

SynthIce: Sintetizador basado en FPGA para Síntesis, Modulación y Reproducción de Audio Digital

Sanchez Cardenas Samuel David, samsanchezca@unal.edu.co,
 Álvarez Chaparro Paula Valentina, palvarezch@unal.edu.co,
 Segovia Arteaga Dayanna, daarteagas@unal.edu.co,

Resumen—Este trabajo presenta el diseño e implementación de *SynthIce*, un sintetizador digital basado en FPGA que permite la síntesis y modulación de sonido en tiempo real. El sistema se desarrolla sobre la FPGA *BlackIce40* utilizando el lenguaje de descripción de hardware Verilog y sigue una metodología de diseño *top-down*. *SynthIce* está diseñado como una herramienta educativa y accesible para la exploración de conceptos de síntesis de sonido digital, permitiendo a los usuarios generar y manipular notas musicales mediante pulsadores, controlar parámetros de modulación con encoders y reproducir pistas almacenadas en memoria. El diseño incorpora módulos de generación de ondas, modulación de frecuencia y conversión digital-analógica para la salida de audio. Se detallan las etapas de especificación, modelado, integración y optimización del sistema, garantizando un alto grado de eficiencia y flexibilidad en la implementación en FPGA.

Abstract—This paper presents the design and implementation of *SynthIce*, an FPGA-based digital synthesizer for real-time sound synthesis and modulation. The system is developed on the *BlackIce40* FPGA using the Verilog hardware description language and follows a *top-down* design methodology. *SynthIce* is designed as an educational and accessible tool for exploring digital sound synthesis concepts, allowing users to generate and manipulate musical notes via push buttons, control modulation parameters with rotary encoders, and playback stored tracks from memory. The design includes waveform generation, frequency modulation, and digital-to-analog conversion modules for audio output. The document details the specification, modeling, integration, and optimization stages, ensuring high efficiency and flexibility in the FPGA implementation.

Palabras Clave—FPGA, Verilog, síntesis de sonido digital, modulación de audio, conversión digital-analógica, *Direct Digital Synthesis* (DDS), *SynthIce*.

I. INTRODUCCIÓN

EL avance de la electrónica digital ha facilitado el desarrollo de sistemas compactos y eficientes para la producción de sonido. Los sintetizadores y controladores MIDI son herramientas fundamentales para músicos y productores, permitiendo la generación y modificación de sonidos digitales a partir de señales de control. Sin embargo, muchos de estos dispositivos cuentan con arquitecturas complejas que pueden ser innecesarias para usuarios principiantes o aplicaciones específicas.

El proyecto *SynthIce: FPGA MIDI Lite para Síntesis de Sonido Digital* tuvo como objetivo el diseño de un sintetizador digital y controlador de notas musicales basado en la FPGA *BlackIce40*, utilizando Verilog. Proporcionando así una

solución accesible y eficiente que cuenta con el uso y creación de módulos de procesamiento paralelo y manipulación de señales mediante Verilog y la FPGA.

Además, el uso de FPGAs, permite implementar diseños complejos como el DDS, superando las limitaciones y costos de los circuitos analógicos. Gracias a su procesamiento paralelo, la FPGA mejora la eficiencia y reduce la latencia en la conversión de fase a amplitud [1]. Además, las FPGAs facilitan la producción simultánea de múltiples sonidos, mejorando la libertad tímbrica en sintetizadores [2].

Esta flexibilidad se refleja en implementaciones precisas y de alto rendimiento. Bergeron y Willson [3] presentaron un sintetizador de frecuencia directa basado en FPGA con alta precisión, mientras que Omran et al. [4] destacaron el uso de técnicas como el sobre-muestreo y la reducción de ruido, elementos igualmente considerados en este proyecto.

De este modo, el presente diseño adoptó un enfoque *top-down*, comenzando desde un alto nivel de abstracción y descomponiendo el sistema en módulos jerárquicos que se implementaron de manera eficiente en la FPGA [5]. Abordando las siguientes etapas fundamentales:

- **Especificación de Funcionalidad:** Se identificaron las características esenciales, requerimientos funcionales, no funcionales, entre otros.
- **Modelado en Verilog y Montaje:** Se describió en Verilog cada módulo del sistema (síntesis, reproducción, modulación), permitiendo su ejecución paralela en la FPGA a parte de las implementaciones físicas.
- **Pruebas y Verificación:** Se simularon y probaron los módulos en la FPGA para garantizar el cumplimiento de los requisitos de funcionalidad y rendimiento.
- **Integración y Optimización:** Se integraron los módulos y se optimizó el rendimiento, la eficiencia energética y la estabilidad del diseño.

Para realizar este proceso correctamente, se desarrollaron los siguientes componentes principales:

- Interfaz de entrada amigable para recibir señales de control y generar notas musicales.
- Sistema de control y modulación para gestionar parámetros de la síntesis en tiempo real.
- Módulo de síntesis de sonido basado en técnicas de generación de ondas digitales.

- Conversión Digital-Analógica y amplificación para la salida de audio.
- Encoder para el ajuste de parámetros de la síntesis.
- Memoria para almacenar una pista almacenada que el sistema reproduce en tiempo real.

La implementación de los módulos en Verilog permitió una descripción modular y optimizada, aprovechando las capacidades paralelas de la FPGA BlackIce40. El proyecto implementó la síntesis y modulación de notas musicales a través de módulos físicos, controlando las ondas generadas en tiempo real, sin depender de tablas de búsqueda o simulaciones externas. Además de la Generación de una pista almacenada en memoria.

II. OBJETIVO

SynthIce busca ofrecer un dispositivo compacto y educativo que permita explorar conceptos de síntesis de sonido digital y diseño de hardware en FPGA, sin la necesidad de hardware especializado costoso. El presente documento detalla el proceso de diseño de *SynthIce* paso a paso, abordando las etapas de modelado, verificación, integración y optimización, así como las decisiones tomadas durante la implementación del sintetizador digital en la FPGA.

III. ESPECIFICACIONES DE DISEÑO

El proyecto *SynthIce* consiste en el diseño y desarrollo de un sintetizador, modulador y controlador de sonido digital basado en una FPGA. Es importante resaltar que en Anexo IX-A se encuentra la teoría musical necesaria para realizar el proceso de diseño.

El sistema será alimentado con 5V, utilizando una de las opciones de alimentación que ofrece la FPGA BlackIce, asegurando compatibilidad con los módulos y componentes del circuito. Esta elección permite una integración eficiente con las entradas y salidas, manteniendo un consumo energético adecuado.

Entradas y Salidas

Las entradas del sistema serán las siguientes:

- 5 pulsadores, cada uno correspondiente a una nota musical específica de la cuarta octava (C4 a B4), incluyendo los semitonos. Estos pulsadores permitirán la selección y reproducción de una o varias notas.
- Un interruptor de botón para reproducir una pista pregrabada que corresponde a una melodía.
- Un encoder rotativo para modular el tono de las notas musicales. Este encoder controlará una frecuencia moduladora de hasta 30 Hz, que será añadida o restada a cada nota generada, permitiendo modificar su tono en tiempo real.
- Un encoder rotativo para ajustar la velocidad de reproducción de las pistas almacenadas. Este encoder permitirá controlar el BPM (pulsos por minuto) de la armonía que se está reproduciendo, con un rango de 60 a 180 BPM.

Por otro lado, las salidas del sistema serán las siguientes:

- Señal de audio combinada y modulada. La salida principal será una señal de audio que combine las notas generadas y la melodía almacenada. Esta señal será enviada a dos parlantes para su reproducción.

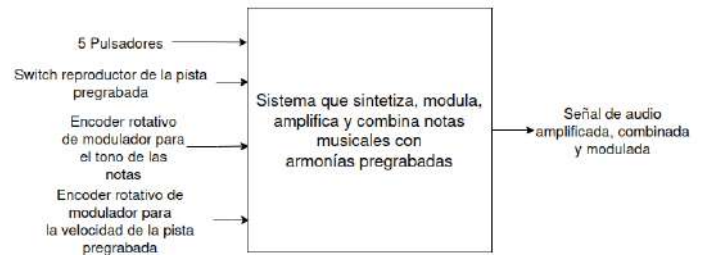


Fig. 1. Diagrama de Caja Negra de SynthIce

Requerimientos

A partir del diseño propuesto y su arquitectura, se han identificado dos categorías de requerimientos fundamentales para garantizar la funcionalidad y el rendimiento óptimo del sistema: requerimientos funcionales, que establecen las características esenciales del sintetizador, y requerimientos no funcionales, que definen criterios de eficiencia, usabilidad y escalabilidad del diseño.

Requerimientos funcionales:

- El sistema debe generar varias notas musicales de la cuarta octava del piano (entre C4 y B4) mediante un teclado interactivo basado en pulsadores. Cada nota debe producirse con una señal de onda cuadrada ajustada a la frecuencia correspondiente.
- El sistema debe almacenar y reproducir al menos una melodía pregrabada con una duración de 4 compases. La reproducción de la pista se debe controlar con un swithch de botón.
- El sistema debe contar con una etapa de modulación en tiempo real controlada por encoders, permitiendo:
 - Ajustar la frecuencia de modulación de las notas individuales dentro de un rango configurable (hasta 30 Hz), sin alterar la tonalidad ni generar distorsiones audibles.
 - Controlar la velocidad de reproducción de la pista almacenada, modificando su BPM en un rango de 60 a 180 BPM sin interrupciones ni pérdida de calidad.
- El sistema debe permitir la combinación de señales musicales, asegurando:
 - La reproducción simultánea de múltiples notas cuando se presionen varios pulsadores, permitiendo la formación de acordes.
 - La superposición de una nota generada manualmente con la reproducción de una melodía pregrabada.
- La salida del sistema debe ser una señal de audio digital combinada y modulada, compatible con un amplificador externo para su conversión a señal analógica y reproducción en altavoces.

Requerimientos no funcionales:

- El sistema debe optimizar el uso de recursos en la FPGA, asegurando una implementación eficiente en términos de memoria y lógica combinacional sin exceder la capacidad de la BlackIce40.
- Debe contar con una interfaz de usuario intuitiva, accesible para usuarios sin conocimientos avanzados en electrónica o síntesis de sonido. Los controles deben ser de fácil manipulación y respuesta inmediata.
- El diseño físico debe ser compacto y robusto, adecuado para su montaje en una PCB, asegurando portabilidad y durabilidad.
- El sistema debe ser compatible con periféricos de hardware estándar, como altavoces, displays y amplificadores, permitiendo su integración con equipos de audio convencionales.
- La alimentación debe realizarse mediante USB Micro de 5V, garantizando que la FPGA y los módulos asociados puedan operar sin necesidad de fuentes de alimentación externas adicionales.
- Se debe minimizar el consumo energético, asegurando que el sistema funcione correctamente con la alimentación USB sin superar los límites de corriente permitidos.
- El diseño debe permitir expansión y actualización modular, facilitando la incorporación de nuevas funciones como efectos de sonido o controladores adicionales sin modificar significativamente el hardware.

Dentro de este marco de ideas, para el cumplimiento de cada requerimiento funcional se decidió que el sistema estará compuesto por tres módulos principales:

- Sintetizador
- Reproductor de Pistas
- Reproductor de Audio

Los cuales poseen la siguiente interconexión:

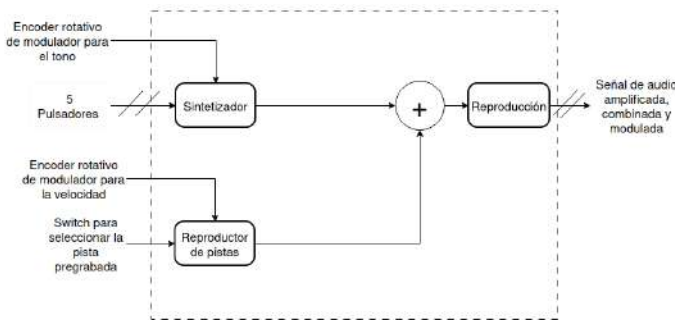


Fig. 2. Diagrama de Caja Gris de SynthIce

Este sistema permite sintetizar y modular señales de audio a partir de dos fuentes: una entrada manual mediante pulsadores y la reproducción de pistas almacenadas en memoria. La entrada manual cuenta con 5 pulsadores que activan un sintetizador de onda cuadrada con frecuencias correspondientes a notas musicales. Además, un encoder externo permite modular la frecuencia en un rango de 0 a 30 Hz.

Por otro lado, el sistema reproduce melodías almacenadas

mediante un divisor de frecuencias dinámico, con un modulador de velocidad ajustable por encoder. Ambas señales se combinan en un sumador análogo, se amplifican y procesan para generar el audio de salida. Este sintetizador digital ofrece flexibilidad en la manipulación del tono y la velocidad en tiempo real.

A. Sintetizador

El módulo de *Sintetizador* se encargará de generar las notas musicales mediante una señal cuadrada, que es la forma de onda básica utilizada para la síntesis de sonidos en este diseño. Al presionar uno de los pulsadores correspondientes a una nota (C4, C#4, D4, etc.), este módulo utilizará el reloj de la FPGA y los divisores de frecuencia para generar la frecuencia correcta para cada nota.

El diagrama general para este modulo corresponde al que se observa en la Figura 3.

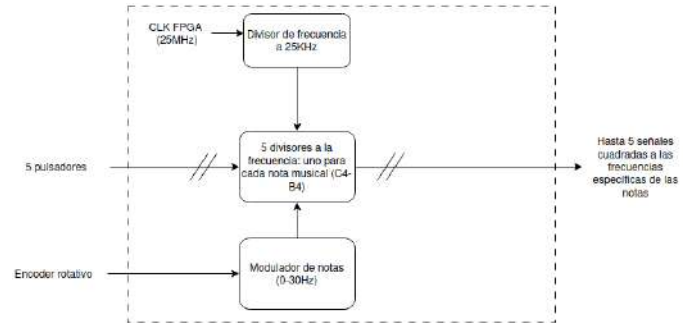


Fig. 3. Diagrama para la generación de notas.

Entradas:

- **Pulsadores:** Cada pulsador representa una nota específica. El módulo convierte la señal del pulsador en un código, el cual es mapeado a un divisor de reloj específico para generar la frecuencia correspondiente a la nota seleccionada.
- **Reloj FPGA:** La FPGA cuenta con un reloj de alta frecuencia (25 MHz). Este reloj es demasiado rápido para la generación de ondas audibles, por lo que es necesario dividirlo.

Salidas:

- **Ondas cuadradas:** Este módulo alterna la salida de la señal de acuerdo con la frecuencia determinada por el divisor de reloj, generando una onda cuadrada con ciclo de trabajo del 50%.

Proceso:

- **Divisor de reloj:** El divisor de reloj ajusta la frecuencia del reloj principal para producir la frecuencia de la nota seleccionada. Se usa un contador que cambia el estado de salida cuando alcanza un valor específico:

$$f_{salida} = \frac{f_{entrada}}{N} \quad (1)$$

Donde $f_{entrada}$ es la frecuencia del reloj original y N es el divisor de frecuencia calculado para la nota

seleccionada.

A dicha sección se le adiciona otro divisor de frecuencia diseñado para generar una onda cuadrada cuya frecuencia de salida puede ajustarse dinámicamente mediante la entrada de modulación que corresponde a uno de los encoders. La salida se encarga de modificar la señal generada de las notas para permitir variaciones en el tono, lo cual es útil en un sintetizador para efectos de vibrato o cambios sutiles en la afinación.

Este módulo es fundamental, ya que es el encargado de transformar las señales digitales que representan las notas en ondas cuadradas, procesarlas y modularlas. Los pulsadores a usar son los de la Figura 4.



Fig. 4. Teclado de membrana. Tomado de: <https://www.mactronica.com.co/teclado-membrana-de-4-canales>

B. Reproductor de Pistas

El módulo *Reproductor de Pistas* se encarga de gestionar la reproducción de las pistas almacenadas en memoria. Este módulo tiene dos tareas principales: almacena la pista pregrabada en un módulo y la reproduce mediante el switch de botón, junto con ajustar la velocidad de reproducción de la pista mediante el encoder que controla los BPM (tempo) de la reproducción.

Entradas:

- Switch de reproducción de pista: Un interruptor para reproducir la melodía, cada vez que se presione se volverá a reproducir desde el inicio.
- Encoder de velocidad de pista: Un encoder rotativo que ajustará la velocidad de reproducción de la pista, controlando el BPM de la armonía.

Salidas:

- Señales de la pista seleccionada: La señal de la pista, que contiene la secuencia de notas correspondiente.

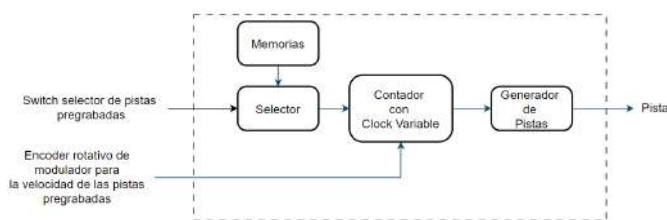


Fig. 5. Diagrama Módulo de Generación de Pistas

El *Reproductor de Pistas* está compuesto por varios bloques funcionales. En primer lugar, tiene como entrada al *Switch de reproducción de pista* con la que permite que se propague. Además, el encoder rotativo de modulación de velocidad

de pista controla la velocidad de reproducción de la pista, ajustando los BPM de la armonía, lo que modifica la velocidad a la que los acordes pregrabados (pistas) se reproducen.

El bloque de *Memoria* almacena el valor de las frecuencias de las notas de los acordes de la pista seleccionada en un módulo que se comporta como una memoria ROM de 32 bytes. El *reproductor* es responsable de extraer las frecuencias de la memoria, y el *Contador con Clock Variable* recorre la memoria con un reloj variable que se ajusta con la velocidad proporcionada por el encoder rotativo. Este contador se utiliza para generar la secuencia de las notas del acorde en función de la velocidad de reproducción configurada. Finalmente, el *divisor de frecuencias dinámico* toma los valores de las frecuencias y genera señales con una frecuencia de salida variable en tiempo real que se enviarán al sistema de reproducción de audio.

Este módulo es crucial para la integración de la pista almacenada, permitiendo que se reproduzcan a diferentes velocidades, y combinándolas con las notas generadas en el sintetizador.

C. Reproductor de Audio

El módulo *Reproductor de Audio* es responsable de convertir la señal digital combinada de las notas generadas y las pistas reproducidas en una señal analógica que pueda ser compatible con los parlantes. Este módulo utilizará un amplificador para convertir la señal digital a analógica y enviarla a los parlantes.

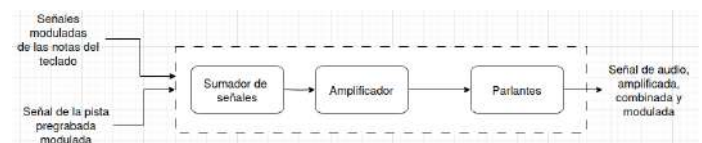


Fig. 6. Diagrama de reproducción

Entradas:

- señales digitales provenientes del Sintetizador y reproductor de Pistas.

Salidas:

- Señales de audio analógicas: La señal convertida a analógica que será enviada a los parlantes para su reproducción.

Este módulo es esencial para la salida final del sistema, ya que convierte las señales digitales a una forma que puede ser escuchada por el usuario. Para lograrlo, combina las señales de entrada de manera análoga para luego amplificarlas mediante un PAM8403, un amplificador de audio de clase D que proporciona hasta 3W por canal con una alimentación de 5V.



Fig. 7. amplificador propuesto para el módulo de reproducción. Tomado de <https://www.zamux.co/modulo-amplificador-pam-8403-con-potenciometro>

Este amplificador es eficiente en términos de consumo energético y permite una reproducción clara del sonido sintetizado, minimizando la distorsión y el ruido. La señal amplificada es enviada a dos parlantes de 8W, asegurando que los efectos de modulación y síntesis sean perceptibles en la salida de audio.



Fig. 8. parlantes propuestos para el módulo de reproducción. Tomado de <https://www.mactronica.com.co/parlante-8-ohm-05w-5-cm-mini-altavoz>

Se busca que este sistema modular permita la creación de un sintetizador digital funcional, con capacidad para modificar el tono de las notas, controlar la velocidad de las armonías y reproducir dos pistas almacenadas. Durante el desarrollo, se priorizará que el sistema sea fácil de controlar, eficiente en el uso de recursos y de bajo consumo energético, garantizando así un diseño optimizado tanto en funcionalidad como en rendimiento.

IV. DISEÑO

A continuación, se presentará el diseño de cada módulo del sistema, detallando su desarrollo paso a paso, la programación en Verilog, las pruebas realizadas y la integración en la FPGA.

Para la estructura general y el funcionamiento articulado de la FPGA, tenemos el archivo `top.v` que podemos observar en la sección Anexos IX-B1. El módulo incluye los siguientes archivos para la implementación:

```
'include "../sintetizador.v"
'include "../reproductorPista.v"
```

Estos módulos están destinados a gestionar la generación de señales de audio y la reproducción de pistas predefinidas. Su uso permite una mayor escalabilidad y mantenibilidad del código.

Descripción de señales

El módulo `top` cuenta con una serie de señales de entrada y salida que permiten la interacción con el hardware.

Entradas del sintetizador

- **Notas musicales:** Representadas por señales como `Do`, `DoS`, `Re`, `ReS`, `Mi`, `Fa`, `FaS`, `Sol`, `SolS`, `La`, `LaS`, `Si`, permiten la activación de diferentes tonos en el sintetizador.
- **Encoder de control:** Señales `canalA` y `canalB` utilizadas para modificar parámetros en tiempo real.
- **Señal de reset:** La entrada `sw` se encarga de reiniciar los ajustes del sintetizador.

Entradas del Reproductor de Pistas

- **Control de pista:** Señales `PcanalA` y `PcanalB` que permiten la selección de diferentes pistas.
- **Reset del encoder:** La señal `Psw` reinicia el control de pistas.
- **Botón de reproducción:** Activación o desactivación de la pista a través de la señal `boton`.

Salidas del Sistema

- **Generación de onda:** Cada nota produce una señal de salida específica (`ondaDo`, `ondaDoS`, `ondaRe`, ... `ondaSi`).
- **Salida de la pista:** La reproducción de audio se emite a través de la señal `pista`.

Flujo de Datos y Funcionamiento El sistema funciona de la siguiente manera:

- 1) Cuando se presiona una tecla (entrada de nota), el sintetizador genera una onda correspondiente.
- 2) Si se usa el encoder del sintetizador, se pueden modificar características del sonido en tiempo real, correspondientes al efecto de *vibrato*.
- 3) Si se usa el encoder del reproductor de pistas permite modificar en tiempo real la velocidad de la melodía pre guardada.

En síntesis se obtiene lo que se observa en la Figura 9.

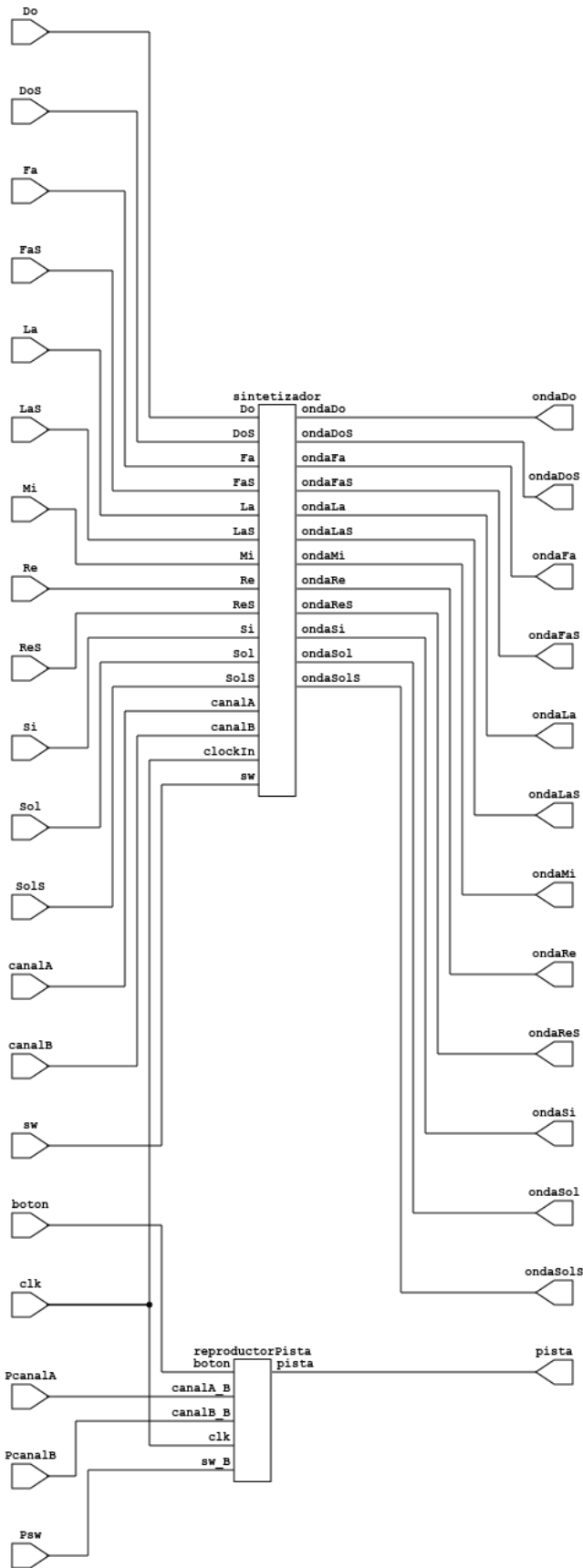


Fig. 9. Diagrama top.v

A. Módulo Sintetizador

Es el núcleo del sistema, combinando señales del pulsador y el encoder para sintetizar el sonido. Implementa una lógica de control que selecciona la frecuencia adecuada para cada nota. Integra 5 generadores de notas, un divisor de frecuencia base y un módulo de un encoder. En la sección de Anexos IX-B2 se puede observar el código completo del módulo en Verilog.

Componentes Clave:

- Divisor de 25 MHz a 25 kHz (frecDivMain.v).
- Contador de 5 bits (encoderMod.v).
- 5 instancias de pulsadorNote.v (notas musicales).

Cada pulsador activa una nota cuya frecuencia es modulada por el contador del encoder.

Con las entradas bien definidas, se delimitó el comportamiento del módulo sintetizador como se puede observar en la Figura 10:

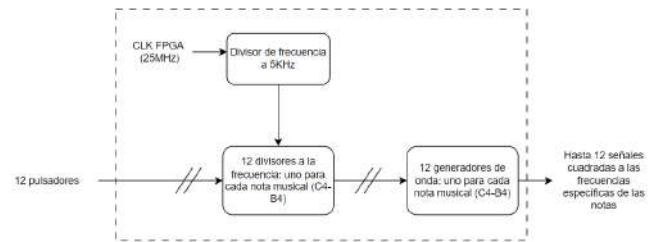


Fig. 10. Módulo Sintetizador delimitado.

En síntesis se obtiene lo que se observa en la Figura 9.

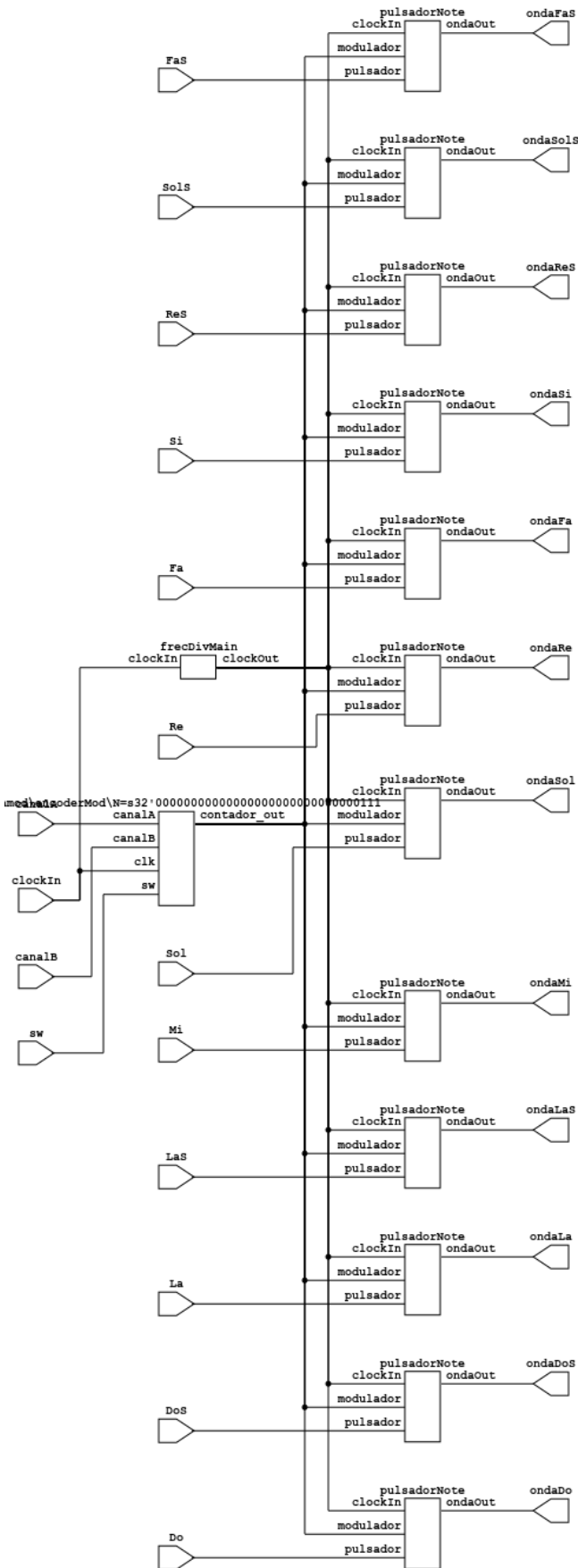


Fig. 11. Diagrama sintetizador.v

1) *Manejo de Entradas:* En este módulo, el propósito general es activar una nota musical solo cuando se presiona un pulsador. Se utilizarán tres teclados de membrana matricial 1x4 como entrada para seleccionar las notas musicales correspondientes. Cada tecla del teclado se conecta a la FPGA mediante una configuración de líneas de entrada/salida, permitiendo la lectura de las señales generadas al presionar las teclas como se puede observar en la Figura 12.

El código de Verilog que explica el comportamiento está en la sección de Anexos IX-B4.

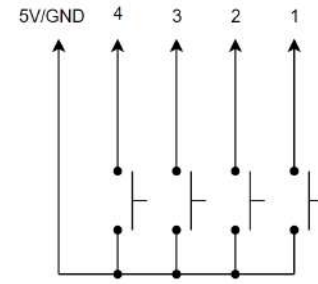


Fig. 12. Esquema de conexión para teclado de membrana matricial 1x4.

Observando el esquema, es claro que cada una de las membranas del teclado se comporta como un pulsador normalmente abierto. Por lo cual es posible conectar la quinta entrada a una fuente, y en cada salida colocar una resistencia para después dirigirse a la FPGA. Sin embargo, para simplificar los componentes físicos y los futuros diseños de PCB, es también posible realizar la siguiente conexión:

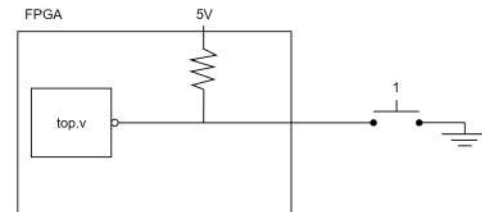


Fig. 13. Esquema con Resistencia de pull up

Como se muestra en la Figura 13, se utiliza una resistencia de pull-up en cada línea de entrada del teclado. Esto asegura que, cuando no se presiona ninguna tecla, la señal de entrada se mantenga en un estado alto (5V) debido a la resistencia que se conecta entre el pin de entrada y la fuente de alimentación. Al presionar una tecla, el pin de entrada se conecta a tierra (0V), lo que genera una señal baja que puede ser detectada por la FPGA. Sin embargo, para manejar una lógica positiva se niega la entrada. De modo que, el 1 lógico se presentará al pulsar el teclado y el 0 lógico cuando no se pulsa y sigue normalmente abierto.

La utilización de la resistencia de pull-up interna simplifica el diseño, ya que no es necesario agregar resistencias externas, reduciendo la cantidad de componentes físicos y facilitando el diseño de la PCB. Para configurar la resistencia de pull-up en el archivo de restricciones (‘.xdc’), se asigna la propiedad

‘PULLUP’ a cada pin de entrada correspondiente al teclado de membrana, de la siguiente manera:

```
set_property PULLUP true [get_ports] DO
```

Con base en lo anterior se pueden encontrar las siguientes características.

Parámetros:

- **FrecIn:** Frecuencia de entrada (ej: 25 kHz).
- **FrecOut:** Frecuencia de la nota (ej: 440 Hz para La4).

Entradas/Salidas:

- Entradas: `clockIn`, `pulsador`, `modulador` (5 bits).
- Salida: `ondaOut`

Funcionamiento: Instancia `frecDiv.v` para generar la frecuencia. Habilita la salida solo si `pulsador` está activo.

Dependencias: Requiere `frecDiv.v` para la generación de frecuencia.

En síntesis se obtiene lo que se observa en la Figura 14.

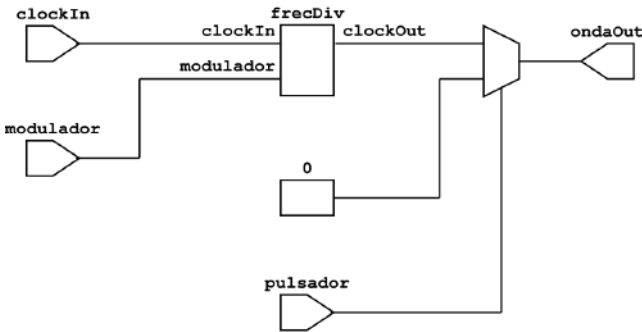


Fig. 14. Diagrama `pulsadorNote.v`

2) *Divisores de frecuencias:* Las notas se manejan en paralelo. Lo cual permite que el usuario pueda tocar tantas notas como quiera simultáneamente. El objetivo general es generar una señal de salida con frecuencia configurable desde una señal de entrada.

Para esto tenemos en este modulo, dos divisores de frecuencia principales:

frecDivMain

Este modulo lleva el clock de entrada de 25 MHz, que es el de la FPGA a un clock de salida de 25 kHz, ya que es una frecuencia demasiado alta en comparación a las frecuencias que se quieren obtener y ademas se utilizara para otro divisor de frecuencia explicado a continuación, así como al divisor de frecuencia de la reproducción de melodías. El código en Verilog se lo puede ver detenidamente en la sección de Anexos IX-B5.

Parámetros:

- **FrecIn:** Frecuencia de entrada (ej: 25 MHz).
- **FrecOut:** Frecuencia deseada de salida (ej: 25 kHz).

Entradas/Salidas:

- Entrada: `clockIn` (reloj de la FPGA).
- Salida: `clockOut` (señal dividida).

Funcionamiento: Calcula el divisor $N = \frac{FrecIn}{FrecOut}$. Invierte `clockOut` cada $N/2$ ciclos para generar una onda cuadrada.

En síntesis se obtiene lo que se observa en la Figura 15.

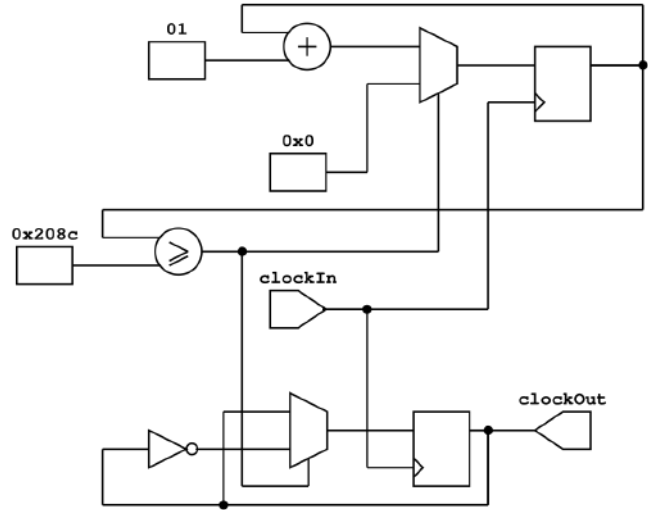


Fig. 15. Diagrama `frecDivMain.v`

frecDiv

El objetivo principal es generar una señal modulable mediante un divisor de frecuencia ajustable. El código en Verilog se lo puede ver detenidamente en la sección de Anexos IX-B6.

Parámetros:

- **FrecIn:** Frecuencia de entrada (ej: 25 kHz).
- **FrecOut:** Frecuencia deseada de salida (ej: 440 Hz).

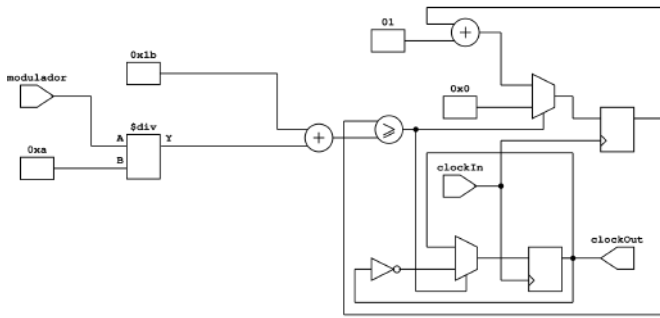
Entradas/Salidas:

- Entrada: `clockIn`, `modulador` (7 bits).
- Salida: `clockOut`.

Funcionamiento:

Calcula $N = \frac{FrecIn}{2 \times FrecOut}$. Ajusta dinámicamente el divisor usando `modulador/10`.

En síntesis se obtiene lo que se observa en la Figura 16.

Fig. 16. Diagrama `frecDiv.v`

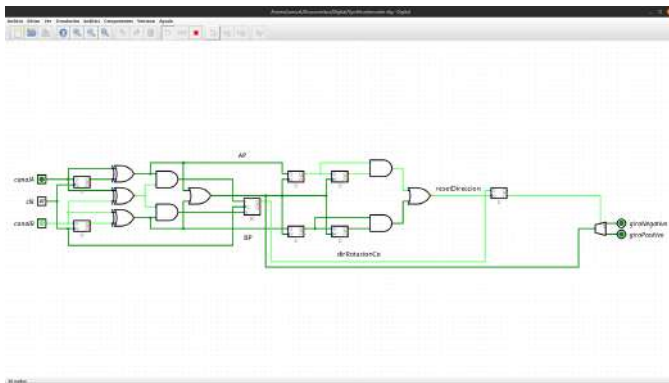
3) *Modulador de notas*: Para el encoder rotativo se necesita dos módulos principales, de manera secuencial, en el cual se define primero, si se va a sumar o restar y el contador, que define que tanto va a aumentar o disminuir la frecuencia.

Para esto tenemos un concepto básico musical llamado *Vibrato*, en el que profundizaremos en la sección de Anexos IX-A5.

encoder.v

El código en Verilog se puede encontrar en la sección de Anexos IX-B8. Se necesita decodificar las señales de un encoder rotativo para determinar la dirección de giro (`giroPositivo` o `giroNegativo`).

Para este diseño, se usó inicialmente el software *Digital*, que resultó en la configuración que se observa en la Figura 17.

Fig. 17. Encoder en *Digital*.

Algunos de los módulos claves son:

- `divFrec`: Divisor de frecuencia de 50 MHz a 500 Hz.
- `DIG_D_FF_1bit`: Flip-flop D con salida Q y \bar{Q} para sincronizar `canalA` y `canalB` con el reloj dividido.
- `DIG_JK_FF`: Flip-flop JK para lógica de detección de dirección.
- `Demux1`: Demultiplexor 1:2 para asignar la salida según la dirección.

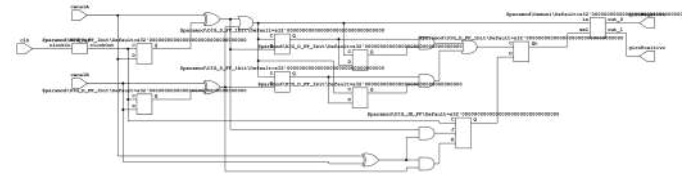
Entradas/Salidas:

- Entradas: `clk` (25 MHz), `canalA`, `canalB` (señales del encoder).
- Salidas: `giroPositivo`, `giroNegativo` (indicadores de dirección).

Integración con el Sistema:

Proporciona las señales `giroPositivo` y `giroNegativo` a `encoderMod.v` para modular el contador de 5 bits.

En síntesis se obtiene lo que se observa en la Figura 18.

Fig. 18. Diagrama `encoder.v`

encoderMod.v

El objetivo es Utilizar las señales para modular un contador de 5 bits (0–31), que ajusta parámetros en tiempo real. El código en Verilog se puede encontrar en la sección de Anexos IX-B9.

Parámetros:

- $N = 5$: Tamaño del contador (0–31).
- `MAX_COUNT = 31`: Límite superior.

Entradas/Salidas:

- Entradas: `clk`, `canalA`, `canalB`, `sw`.
- Salida: `contador_out` (5 bits).

Funcionamiento: Detecta dirección del encoder mediante flancos de subida. Incrementa/decrementa el contador dentro de 0–31.

Dependencias: Requiere `encoder.v` para decodificar señales del encoder.

En el sintetizador, `contador_out` se conecta al puerto modulador de `pulsadorNote.v`, permitiendo ajustar dinámicamente la frecuencia de las notas según el giro del encoder.

En síntesis se obtiene lo que se observa en la Figura 19.

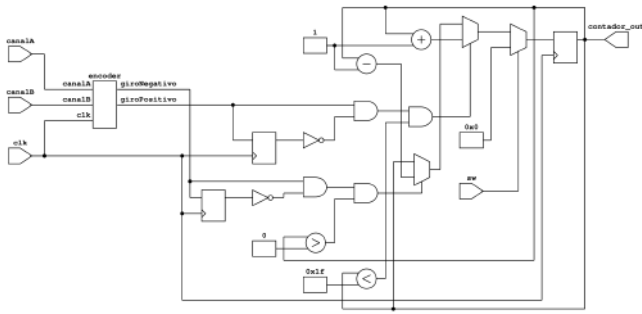


Fig. 19. Diagrama encoderMod.v

B. Módulo generador de pistas

El módulo del generador de pistas gestiona la preproducción de pistas que serán armonías mediante la manipulación de señales digitales en la FPGA.

Esta sección se compone de los siguientes módulos:

reproductorPista: Es el módulo principal del reproductor de melodía que controla la lectura de la ROM y la generación de la señal.

Este módulo en general, recibe un clk de alta frecuencia y lo divide para obtener diferentes relojes de control. Posterior a ello, usa un contador y la ROM para obtener la frecuencia de la nota que debe sonar para luego enviar la frecuencia obtenida al módulo divFrecMain para generar la onda correspondiente. Adicionalmente se modula la frecuencia usando encoderModB con el que se controla la velocidad de reproducción y se envía la señal final al módulo de reproducción.

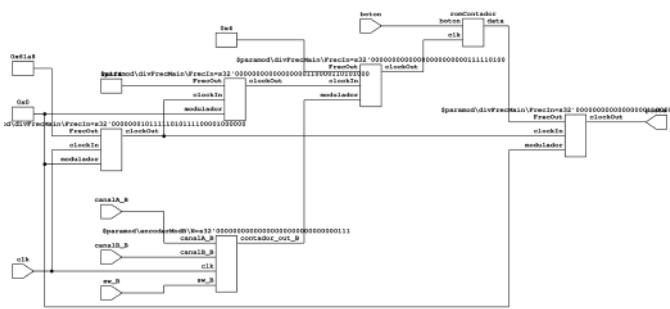


Fig. 20. Diagrama reproductor de pista

romContador: La ROM (romContador) es el módulo que se comporta como una memoria ROM que almacena una secuencia de frecuencias correspondientes a las notas de una pista musical. En ella, se hace uso de un contador que recorre las direcciones de la ROM para reproducir la secuencia de notas.

Cuando el botón de activación está presionado, el contador avanza y la ROM envía la frecuencia de la nota actual a la salida.

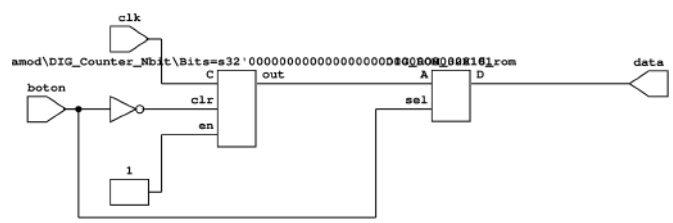


Fig. 21. Diagrama romContador.v

Dentro de este se implementa otro módulo, **DIGCounterNbit** es un contador de N bits que se utiliza para recorrer las direcciones de la ROM. Su función principal es incrementar un valor (el contador) cada vez que recibe un flanco de subida en su señal de reloj (C), lo que permite avanzar en la secuencia de notas almacenada en la ROM.

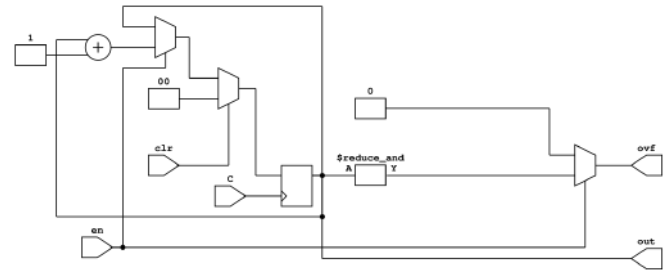


Fig. 22. Diagrama DIGCounterNbit

divFrecMain: En primer lugar, este módulo ajusta la velocidad de reproducción de la pista. Toma una señal del reloj de 25 MHz de la FPGA y genera una señal de salida con una frecuencia más baja (2 Hz - 5Hz). De igual manera toma la frecuencia de la nota actual, que es proporcionada por la ROM, y la utiliza para calcular el divisor necesario para generar una señal de salida con esa frecuencia. Corresponde al divisor para leer la melodía pregrabada, que permite definir la frecuencia de salida (FrecOut) de manera dinámica y ajustarla en tiempo real. En lugar de utilizar valores fijos, este módulo recalcula continuamente el divisor en función de la frecuencia deseada cada vez que se instancia, lo que lo hace ideal para generar múltiples notas musicales con diferentes frecuencias sin necesidad de múltiples divisores individuales.

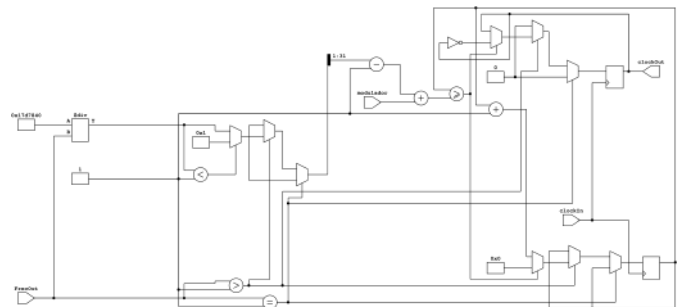


Fig. 23. Diagrama divFrecMain.v

Cada vez que se instancia el módulo, debe recalcular el divisor necesario para generar la frecuencia de salida deseada. Esto permite que el generador de pistas sea flexible y pueda manejar diferentes frecuencias de notas sin necesidad de hardware adicional.

encoderModB: Este módulo permite ajustar la velocidad de reproducción de la pista controlada por su entrada de un encoder. Detecta los giros del encoder y actualiza un contador que se utiliza para modular la frecuencia de la pista. En este módulo se implementa una FSM que tiene 4 estados, correspondientes a las combinaciones de las señales A y B en el encoder para variar la velocidad de reproducción de la melodía. El sistema comienza en el estado 00 (reposo o inicio), desde donde puede pasar a 01 si A=0, B=1 (posible giro negativo) o a 10 si A=1, B=0 (posible giro positivo). En el estado 01, si A=1, B=1, el sistema avanza al estado 11, confirmando un giro negativo; en caso contrario, si A=0, B=0, regresa a 00, indicando ruido o falta de movimiento. De manera similar, en el estado 10, si A=1, B=1, se confirma un giro positivo, mientras que si A=0, B=0, vuelve a 00. Finalmente, el estado 11 actúa como una transición antes de volver al inicio: si A=0, B=1, regresa a 01 (continuación del giro negativo); si A=1, B=0, cambia a 10 (continuación del giro positivo); y si A=0, B=0, retorna a 00, indicando el fin del movimiento.

C. Reproductor de audio

El módulo de reproducción de audio tiene la tarea de convertir las señales digitales generadas por la FPGA en una señal analógica para ser amplificada y reproducida por parlantes. Este proceso se realiza mediante una red de resistencias sumadoras y el amplificador PAM8403.

1) **Generación de Señales Digitales en la FPGA:** Cada una de las **13 señales digitales** generadas por la FPGA corresponde a una onda cuadrada:

- **5 señales:** Cada una asociada a un botón que representa una nota musical.
- **1 señal:** Corresponde a la melodía almacenada en la ROM.

Estas señales son trenes de pulsos que oscilan entre 0V y 5V con una frecuencia determinada por la nota musical.

2) **Conversión Digital a Analógica mediante Red de Resistencias:** Para convertir estas señales digitales en una señal analógica combinada, se emplea una **red de resistencias en paralelo**. Cada señal pasa por una resistencia antes de llegar a un nodo común.

Principio de Funcionamiento

Cada señal digital se suma en el nodo común y genera un voltaje analógico proporcional al número de señales activas. Matemáticamente, la salida V_{out} se expresa como:

$$V_{out} = \frac{1}{13} \sum_{i=1}^{13} V_i \quad (2)$$

Donde V_i es el voltaje de cada una de las 13 señales digitales.

Esquema del Circuito

El esquema del sumador resistivo

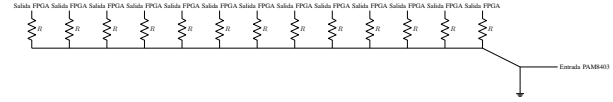


Fig. 24. Circuito sumador de resistencias conectando la FPGA al amplificador PAM8403.

Cada resistencia tiene un valor típico entre 100Ω, asegurando una conversión adecuada sin sobrecargar la FPGA.

3) **Amplificación con PAM8403:** El amplificador **PAM8403** se encarga de amplificar la señal analógica proveniente de la red de resistencias para que pueda ser reproducida por los parlantes.

Conexión del PAM8403

El amplificador tiene dos canales de entrada:

- **Canal Izquierdo (L):** Recibe la señal combinada de las 5 notas musicales.
- **Canal Derecho (R):** Recibe la señal de la melodía almacenada en ROM.

Diagrama de Conexión

Se representa como la Figura 25:

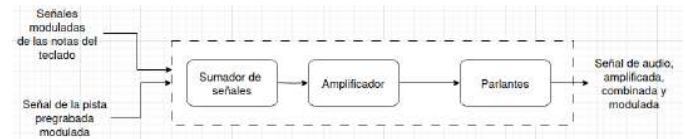


Fig. 25. Esquema para la amplificación

4) Funcionamiento Completo del Sistema:

- 1) **La FPGA genera señales digitales** de onda cuadrada para las notas y la melodía.
- 2) **Las señales digitales pasan por la red de resistencias**, convirtiéndose en una señal analógica combinada.
- 3) **El amplificador PAM8403 recibe y amplifica la señal**, distribuyéndola a los parlantes.
- 4) **Los parlantes reproducen el sonido:** el izquierdo para las notas en tiempo real y el derecho para la melodía almacenada.

V. MONTAJE Y PRUEBAS

En primera instancia se realizó el esquemático en el que se basaron todos los diseños, en la Figura 26 lo observamos hecho en *Kicad*.

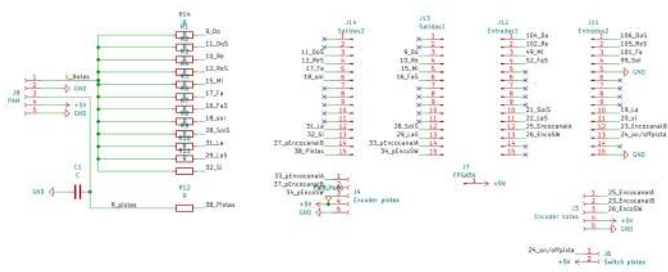


Fig. 26. Esquemático proyecto

Después se pudo realizar las pruebas correspondientes a los códigos y simulaciones, se pudo hacer las pruebas en una protoboard, tal como lo vemos en la Figura 27.

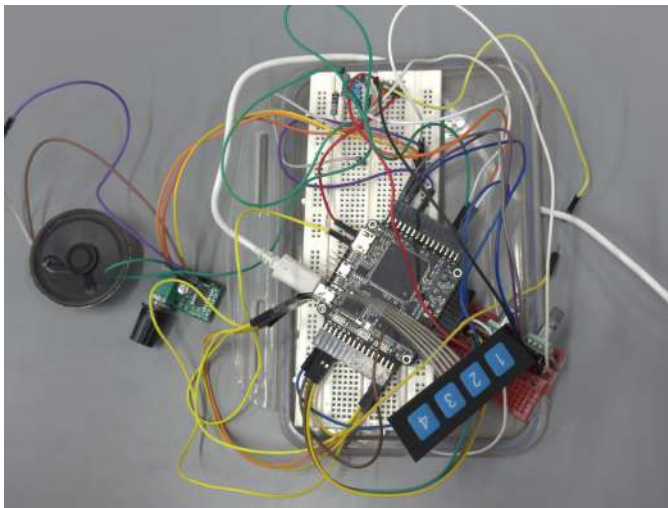


Fig. 27. Montaje en protoboard

En la etapa de montaje del proyecto, inicialmente se diseñó una PCB para integrar y organizar los componentes de manera compacta y eficiente como lo observamos en la Figura 28.

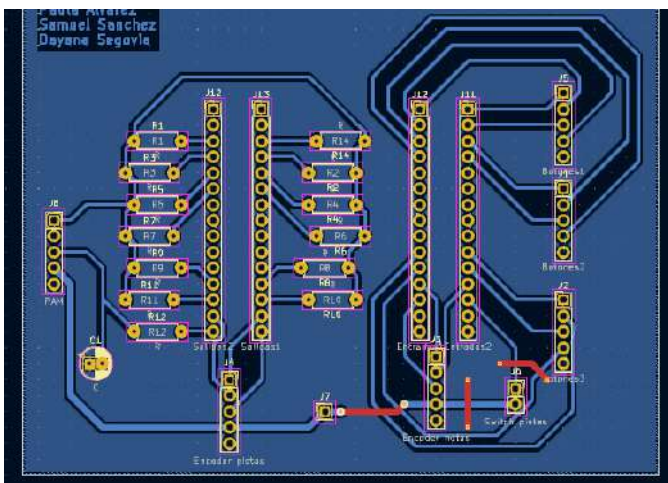
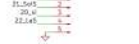


Fig. 28. Diseño para impresión de PCB

Sin embargo, debido a inconvenientes en la impresión de la tarjeta, nos vimos en la necesidad de recurrir a una baquela

universal como alternativa. Esto implicó un ensamblaje manual más detallado, asegurando la correcta disposición y conexión de los módulos mediante soldadura punto a punto. A pesar del cambio en la metodología, se garantizó la funcionalidad del sistema, manteniendo la distribución de señales y la correcta integración de los componentes del sintetizador.



Posterior a la impresión y correspondiente soldadura de los elementos necesarios obtuvimos lo que se puede observar en la Figura 29.

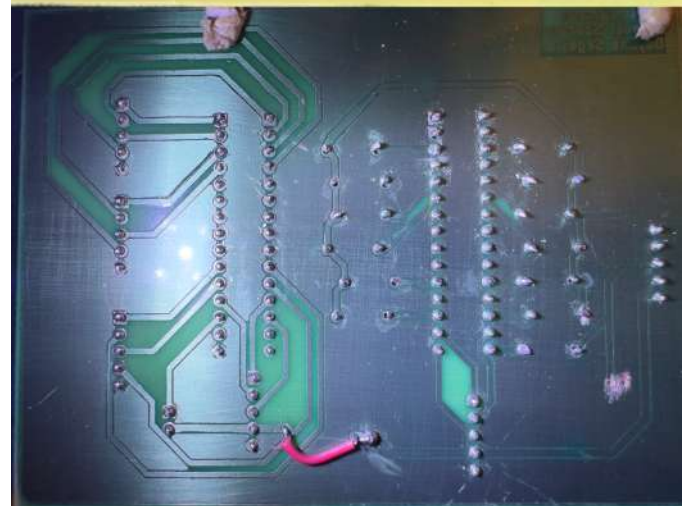
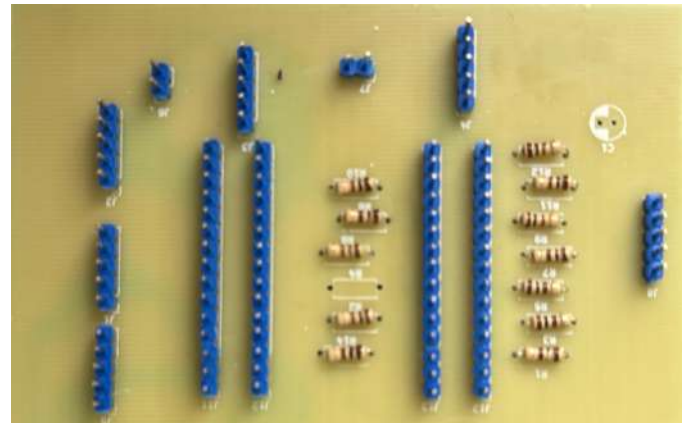


Fig. 29. Impresión PCB

Como fue mencionado hubo errores de impresión y soldadura, por lo que en la Figura 30 podemos ver el resultado en la baquela universal.

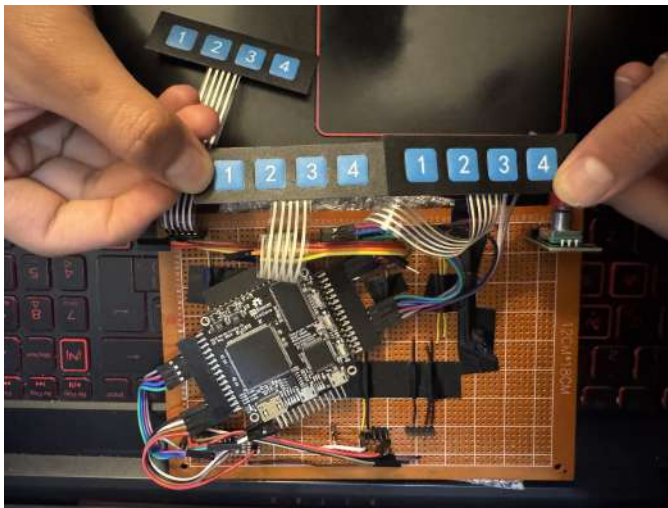


Fig. 30. Montaje baquela Universal

Para el diseño de la carcasa 3D utilizamos el software *Inventor*, el resultado del modelo fue el observado en la Figura 31.

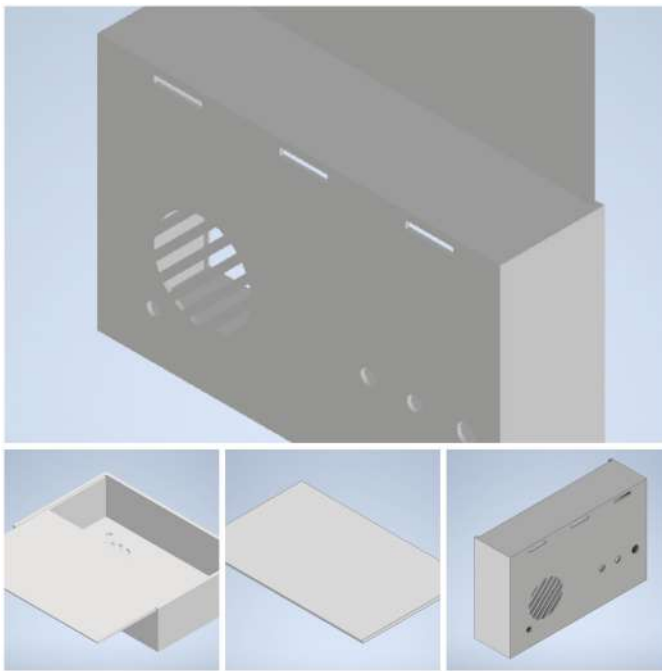


Fig. 31. Diseño de carcasa

El resultado final del proyecto se puede ver en la Figura ??.

VI. DESCRIPCIÓN Y COMPARACIÓN DE LA SOLUCIÓN

En comparación con los resultados esperados, la implementación del sintetizador en FPGA se llevó a cabo de manera exitosa, cumpliendo con la totalidad del calendario establecido. Todas las funcionalidades principales fueron alcanzadas conforme a lo planificado, con la excepción de algunos objetivos no funcionales que se plantearon al inicio del proyecto.

En primera instancia se omitieron los indicadores led, esta desviación se debió a que una de las regletas de la FPGA presentó fallos operativos, mientras que la otra disponible estaba destinada al manejo de señales analógicas, lo que imposibilitó su uso para este propósito.

Además el objetivo era generar varias notas por lo que realizaron varias pruebas de hasta cuántas notas podía generar la FPGA dentro de sus limitaciones; en los resultados obtenidos fue posible generar las 5 primeras notas; a continuación se enumeran las restricciones que impone la FPGA para que no se puedan generar más notas.

• Consumo de Corriente y Limitación de Pines de la FPGA

Las FPGAs tienen una capacidad limitada de corriente que pueden suministrar a sus pines de entrada/salida (GPIO). Cada membrana consume una pequeña cantidad de corriente cuando está activa. Si la suma de estas corrientes supera la capacidad de los reguladores internos de la FPGA o la corriente máxima permitida por los GPIO, pueden ocurrir fallos en la detección de señales o incluso daños en la FPGA.

La FPGA BlackIce40 tiene restricciones de corriente, donde cada GPIO puede suministrar aproximadamente **8 mA** y el consumo total del banco de pines no debe superar **50 mA**. Si se excede este límite al conectar múltiples teclados de membrana, la FPGA puede no reconocer correctamente las entradas o entrar en un estado errático.

• Resistencia de Pull-Up Insuficiente o Interacción entre Membranas

Los teclados de membrana funcionan con un circuito de **resistencias pull-up** para mantener la suma analógica que se hace hacia el PAM. Si la resistencia es demasiado grande ($>20k\Omega$), el **tiempo de subida de la señal** (rise time) aumenta y la FPGA puede no detectar correctamente los cambios de estado. Por otro lado, si la resistencia es demasiado baja ($<4.7k\Omega$), el consumo de corriente aumenta y puede contribuir a la saturación del sistema. Además, si se usan **resistencias pull-up compartidas entre varias membranas**, se pueden generar **fugas de corriente** entre líneas, provocando detecciones erróneas.

• Contaminación de Señales y Ruido Eléctrico

Cuando se conectan múltiples teclados de membrana a la FPGA, pueden ocurrir problemas de **interferencia electromagnética (EMI)** y **acoplamiento de señales entre líneas adyacentes**. Este fenómeno se da cuando

los cables que transportan señales digitales inducen **corrientes parásitas** en líneas cercanas, causando pulsos fantasma o falsas detecciones.

Otro problema es la **capacitancia parásita** de los cables y pines, lo cual puede provocar retardos en las señales y fallos en la detección de teclas. Este efecto es más pronunciado si se usan **cables largos sin blindaje**, ya que pueden actuar como antenas y captar interferencias externas.

VII. CONCLUSIONES

El desarrollo de SynthIce permitió la implementación de un sintetizador digital funcional basado en FPGA, demostrando la viabilidad de la síntesis y modulación de sonido en tiempo real a través de un hardware reconfigurable. La arquitectura del sistema, compuesta por módulos de síntesis, reproducción de melodía y procesamiento de audio, permitió generar y controlar sonidos digitales con precisión. La integración de pulsadores para la ejecución de notas, encoders para la modulación y memoria ROM para la reproducción de pistas almacenadas, consolidó un sistema versátil con capacidades avanzadas de control de sonido.

A lo largo del proceso, se logró optimizar el uso de los recursos de la FPGA BlackIce40, asegurando una ejecución eficiente sin comprometer la calidad del audio generado. La implementación en Verilog permitió una estructura modular, facilitando la escalabilidad y futuras mejoras en el diseño. Sin embargo, se presentaron desafíos en la etapa de montaje, principalmente por inconvenientes con la impresión de la PCB, lo que obligó a utilizar una baquela universal para el ensamblaje; junto con las limitaciones de potencia que posee la FPGA por las que tuvimos que ajustar nuestros objetivos. Aún así la funcionalidad del sistema se mantuvo y se logró la integración efectiva de los componentes.

Este proyecto valida el potencial y menciona las limitaciones de las FPGAs para la síntesis de sonido, destacando ventajas como el procesamiento paralelo y la baja latencia en comparación con soluciones basadas en microcontroladores o software. Aún así al ser un diseño educativo, SynthIce proporciona una base sólida para la exploración de conceptos de síntesis digital y procesamiento de señales en hardware.

VIII. TRABAJOS FUTUROS

A partir de los resultados obtenidos, se identifican varias áreas de mejora y expansión para futuras versiones de SynthIce:

Implementación de Nuevas Formas de Onda Actualmente, el sintetizador genera ondas cuadradas, pero sería interesante integrar ondas sinusoidales, triangulares y diente de sierra para ampliar la gama de sonidos y mejorar la calidad tonal. Esto se puede lograr mediante técnicas de síntesis digital directa (DDS) o mediante la interpolación de tablas de ondas. El sistema actualmente reproduce una pista almacenada en ROM, pero podría ampliarse para admitir múltiples melodías o incluso una interfaz para que los usuarios carguen sus propias secuencias. El uso de memorias externas como EEPROM o tarjetas SD permitiría almacenar un repertorio

más amplio.

Implementación de una Interfaz MIDI Un siguiente paso sería agregar compatibilidad MIDI, permitiendo la conexión con otros dispositivos musicales, como controladores externos o estaciones de trabajo digitales (DAWs). Esto haría que SynthIce sea una herramienta aún más potente para músicos y productores.

Mejoras en la Salida de Audio dado a que el sistema actualmente usa un amplificador PAM8403 con una red de resistencias para la conversión digital-analógica. Se podría mejorar la calidad del sonido con DACs de mayor precisión o implementando técnicas de filtrado para suavizar las señales de salida.

Dado el inconveniente con la impresión de la PCB, un trabajo futuro clave es la revisión y optimización del diseño del circuito impreso, asegurando que pueda ser fabricado sin errores y con una distribución de componentes más eficiente.

Estas mejoras permitirían a SynthIce evolucionar hacia un sintetizador más completo, con mejor calidad de audio y mayor versatilidad para aplicaciones educativas y musicales.

REFERENCES

- [1] C. Shan, Z. Chen, H. Yuan, and W. Hu, "Design and implementation of a fpga-based direct digital synthesizer," in *2011 International Conference on Electrical and Control Engineering*, 2011, pp. 614–617.
- [2] S. Yoshida and Y. Yamaguchi, "A study of an fpga synthesizer," in *2010 International Conference on Audio, Language and Image Processing*, 2010, pp. 1205–1210.
- [3] M. Bergeron and A. N. Willson, "A 1-ghz direct digital frequency synthesizer in an fpga," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 329–332.
- [4] H. Omran, K. Sharaf, and M. Ibrahim, "An all-digital direct digital synthesizer fully implemented on fpga," in *2009 4th International Design and Test Workshop (IDT)*, 2009, pp. 1–6.
- [5] D. Harris and S. Harris, *Digital Design and Computer Architecture: RISC-V Edition*, 1st ed. Morgan Kaufmann, 2022.
- [6] Lemnismath, "¿por qué tenemos 12 notas musicales? — música y matemáticas," YouTube, 2017, consultado el 6 de febrero de 2025. [Online]. Available: <https://www.youtube.com/watch?v=P7iC-fbdKmQ>
- [7] H. Landolfi, "Análisis del sonido del piano: Armónicos y parciales," YouTube, Escuela de Tecnología Pianística de Buenos Aires, 2021, consultado el 6 de febrero de 2025. [Online]. Available: <https://www.youtube.com/watch?v=hrEOX0C8TqU>
- [8] E. Herrera, *Teoría musical y armonía moderna*, Vol. 2. Berklee College of Music, 2005, vol. 2, consultado el 6 de febrero de 2025.

IX. ANEXOS

A. Fundamentos de Teoría Musical para la Síntesis Digital

La música occidental se estructura en torno a escalas y frecuencias específicas que permiten la creación de armonía y melodía. Para el diseño del sintetizador digital basado en FPGA, es crucial comprender conceptos básicos como los armónicos, la escala cromática, y la relación entre frecuencia y nota musical.

1) *Escala Cromática y la cuarta octava*: La escala cromática está compuesta por 5 notas que se distribuyen en semitonos. La frecuencia de cada nota se obtiene mediante la fórmula:

$$f_n = 440 \times \left(\sqrt[12]{2} \right)^n \quad (3)$$

Donde f_n es la frecuencia de la nota n , y 440 Hz corresponde a la nota La4 (referencia estándar). En la cuarta octava, las notas varían desde el Do4 hasta el Si4. [6]

Nota	Frecuencia (Hz)	Entero (Hz)
Do4	$440 \times \left(\sqrt[12]{2} \right)^{-9}$	262
Do#4/Reb4	$440 \times \left(\sqrt[12]{2} \right)^{-8}$	277
Re4	$440 \times \left(\sqrt[12]{2} \right)^{-7}$	294
Re#4/Mib4	$440 \times \left(\sqrt[12]{2} \right)^{-6}$	311
Mi4	$440 \times \left(\sqrt[12]{2} \right)^{-5}$	330
Fa4	$440 \times \left(\sqrt[12]{2} \right)^{-4}$	349
Fa#4/Solb4	$440 \times \left(\sqrt[12]{2} \right)^{-3}$	370
Sol4	$440 \times \left(\sqrt[12]{2} \right)^{-2}$	392
Sol#4/Lab4	$440 \times \left(\sqrt[12]{2} \right)^{-1}$	415
La4	440	440
La#4/Sib4	$440 \times \left(\sqrt[12]{2} \right)^1$	466
Si4	$440 \times \left(\sqrt[12]{2} \right)^2$	494

La segunda octava musical abarca desde el Do2 hasta el Si2. Así bien, para pasar de octava es necesario multiplicar o dividir por dos la frecuencia, por ejemplo la frecuencia de $f_{D3} = \frac{1}{2} f_{D4}$.

TABLE I
ESCALA CROMÁTICA EN LA SEGUNDA OCTAVA MUSICAL

Nota	Frecuencia (Hz)	Entero (Hz)
Do2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-9})$	65
Do#2/Reb2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-8})$	69
Re2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-7})$	73
Re#2/Mib2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-6})$	78
Mi2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-5})$	82
Fa2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-4})$	87
Fa#2/Solb2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-3})$	93
Sol2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-2})$	98
Sol#2/Lab2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^{-1})$	104
La2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^0)$	110
La#2/Sib2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^1)$	117
Si2	$\frac{1}{4} \times (440 \times \left(\sqrt[12]{2} \right)^2)$	123

2) *Tonos y Semitonos*: Un *semitono* es la menor distancia entre dos notas en la música occidental, mientras que un *tono* está compuesto por dos semitonos. La relación de frecuencia entre dos notas separadas por un semitono es:

$$f_2 = f_1 \times \sqrt[12]{2} \quad (4)$$

Para un tono completo (dos semitonos), la relación de frecuencia es:

$$f_2 = f_1 \times \sqrt[6]{2} \quad (5)$$

3) *Armónicos y Timbre Musical*: Un sonido musical no está compuesto por una sola frecuencia, sino por una combinación de armónicos. Estos son múltiplos enteros de la frecuencia fundamental (f_0): [6]

$$f_n = n \cdot f_0 \quad (6)$$

Donde n es el número del armónico. La amplitud de estos armónicos define el *timbre* del sonido, diferenciando un piano de un violín, aunque toquen la misma nota.

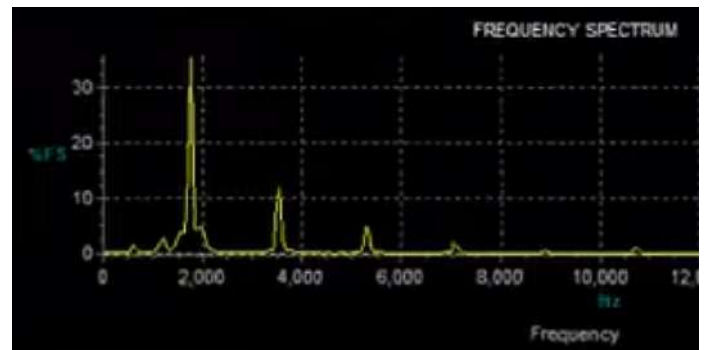


Fig. 32. Representación del espectro de armónicos de una nota en un piano. [7]

De modo que se puede estimar que, si el máximo de amplitud del armónico fundamental es 1, la amplitud del

segundo y tercer armónico de un piano tiene un valor de 0.34 y 0.16 aproximadamente.

4) *Acordes y su Relación de Frecuencia:* Un acorde es un conjunto de tres o más notas que suenan simultáneamente, creando una estructura armónica. En el caso de los acordes de *tríada*, se componen de tres notas: la *tónica*, la *tercera* y la *quinta*.

- **Acorde Mayor:** Formado por la tónica (f_0), una tercera mayor ($f_0 \times (\sqrt[12]{2})^4$) y una quinta justa ($f_0 \times (\sqrt[12]{2})^7$).
- **Acorde Menor:** Similar al mayor, pero la tercera es menor ($f_0 \times (\sqrt[12]{2})^3$).

Las relaciones de frecuencia en estos acordes producen sensaciones de consonancia y disonancia. Debido a los armónicos de cada una de las notas tocadas, si las notas tienen consonancia sus armónicos o fundamentales coinciden, mientras que en disonancia sucede lo contrario. [8]

Así un planteamiento o implementación inicial para la creación de ondas cuadradas en la FPGA en Verilog es la siguiente:

```
1 module square_wave #(parameter N = 32) (
2     input clk,
3     input reset,
4     input [N-1:0] phase_inc,
5     output reg wave
6 );
7     reg [N-1:0] phase_acc = 0;
8
9     always @(posedge clk or posedge reset) begin
10         if (reset) phase_acc <= 0;
11         else phase_acc <= phase_acc + phase_inc;
12
13         wave <= (phase_acc < {N{1'b1}} >> 1) ? 1'b0
14             : 1'b1;
15     end
16 endmodule
```

De modo que, la asignación de notas musicales se da mediante la Tabla II de frecuencias la siguiente manera.

Nota	Frecuencia (Hz)	Phase_inc (hex)
C4	261.63	0x16E99
C#4	277.18	0x18B4A
D4	293.66	0x1A6C3
...

TABLE II
TABLA DE FRECUENCIAS

5) *Vibrato:*

- Requiere LFO de baja frecuencia (0.1-30 Hz)
- Modulación de frecuencia:

$$f_{\text{mod}} = f_{\text{base}} + \Delta f \cdot \text{LFO}(t) \quad (7)$$

- Implementación:

```
1 reg [31:0] lfo_phase;
2 wire [15:0] lfo_out = lfo_phase[31:16]; // Usar
   MSBs
3
4 always @(posedge clk) begin
5     lfo_phase <= lfo_phase + (5 * (2 32)) / 12
   _000_000; // LFO 5 Hz
6 end
7
```

```
8 assign modulated_phase_inc = base_phase_inc + (
   lfo_out * depth >> 8);
9
```

B. Códigos**1) top.v:**

```

1 `include "../sintetizador.v"
2 `include "../reproductorPista.v"
3
4 module top (
5     input wire clk,                // Reloj de 25 MHz
6     de la FPGA
7
8     // Entradas del sintetizador (notas y encoder
9     para modificar sonido en vivo)
10    input wire Do, DoS, Re, ReS, Mi, Fa, FaS, Sol,
11    SolS, La, LaS, Si, // Pulsadores de notas
12    input wire canalA, canalB, // Encoder del
13    sintetizador
14    input wire sw, // Reset del
15    encoder del sintetizador
16
17    // Entradas del reproductor de pistas (encoder B
18    y bot n de inicio/parada)
19    input wire PcanalA, PcanalB, // Encoder del
20    reproductor de pistas
21    input wire Psw, // Pulsador para
22    resetear el encoder B
23    input wire boton, // Bot n de
24    activaci n/desactivaci n de la pista
25
26    // Salidas de onda del sintetizador
27    output wire ondaDo, ondaDoS, ondaRe, ondaReS,
28    ondaMi,
29    output wire ondaFa, ondaFaS, ondaSol, ondaSolS,
30    ondaLa, ondaLaS, ondaSi,
31
32    // Salida de la pista generada por el
33    reproductor
34    output wire pista
35
36 );
37
38 // Instancia del sintetizador, controlado por su
39 propio encoder
40 sintetizador mi_sintetizador (
41     .clockIn(clk),
42     .Do(Do), .DoS(DoS), .Re(Re), .ReS(ReS), .Mi(
43     Mi),
44     .Fa(Fa), .FaS(FaS), .Sol(Sol), .SolS(SolS),
45     .La(La), .LaS(LaS), .Si(Si),
46     .canalA(canalA),
47     .canalB(canalB),
48     .sw(sw),
49     .ondaDo(ondaDo), .ondaDoS(ondaDoS), .ondaRe(
50     ondaRe), .ondaReS(ondaReS), .ondaMi(
51     ondaMi),
52     .ondaFa(ondaFa), .ondaFaS(ondaFaS), .ondaSol(
53     ondaSol), .ondaSolS(ondaSolS),
54     .ondaLa(ondaLa), .ondaLaS(ondaLaS), .ondaSi(
55     ondaSi)
56 );
57
58 // Instancia del reproductor de pistas, con su
59 propio encoder
60 reproductorPista reproductorPista_inst (
61     .clk(clk), // Reloj de 25 MHz
62     de la FPGA
63     .boton(boton), // Control de
64     activaci n
65     .canalA_B(PcanalA), // Se actualiza el
66     nombre de la se al para evitar
67     conflictos
68     .canalB_B(PcanalB),
69     .sw_B(Psw),
70     .pista(pista)
71 );
72
73 endmodule

```

2) sintetizador.v:

```

1 `include "../frecDivMain.v"
2 `include "../pulsadorNote.v"
3 `include "../encoderMod.v"
4
5 module sintetizador (
6     input wire clockIn, // Reloj de 25 MHz de la
7     FPGA
8     input wire Do, DoS, Re, ReS, Mi, Fa, FaS, Sol,
9     SolS, La, LaS, Si, // Pulsadores para las
10    notas
11    input wire canalA, canalB, sw, // Entradas del
12    encoder y pulsador de reset
13    output wire ondaDo, ondaDoS, ondaRe, ondaReS,
14    ondaMi,
15    output wire ondaFa, ondaFaS, ondaSol, ondaSolS,
16    ondaLa, ondaLaS, ondaSi // Salidas de onda
17 );
18
19 wire clock25kHz; // Se al de 25 kHz generada
20 por frecDivMain
21 wire [4:0] contador; // Contador de 5 bits del
22 encoder
23
24 // Divisor de frecuencia de 25 MHz a 25 kHz
25 frecDivMain #(
26     .FrecIn(25000000),
27     .FrecOut(25000)
28 ) divisorBase (
29     .clockIn(clockIn),
30     .clockOut(clock25kHz)
31 );
32
33 // Instancia del m dulo encoderMod (contador de
34 7 bits)
35 encoderMod #(.N(7)) encoder_inst (
36     .clk(clockIn),
37     .canalA(canalA),
38     .canalB(canalB),
39     .sw(sw),
40     .contador_out(contador)
41 );
42
43 // Instancia de cada nota con modulaci n del
44 contador
45 pulsadorNote #(.FrecIn(25000), .FrecOut(262))
46 Do_inst (
47     .clockIn(clock25kHz), .pulsador(Do), .
48     ondaOut(ondaDo), .modulador(contador)
49 );
50 pulsadorNote #(.FrecIn(25000), .FrecOut(277))
51 DoS_inst (
52     .clockIn(clock25kHz), .pulsador(DoS), .
53     ondaOut(ondaDoS), .modulador(contador)
54 );
55 pulsadorNote #(.FrecIn(25000), .FrecOut(294))
56 Re_inst (
57     .clockIn(clock25kHz), .pulsador(Re), .
58     ondaOut(ondaRe), .modulador(contador)
59 );
60 pulsadorNote #(.FrecIn(25000), .FrecOut(311))
61 ReS_inst (
62     .clockIn(clock25kHz), .pulsador(ReS), .
63     ondaOut(ondaReS), .modulador(contador)
64 );
65 pulsadorNote #(.FrecIn(25000), .FrecOut(330))
66 Mi_inst (
67     .clockIn(clock25kHz), .pulsador(Mi), .
68     ondaOut(ondaMi), .modulador(contador)
69 );
70 pulsadorNote #(.FrecIn(25000), .FrecOut(349))
71 Fa_inst (
72     .clockIn(clock25kHz), .pulsador(Fa), .
73     ondaOut(ondaFa), .modulador(contador)
74 );
75 pulsadorNote #(.FrecIn(25000), .FrecOut(370))
76 FaS_inst (

```

```

54         .clockIn(clock25kHz), .pulsador(FaS), .
55             ondaOut(ondaFaS), .modulador(contador)
56     );
57     pulsadorNote #(.FrecIn(25000), .FrecOut(392))
58         Sol_inst (
59             .clockIn(clock25kHz), .pulsador(Sol), .
60             ondaOut(ondaSol), .modulador(contador)
61         );
62     pulsadorNote #(.FrecIn(25000), .FrecOut(415))
63         SolS_inst (
64             .clockIn(clock25kHz), .pulsador(SolS), .
65             ondaOut(ondaSolS), .modulador(contador)
66         );
67     pulsadorNote #(.FrecIn(25000), .FrecOut(440))
68         La_inst (
69             .clockIn(clock25kHz), .pulsador(La), .
70             ondaOut(ondaLa), .modulador(contador)
71         );
72     pulsadorNote #(.FrecIn(25000), .FrecOut(466))
73         LaS_inst (
74             .clockIn(clock25kHz), .pulsador(LaS), .
75             ondaOut(ondaLaS), .modulador(contador)
76         );
77     pulsadorNote #(.FrecIn(25000), .FrecOut(494))
78         Si_inst (
79             .clockIn(clock25kHz), .pulsador(Si), .
80             ondaOut(ondaSi), .modulador(contador)
81         );
82 endmodule

```

3) reproductorPista.v:

```

1  `include "romContador.v"
2  `include "encoderModB.v"
3
4  module reproductorPista (
5      input wire clk,           // Reloj de la FPGA (25
6          MHz)
7      input wire boton,         // Control de activaci n
8      input wire canalA_B,      // Se al del encoder B (
9          canal A)
10     input wire canalB_B,       // Se al del encoder B (
11         canal B)
12     input wire sw_B,           // Pulsador de reset del
13         encoder B
14     output wire pista          // Salida de la se al
15         musical
16 );
17
18     wire clk_2Hz;
19     wire clk_25kHz;
20     wire clk_500Hz;
21     wire [15:0] frecPista;
22     wire [6:0] contador_B; // Salida del encoderModB
23                             usada para la modulaci n
24
25     // Instancia del m dulo encoderModB (contador
26     // de 7 bits para la modulaci n)
27     encoderModB #(.N(7)) encoderB_inst (
28         .clk(clk),
29         .canalA_B(canalA_B),
30         .canalB_B(canalB_B),
31         .sw_B(sw_B),
32         .contador_out_B(contador_B) // Salida del
33         contador (modulaci n)
34     );
35
36     // Divisor de frecuencia para generar 25 kHz
37     // a partir de 25 MHz
38     divFrecMain #(
39         .FrecIn(25000000)
40     ) div25kHz (
41         .clockIn(clk),
42         .FrecOut(16'd25000),
43         .modulador(7'd0), // Sin modulaci n en
44         este divisor
45         .clockOut(clk_25kHz)
46     );
47
48     divFrecMain #(
49         .FrecIn(25000)
50     ) div500Hz (
51         .clockIn(clk_25kHz),
52         .FrecOut(16'd500),
53         .modulador(7'd0), // Sin modulaci n en
54         este divisor
55         .clockOut(clk_500Hz)
56     );
57
58     // Divisor de frecuencia para generar 2 Hz a
59     // partir de 500 Hz
60     divFrecMain #(
61         .FrecIn(500)
62     ) div2Hz (
63         .clockIn(clk_500Hz),
64         .FrecOut(16'd4),
65         .modulador(contador_B), // Sin modulaci n
66         en este divisor
67         .clockOut(clk_2Hz)
68     );
69
70     // Instancia de romContador
71     romContador rom_inst (
72         .clk(clk_2Hz),
73         .boton(boton),
74         .data(frecPista)
75     );
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

64 // Divisor de frecuencia para generar la onda
    cuadrada seg n frecPista con modulaci n
65 divFrecMain #(
66     .FrecIn(25000) // Reloj de entrada de 25
        kHz
67 ) divPista (
68     .clockIn(clk_25kHz),
69     .FrecOut(frecPista), // Usa la frecuencia
        obtenida desde ROM
70     .modulador(16'd0), // Usa el contador del
        encoder B como modulaci n
71     .clockOut(pista)
72 );
73
74 endmodule
75
76 // M dulo de Divisi n de Frecuencia (Sin cambios)
77 module divFrecMain #(
78     parameter integer FrecIn = 25000000 //
        Frecuencia de entrada en Hz
79 ) (
80     input wire clockIn, // Se al de entrada
        (reloj)
81     input wire [15:0] FrecOut, // Frecuencia de
        salida variable
82     input wire [6:0] modulador, // Modulaci n para
        ajustar la frecuencia
83     output reg clockOut // Se al de salida
        generada
84 );
85
86 reg [31:0] count = 0;
87 reg [31:0] N;
88 reg [31:0] NHalf;
89
90 always @(posedge clockIn) begin
91     if (FrecOut == 16'h1) begin
92         clockOut <= 1'b0; // Si FrecOut es 1,
            la salida es 0
93     end else if (FrecOut > 1) begin
94         N = FrecIn / FrecOut; // Calcula el
            divisor dinmicamente
95         if (N < 1) N = 1; // Evita valores
            menores a 1
96         NHalf = N >> 1; // Divisi n por
            2 m s eficiente
97
98         if (count >= ((NHalf - 1) + modulador))
            begin
99             count <= 0;
100             clockOut <= ~clockOut; // Invierte
                la se al para generar la onda
                cuadrada
101         end else begin
102             count <= count + 1;
103         end
104     end else begin
105         clockOut <= 0; // Si FrecOut es 0,
            salida en bajo
106     end
107 end
108
109 endmodule

```

```

4) pulsadorNote.v:
1 `include "../frecDiv.v"
2
3 module pulsadorNote #(
4     parameter integer FrecIn = 25000, //
        Frecuencia de entrada fija (25 kHz)
5     parameter integer FrecOut = 440 //
        Frecuencia de salida fija (por nota)
6 ) (
7     input wire clockIn, // Reloj base de la
        FPGA (25 kHz)
8     input wire pulsador, // Entrada del
        pulsador (con pull-up, activa en 0)
9     input wire [4:0] modulador, // Entrada de
        modulaci n (5 bits)
10    output wire ondaOut // Onda cuadrada
        generada cuando se presiona el pulsador
11 );
12
13 wire clockDiv; // Salida del divisor de
        frecuencia
14 wire pulso_activo; // Pulsador negado para
        lgica positiva
15
16 assign pulso_activo = ~pulsador; // Negar la
        se al porque tiene pull-up
17
18 // Instancia del divisor de frecuencia con
        modulaci n
19 frecDiv #(
20     .FrecIn(FrecIn),
21     .FrecOut(FrecOut)
22 ) divisor (
23     .clockIn(clockIn),
24     .clockOut(clockDiv),
25     .modulador(modulador) // Entrada del
        modulador
26 );
27
28 // Solo deja pasar la se al cuando se presiona
        el pulsador
29 assign ondaOut = pulso_activo ? clockDiv : 1'b0;
30
31 endmodule

```

5) *frecDivMain.v*:

```

1 module frecDivMain #(
2     parameter integer FrecIn = 25000000, //
      Frecuencia de entrada en Hz
3     parameter integer FrecOut = 1500      //
      Frecuencia de salida en Hz
4 ) (
5     input wire clockIn,    // Se al de entrada (
      reloj de la FPGA)
6     output reg clockOut    // Se al de salida con la
      frecuencia deseada
7 );
8
9     // Calcular automaticamente el divisor N
10    localparam integer N = FrecIn / FrecOut;
11    localparam integer NHalf = N / 2; // Para
      generar una onda cuadrada
12
13    integer count = 0;
14
15    always @(posedge clockIn) begin
16        if (count >= (NHalf - 1)) begin
17            count <= 0;
18            clockOut <= ~clockOut; // Invertir la
      salida cada N/2 ciclos
19        end else begin
20            count <= count + 1;
21        end
22    end
23
24 endmodule

```

6) *frecDiv.v*:

```

1 module frecDiv #(
2     parameter integer FrecIn = 25000, // Frecuencia
      de entrada fija en Hz
3     parameter integer FrecOut = 440    // Frecuencia
      de salida base en Hz
4 ) (
5     input wire clockIn,                // Se al de
      entrada (reloj de la FPGA)
6     input wire [6:0] modulador,        // Modulaci3n
      para ajustar la frecuencia
7     output reg clockOut = 0            // Se al de
      salida con la frecuencia deseada
8 );
9
10    // Divisor dinmico N y su mitad
11    localparam integer N = FrecIn / (2 * FrecOut);
      // Divisor base calculado en tiempo de
      compilaci3n
12    integer count = 0;
13
14    // Generaci3n de la onda cuadrada con
      modulaci3n
15    always @(posedge clockIn) begin
16        if (count >= ((N - 1) + (modulador / 10)))
17            begin
18                count <= 0;                // Reiniciar
      el contador
19                clockOut <= ~clockOut;    // Invertir la
      salida cada N/2 ciclos modulado
20            end else begin
21                count <= count + 1;        // Incrementar
      el contador
22            end
23        end
24    end
25 endmodule

```


7) romContador.v:

```

1  /*
2  * Generated by Digital. Don't modify this file!
3  * Any changes will be lost if this file is
4  * regenerated.
5  */
6  module DIG_Counter_Nbit
7  #(
8      parameter Bits = 2
9  )
10 (
11     output [(Bits-1):0] out,
12     output ovf,
13     input C,
14     input en,
15     input clr
16 );
17     reg [(Bits-1):0] count;
18
19     always @ (posedge C) begin
20         if (clr)
21             count <= 'h0;
22         else if (en)
23             count <= count + 1'b1;
24     end
25
26     assign out = count;
27     assign ovf = en? &count : 1'b0;
28
29     initial begin
30         count = 'h0;
31     end
32 endmodule
33
34 module DIG_ROM_32X16_rom (
35     input [4:0] A,
36     input sel,
37     output reg [15:0] D
38 );
39     reg [15:0] my_rom [0:31];
40
41     always @ (*) begin
42         if (~sel)
43             D = 16'h1; // Salida en alta impedancia
44                         // si sel es 0
45         else
46             D = my_rom[A];
47     end
48
49     initial begin
50         my_rom[0] = 16'h14A; // Mi (E) - 329.63 Hz
51         my_rom[1] = 16'h14A; // Mi (E) - 329.63 Hz
52         my_rom[2] = 16'h15D; // Fa (F) - 349.23 Hz
53         my_rom[3] = 16'h18F; // Sol (G) - 392.00
54                         // Hz
55         my_rom[4] = 16'h18F; // Sol (G) - 392.00
56                         // Hz
57         my_rom[5] = 16'h15D; // Fa (F) - 349.23 Hz
58         my_rom[6] = 16'h14A; // Mi (E) - 329.63 Hz
59         my_rom[7] = 16'h126; // Re (D) - 293.66 Hz
60         my_rom[8] = 16'h106; // Do (C) - 261.63 Hz
61         my_rom[9] = 16'h106; // Do (C) - 261.63 Hz
62         my_rom[10] = 16'h126; // Re (D) - 293.66 Hz
63         my_rom[11] = 16'h14A; // Mi (E) - 329.63 Hz
64         my_rom[12] = 16'h14A; // Mi (E) - 329.63 Hz
65         my_rom[13] = 16'h126; // Re (D) - 293.66 Hz
66         my_rom[14] = 16'h126; // Re (D) - 293.66 Hz
67         my_rom[15] = 16'h126; // Re (D) - 293.66 Hz
68         my_rom[16] = 16'h14A; // Mi (E) - 329.63 Hz
69         my_rom[17] = 16'h14A; // Mi (E) - 329.63 Hz
70         my_rom[18] = 16'h15D; // Fa (F) - 349.23 Hz
71         my_rom[19] = 16'h18F; // Sol (G) - 392.00
72                         // Hz
73         my_rom[20] = 16'h18F; // Sol (G) - 392.00
74                         // Hz
75         my_rom[21] = 16'h15D; // Fa (F) - 349.23 Hz

```

```

76         my_rom[22] = 16'h14A; // Mi (E) - 329.63 Hz
77         my_rom[23] = 16'h126; // Re (D) - 293.66 Hz
78         my_rom[24] = 16'h106; // Do (C) - 261.63 Hz
79         my_rom[25] = 16'h106; // Do (C) - 261.63 Hz
80         my_rom[26] = 16'h126; // Re (D) - 293.66 Hz
81         my_rom[27] = 16'h14A; // Mi (E) - 329.63 Hz
82         my_rom[28] = 16'h126; // Re (D) - 293.66 Hz
83         my_rom[29] = 16'h106; // Do (C) - 261.63 Hz
84         my_rom[30] = 16'h106; // Do (C) - 261.63 Hz
85         my_rom[31] = 16'h106; // Do (C) - 261.63 Hz
86
87     end
88 endmodule
89
90 module romContador (
91     input clk,
92     input boton,
93     output [15:0] data
94 );
95     wire [4:0] s0;
96     wire s1;
97     assign s1 = ~boton;
98
99     DIG_Counter_Nbit #(
100         .Bits(5)
101     ) DIG_Counter_Nbit_i0 (
102         .en(1'b1),
103         .C(clk),
104         .clr(s1),
105         .out(s0)
106     );
107
108     DIG_ROM_32X16_rom DIG_ROM_32X16_rom_i1 (
109         .A(s0),
110         .sel(boton),
111         .D(data)
112     );
113 endmodule

```

8) *encoder.v*:

```

1 // Divisor de Frecuencia (renombrado a divFrec)
2 module divFrec #(
3     parameter integer FrecIn = 50000000, //
4         Frecuencia de entrada en Hz (50 MHz)
5     parameter integer FrecOut = 500 //
6         Frecuencia de salida en Hz
7 ) (
8     input wire clockIn, // Se al de
9         entrada (reloj de la FPGA)
10    output reg clockOut = 0 // Se al de salida
11        con la frecuencia deseada
12 );
13
14 // Calcular automáticamente el divisor N
15 localparam integer N = FrecIn / FrecOut;
16 localparam integer NHalf = N / 2; // Para
17     generar una onda cuadrada
18
19 integer count = 0;
20
21 always @(posedge clockIn) begin
22     if (count >= (NHalf - 1)) begin
23         count <= 0;
24         clockOut <= ~clockOut; // Invertir
25             la salida cada N/2 ciclos
26     end else begin
27         count <= count + 1;
28     end
29 end
30 endmodule
31
32 // Flip-Flop tipo D de 1 bit
33 module DIG_D_FF_1bit
34 #(
35     parameter Default = 0
36 )
37 (
38     input D,
39     input C,
40     output Q,
41     output Qn // Cambio: antes era ~Q
42 );
43 reg state;
44
45 assign Q = state;
46 assign Qn = ~state; // Cambio: antes era ~Q
47
48 always @ (posedge C) begin
49     state <= D;
50 end
51
52 initial begin
53     state = Default;
54 end
55 endmodule
56
57 // Flip-Flop tipo JK
58 module DIG_JK_FF
59 #(
60     parameter Default = 1'b0
61 )
62 (
63     input J,
64     input C,
65     input K,
66     output Q,
67     output Qn // Cambio: antes era ~Q
68 );
69 reg state;
70
71 assign Q = state;
72 assign Qn = ~state; // Cambio: antes era ~Q
73
74 always @ (posedge C) begin
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

136 DIG_D_FF_1bit #(.Default(0)) DIG_D_FF_1bit_i5 (
137     D(s9), .C(s7), .Q(s11));
138 DIG_D_FF_1bit #(.Default(0)) DIG_D_FF_1bit_i6 (
139     D(s10), .C(s7), .Q(s12));
140
141 assign s13 = ((s9 & s11) | (s10 & s12));
142
143 // Flip-Flop D con la corrección del nombre de
144 // la señal de salida negada
145 DIG_D_FF_1bit #(.Default(0)) DIG_D_FF_1bit_i7 (
146     .D(s8),
147     .C(s13),
148     .Qn(s14) // Cambio: antes era
149     // ~Q
150 );
151
152 // Demultiplexor para determinar la dirección
153 // del giro
154 Demux1 #(.Default(0)) Demux1_i8 (
155     .sel(s14),
156     .in(s7),
157     .out_0(giroNegativo),
158     .out_1(giroPositivo)
159 );
160
161 endmodule

```

```

9) encoderMod.v:
#include "encoder.v"

module encoderMod #(
    parameter integer N = 5, // Tamaño
    // del contador (5 bits para soportar hasta
    // 31)
    parameter integer MAX_COUNT = 31 // Valor
    // máximo del contador para 5 bits
) (
    input wire clk, // Reloj de 50 MHz
    input wire canalA, // Señal del
    // encoder (canal A)
    input wire canalB, // Señal del
    // encoder (canal B)
    input wire sw, // Pulsador para
    // resetear el contador
    output wire [N-1:0] contador_out // Salida del
    // contador (5 bits)
);

// Señales internas del encoder
wire giroPositivo;
wire giroNegativo;

// Señales para detectar flancos de subida
reg giroPositivo_d = 0;
reg giroNegativo_d = 0;

// Contador de pasos de 5 bits
reg [N-1:0] contador = 0; // Contador de 5
// bits (0 a 31)

// Instancia del módulo encoder
encoder encoder_inst (
    .clk(clk),
    .canalA(canalA),
    .canalB(canalB),
    .giroPositivo(giroPositivo),
    .giroNegativo(giroNegativo)
);

// Detección de flancos de subida sincronizados
// con el reloj principal
wire pos_edge_giroPositivo = giroPositivo & ~
giroPositivo_d;
wire pos_edge_giroNegativo = giroNegativo & ~
giroNegativo_d;

always @(posedge clk) begin
    // Registrar el estado anterior para
    // detectar flancos
    giroPositivo_d <= giroPositivo;
    giroNegativo_d <= giroNegativo;

    // Lógica para actualizar el contador de
    // pasos
    if (sw) begin
        contador <= 0; //
        // Resetear el contador si el pulsador
        // está presionado
    end
    else begin
        if (pos_edge_giroPositivo && contador <
MAX_COUNT)
            contador <= contador + 1; //
            // Incrementar el contador hasta 31
        else if (pos_edge_giroNegativo &&
contador > 0)
            contador <= contador - 1; //
            // Decrementar si el contador es
            // mayor que 0
        end
    end
end

// Salida del contador
assign contador_out = contador;

```

57

58

endmodule