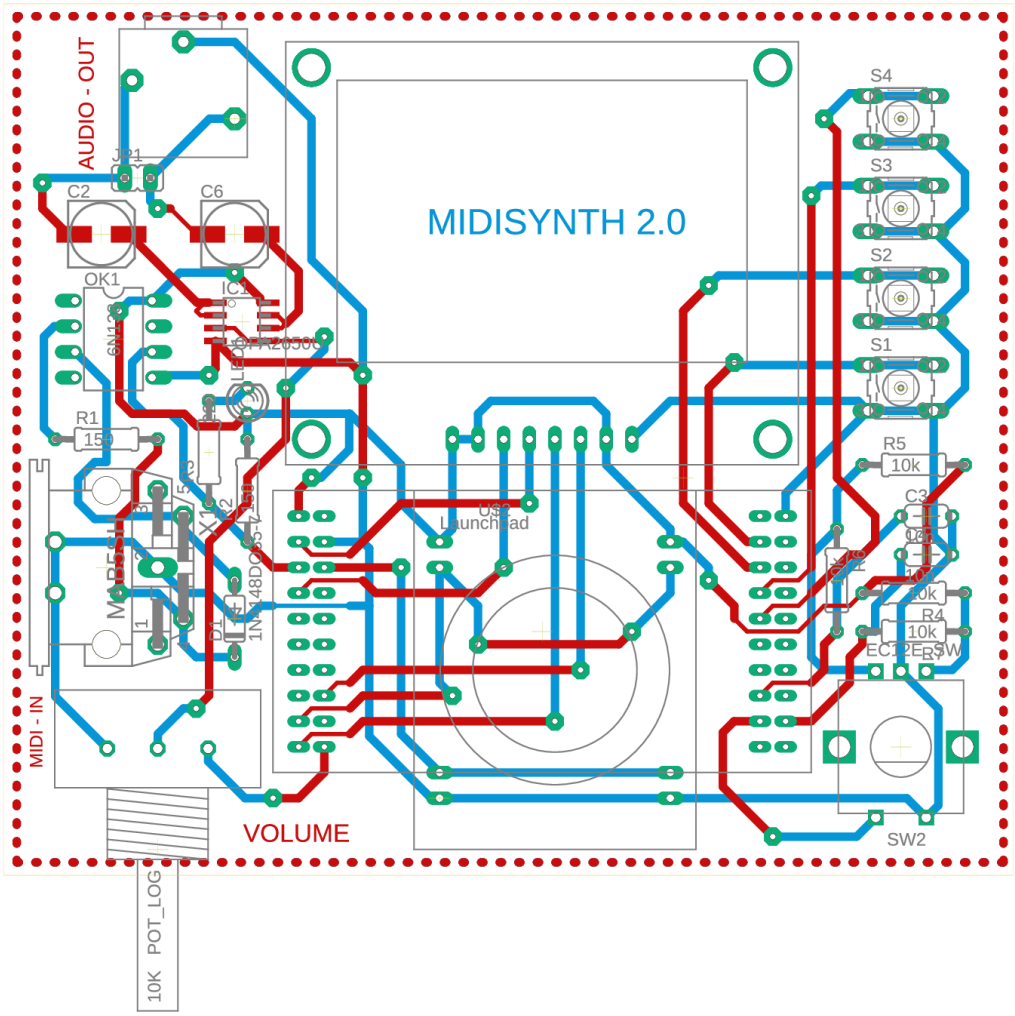
8. “Video Game”



ANEXO III

Diagrama esquemático y fotolito del circuito impreso.





11/12/2019 16:21 f=1.70 C:\Users\pili\Documents\EAGLE\projects\TFG\midisynth.brd