DESARROLLO DE UN DOWNCONVERTER PARA EL LABORATORIO DE COMUNICACIONES DIGITALES

LUIS CARLOS DÍAZ BAUTISTA

MAYRA JULIETH RANGEL SILVA

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA

2011

DESARROLLO DE UN DOWNCONVERTER PARA EL LABORATORIO DE COMUNICACIONES DIGITALES

LUIS CARLOS DÍAZ BAUTISTA

MAYRA JULIETH RANGEL SILVA

Trabajo de grado presentado como requisito para optar al título de Ingenieros Electrónicos.

Director:

PH.D HOMERO ORTEGA BOADA

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA

2011

CONTENIDO

Pág.

LISTADO DE TABLAS

Pág.

LISTADO DE FIGURAS

Pág.

LISTA DE ANEXOS

Pág

RESUMEN

TÍTULO: IMPLEMENTACIÓN DE UN DOWNCONVERTER SOBRE UNA FPGA BAJO EL CONCEPTO DE SOFTWARE DEFINED RADIO USANDO XILINX SYSTEM GENERATOR.[1]

AUTORES: Mayra Julieth Rangel Silva, Luis Carlos Díaz Bautista.[2]

PALABRAS CLAVES: Envolvente Compleja, Software Defined Radio (SDR), DownConverter, FPGA, Xilinx System Generator for DSP.

Este artículo presenta la implementación de un prototipo de un Down Converter con fines pedagógicos para practicar el significado de la Envolvente Compleja en el laboratorio de Comunicaciones Digitales, dando al estudiante la oportunidad de interactuar con un hardware reconfigurable de propósito específico para el entendimiento de este tema.

El desarrollo se hizo bajo el concepto de Software Defined Radio (SDR). Incluye la descripción de las herramientas necesarias para la óptima implementación y visualización de la Envolvente Compleja, su diagrama de constelaciones o diagrama polar usando el ChipScope Pro Analyzer, así como su espectro usando un Guide de Matlab.

Este proyecto se implementó en una FPGA de Xilinx, usando la herramienta de programación gráfica Xilinx System Generator for DSP que trabaja en conjunto con Simulink de MatLab, El dispositivo contiene un software implementado en un microcontrolador que permite la interacción con el usuario final y que reconfigura los núcleos del sistema (Filtros, mezcladores, Generadores de Señal, Interface SPI). La metodología empleada pretende mejorar la eficiencia en el aprendizaje del complejo diseño del sistema, brindando a los estudiantes una herramienta de laboratorio no existente que le permite hacer prácticas que antes no eran posibles por la falta del dispositivo. Este trabajo fue realizado en el Grupo de Investigación RadioGis de la Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander, con apoyo de la Vicerrectoría de Investigaciones y forma parte de un proyecto mayor para el desarrollo de soluciones de Radio Cognitiva basadas en SDR.

[1] . Proyecto de grado desarrollado bajo la modalidad de Investigación.

[2]. Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Ph.D Homero Ortega Boada.

ABSTRACT

TITLE: DOWNCONVERTER IMPLEMENTATION ON FPGA BASED ON SOFTWARE DEFINED RADIO CONCEPT USING XILINX SYSTEM GENERATOR.[3]

AUTHORS: Mayra Julieth Rangel Silva, Luis Carlos Díaz Bautista.[4]

KEYWORDS: Complex Envelope, Software Defined Radio (SDR), DownConverter, FPGA, Xilinx System Generator for DSP.

This paper presents a Down Converter prototype Implementation for educational purposes to practice the meaning of the Complex Envelope on Digital Communications' Laboratory that gives the students the opportunity to interact with an specific purpose, reconfigurable hardware that allows them to understand this topic.

The development was made under the concept of Software Defined Radio (SDR). It includes the tools needed for optimal implementation and visualization of the Complex Envelope, its constellation diagram or polar diagram using Chip Scope Pro Analyzer and its spectrum in a Matlab GUI.

This project was implemented on a Xilinx FPGA, using the graphical programming tool Xilinx System Generator who works with Simulink from Matlab. The device has software mounted on a microcontroller that allows interaction with final user and reconfigures all system cores (Filters, mixers, Signals Generators and SPI interface).The methodology used aims to improve the efficiency in learning the use of the complex design system, bringing the students a unique in its class laboratory equipment that allows to make practices that weren't possible before because this equipment didn't exist. This work was done in RadioGis Group, School of Electrical, Electronics and Telecommunications of the Universidad Industrial de Santander, with the support of the Research Office and it is part of a larger project to develop Cognitive Radio solutions based on SDR.

[3] . Proyecto de grado desarrollado bajo la modalidad de Investigación.

[4]. Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Ph.D Homero Ortega Boada.

Implementación de un DownConverter sobre una FPGA bajo el concepto de Software Defined Radio usando Xilinx System Generator.

Ph.D Homero Ortega Boada, Luis Carlos Díaz Bautista, Mayra Julieth Rangel Silva

Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones, Universidad Industrial de Santander

Bucaramanga, Colombia

homero.ortega@radiogis.uis.edu.co

luis.diaz@radiogis.uis.edu.co

mayra.rangel@radiogis.uis.edu.co

Abstract. *This paper presents a Down Converter for educational purposes to practice the meaning of the Complex Envelope and developed under the concept of Software Defined Radio (SDR). The development includes the tools needed for optimal implementation and visualization of the Complex Envelope, its constellation diagram or polar diagram and its spectrum in real time. This project presents an FPGA implementation, using The Xilinx System Generator tool with Matlab and Simulink. The methodology used aims to improve the efficiency in learning the use of the complex design system. This work was done in RadioGis Group, School of Electrical, Electronics and Telecommunications of the Universidad Industrial de Santander, with the support of the Research Office and it is part of a larger project to develop Cognitive Radio solutions based on SDR.*

Keywords: *Complex Envelope, Software Defined Radio (SDR), DownConverter, FPGA, System Generator.*

Resumen. Este artículo presenta un Down Converter con fines pedagógicos para practicar el significado de la Envolvente Compleja y su desarrollo bajo el concepto de Software Defined Radio (SDR). El desarrollo incluye las herramientas necesarias para la óptima implementación y visualización de la Envolvente Compleja, su diagrama de constelaciones o diagrama polar, así como su espectro en tiempo real. Este proyecto presenta una implementación en FPGA, usando la herramienta System Generator de Xilinx junto a Matlab y Simulink. La metodología empleada pretende mejorar la eficiencia en el aprendizaje del complejo diseño del sistema. Este trabajo fue realizado en el Grupo RadioGis de la Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander, con apoyo de la Vicerrectoría de Investigaciones y forma parte de un proyecto mayor para el desarrollo de soluciones de Radio Cognitiva basadas en SDR.

Palabras Clave: Envolvente Compleja, Software Defined Radio (SDR), DownConverter, FPGA, System Generator

1. INTRODUCCIÓN

La tecnología en las comunicaciones avanza a gran velocidad, proporcionando diferentes aplicaciones y servicios al hombre, avance que no ocurre a la hora de desarrollar procesos pedagógicos para su comprensión. En vista de esta situación, se hace necesaria la construcción de dispositivos que permitan la enseñanza de conceptos claves en las Comunicaciones Digitales y que de una manera didáctica y sencilla permitan afianzar dichos conceptos básicos.

Es por esto, que el presente trabajo expone un sistema Downconverter, bajo el concepto de SDR sobre FPGA. Su gran ventaja es la reconfigurabilidad del equipo y el paralelismo a la hora de desarrollar procesos.
Este trabajo presenta una forma novedosa de aprender el concepto de envolvente compleja en las comunicaciones digitales, brindando una herramienta de fácil uso y didáctica con la que los estudiantes puedan interactuar en un laboratorio. La plataforma de diseño, una FPGA, así como el uso del ChipScope Pro Analyzer, permite crear un sistema físico que otorga total interacción en cada una de las partes del proceso de aprendizaje, pues el concepto mismo de Envolvente Compleja merece prácticas buenas y reales de laboratorio.

Cabe resaltar, que los sistemas modernos de comunicaciones, gracias al SDR, pueden adaptarse a numerosos protocolos de comunicación inalámbrica. Por ejemplo, el modulador de una radiobase moderna, bien puede emitir UMTS, WiMax, WiFi, GSM con un simple cambio de configuración de software. Esto debido a que la envolvente compleja de cualquiera de ellos se genera por software. El transmisor sólo se encarga de llevar esa envolvente compleja a frecuencias altas y el receptor realiza el proceso inverso. Además, un analizador de espectros moderno, basado en la FFT, sin duda debe implementar un Down Converter ya que es la única manera de poder obtener el espectro en tiempo real.

El desarrollo de este artículo está dividido en 7 secciones. Luego de la introducción, se presenta una Descripción del sistema. La sección 3, presenta un breve fundamento teórico. En la sección 4 se exponen las herramientas de diseño de software y hardware usadas en el desarrollo del trabajo. La sección 5 muestra la implementación del sistema y los módulos que lo conforman. Luego, en la sección 6 se hace un análisis de resultados, para finalmente, en la sección 7 obtener las conclusiones del desarrollo y los aportes del mismo.

2. DESCRIPCIÓN DEL SISTEMA

Lo que se quiere con el desarrollo de este proyecto, es obtener la Envolvente Compleja de una señal, es decir, obtener la señal en banda base de cualquier señal modulada que tiene una componente espectral igual a la de la señal modulada, para de esta forma, analizar sus componentes en fase (I) y en cuadratura (Q) (fig. 1), observar su espectro, diagrama polar, entre otros.

Esto se logra con un generador de señales sinusoidales (DDS= Direct Digital Synthesizer)[1], quienes hacen de portadoras y un filtro pasa bajas de coeficientes reconfigurables (FIR Compiler 5.0)[2].

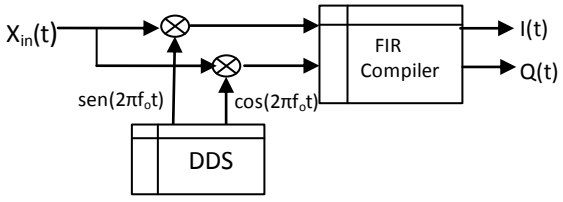


Figura 1. Principio de un Downconverter.

Gracias a la gran variedad de aplicaciones que ofrece System Generator, se hace uso de él en el diseño del sistema, con un diagrama de bloques expuesto en la fig. 2. Es importante resaltar el total uso de los

componentes de la FPGA, tales como conversores, memorias, pantalla, picoblaze y botones, pues se busca explotar al máximo su capacidad.

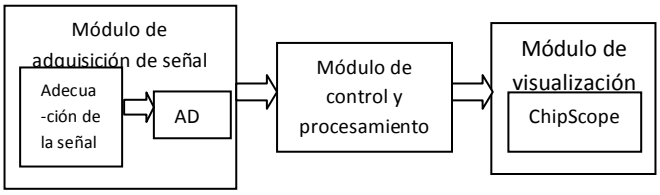


Figura 2. Diagrama de bloques del sistema

Una breve descripción de los bloques:

- Módulo de adquisición: implementa la comunicación con el conversor y entrega datos digitalizados para iniciar el procesamiento.
- Módulo de control y procesamiento: convierte la entrada serial a paralelo, configura el conversor D/A, realiza la programación de picoblaze quien controla todos los elementos del sistema, tales como el DDS compiler, el FIR Compiler, la pantalla, y los botones.
- Módulo de visualización: configura el ChipScope para la visualización en tiempo real de las señales deseadas. También se observan éstas en el dominio del tiempo y la frecuencia, mediante un Guide elaborado en el entorno de Matlab.

3. FUNDAMENTO TEÓRICO

La mayoría de las señales en comunicaciones se modulan tanto en amplitud como fase según la ecuación 1.

$$x(t)=\alpha(t)\cos[\omega_o t+\varphi(t)];\ \omega_o=2\pi fo, \qquad \text{Ec. (1)}$$

donde $\alpha(t)$ es la amplitud y $\varphi(t)$ la fase.

Una señal paso banda x(t) es aquella que tiene concentrada su potencia en una zona concreta del espectro en torno a fo (figura 3) y puede ser representada usando una notación compleja tal como en la ecuación 2.

$$x(t)= \mathrm{Re}\{\bar{x}(t)\, e^{j2\pi fct}\} \qquad \text{Ec. (2)}$$

donde $\bar{x}(t)$ es conocida como la Envolvente Compleja, expresada así:

$$\bar{x}(t)= \alpha(t)e^{j\varphi(t)}= x_I(t) + jx_Q(t) \qquad \text{Ec. (3)}$$

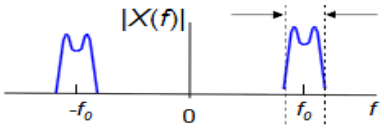


Figura 3. Espectro de una señal paso banda.

De igual forma, las señales paso banda se pueden representar también mediante Componente en Fase [$x_I(t)$] y Componente en Cuadratura [$x_Q(t)$] como se observa en la ecuación 4. Ver figura 4

$$x(t)= x_I(t)\cos(\omega_o t)- x_Q(t)\sin(\omega_o t) \qquad \text{Ec. (4)}$$

donde $x_I(t)= \alpha(t)\cos\varphi(t)$ y $x_Q(t)= \alpha(t)\sin\varphi(t)$

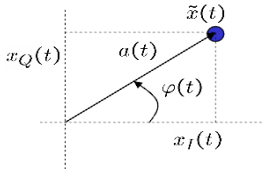
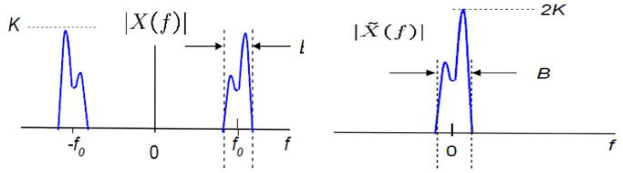


Figura 4. Componentes en fase y en cuadratura de una señal.

La envolvente compleja es una representación fasorial de x(t), y por tanto es una señal compleja equivalente donde el espectro de la envolvente compleja es el mismo de la señal de entrada x(t), con la diferencia de que este primero se encuentra escalado por un factor de 2 y está centrado en la frecuencia 0, es decir, en banda base. Figura 5



(a) *(b)*

Figura 5 (a) Espectro de una señal paso banda. (b) Espectro de la envolvente compleja.

La figura 1 muestra el esquema general con el que se logra obtener la Envolvente compleja: sus componentes en fase y en cuadratura. De la ecuación

2 se puede decir que $\bar{x}(t)$ es el mensaje en banda base o los datos en forma compleja y $e^{j2\pi fct}$ la portadora también en forma compleja. El producto de estas dos señales representa la modulación y x(t), la parte real de este producto, es la onda transmitida. [3]

La gran ventaja de obtener la envolvente compleja de las señales es el menor requerimiento en cuanto procesamiento y frecuencia de muestreo se refiere, pues el análisis de éstas se hace en banda base, logrando mayor eficiencia en recursos y costos.

4. HERRAMIENTAS DE DISEÑO E IMPLEMENTACIÓN

A. Diseño de Hardware

1.) Tarjeta de desarrollo

Una FPGA (Field Programmable Gate Array) es un circuito integrado de propósito general que es programado por un diseñador. La arquitectura de una FPGA consiste en un arreglo de bloques lógicos programables interconectados entre sí mediante canales de conexión verticales y horizontales, los cuales pueden ser programados para duplicar la funcionalidad de puertas lógicas básicas o funciones combinacionales más complejas. [4]

Es programada por medio de la descarga de un archivo de configuración llamado Bitstream en una memoria de acceso aleatorio RAM. Este bitstream es el producto de las herramientas de compilación quienes traducen abstracciones de alto nivel hechas por el diseñador en algo equivalente pero en bajo nivel y ejecutable.

La tarjeta de desarrollo seleccionada en el presente proyecto, Spartan 3A, se destaca por ser de muy bajo costo, con lógica de alto rendimiento, poseer abundantes recursos de lógica lexible, entre otros, los cuales hacen de ella un dispositivo apropiado para un amplio rango de aplicaciones electrónicas, tales como acceso a banda ancha, redes domésticas, equipamiento de televisión digital, entre otros. En la tabla 1 se pueden observar sus principales características:

TABLA 1.
CARACTERÍSTICAS PRINCIPALES DE LA FPGA SPARTAN 3A.

Dispositivo: Spartan 3A Características:	
Modelo	XC3S700A
System Gates	700k
Equivalent logic cells	13248
Array size	48x32
Slices	5888
Distributed RAM bits [1]	92k
Block RAM bits	360k
Dedicated multipliers	20
DCMs	8
Maximum user I/O	372
Maximum differential I/O pairs	165

2.) Herramientas de Xilinx y entorno de desarrollo ISE

Xilinx es una empresa que se ha encargado de crear herramientas EDA (Electronic Design Automation) para facilitar el diseño, la implementación y la programación de las FPGA. ISE (Integrated Synthesis Enviroment) es una de ellas y en su entorno se tiene: Diseño mediante VHDL, Verilog-HDL, captura esquemática, síntesis, simulación, implementación y por último, programación del dispositivo.

3.) iMPACT

IMPACT es una herramienta de configuración que permite implementar el diseño en el hardware compilando archivos *.bit, *.mpm, además de programar la memoria flash. [5]

4.) ChipScope Pro y comunicación con el PC

Las herramientas del ChipScope Pro proporcionan una exploración del circuito en tiempo real con un bajo costo. Al insertar núcleos (ICON, ILA, VIO, ATC2) en el código VHDL, se puede depurar y verificar la lógica de la FPGA, la actividad de los

buses, capturando señales a velocidades del sistema y observando todas las señales internas en la Spartan.

Este consta de 3 componentes: Core inserter, Core Generator y Chipscope Pro Analyzer y soporta los cables USB y IV Paralelo para la comunicación entre el PC y el dispositivo en la cadena del JTAG. [6]

B. Diseño De Software

1.) System Generator (SysGen)

Xilinx System Generator for DSP es una herramienta software de modelado y programación gráfica basada en MathWorkMATLAB/Simulink. Permite diseñar sobre un entorno de alto nivel, flexible, robusto y fácil de usar, sistemas DSP de alto rendimiento para un determinado hardware, mediante funciones lógicas, memorias y funciones DSP ideales para el filtrado digital, análisis espectral y comunicaciones digitales.

Esta herramienta convierte el modelo de bloques de Xilinx de Simulink en una muy eficiente implementación de hardware que combina el código VHDL personal con la propiedad intelectual de los bloques que ya están listos para ejecutarse con alto rendimiento en la FPGA.

El diseño es creado en el entorno de Matlab, usando librerías de Xilinx. Su gran ventaja es el tener acceso a la herramienta de simulación de Simulink y de esta forma verificar y modificar el funcionamiento del sistema. Esta asociación de SysGen con Matlab, genera nuevos blocksets en el toolbox de Simulink:

- Xilinx Blockset: es una familia de librerías que contiene bloques básicos del System Generator. Algunos para el acceso al hardware específico del dispositivo y otros para procesamiento de señales y algoritmos avanzados de comunicación.
- Xilinx Reference Blockset: contiene bloques compuestos de SysGen que implementan un amplio rango de funciones.
- Xilinx XtremeDSP kit: contiene bloques diseñados especialmente para el procesamiento digital de señales y aplicaciones DSP. [7]

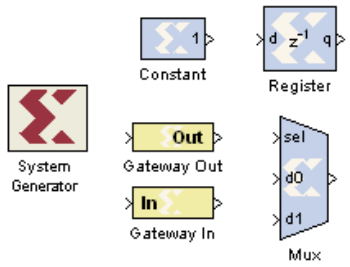


Figura 6. Ejemplos de bloques de SysGen

En la fig. 6 se muestran algunos de los bloques de SysGen.

2.) Interfaz de la comunicación

La comunicación con el PC, se hace mediante el cable JTAG y ChipScope. Esta herramienta de debugging, permite observar, analizar y verificar todas las señales internas de la tarjeta y así lograr un mejor desempeño del dispositivo.

5. IMPLEMENTACIÓN

El esquema general del proyecto es:

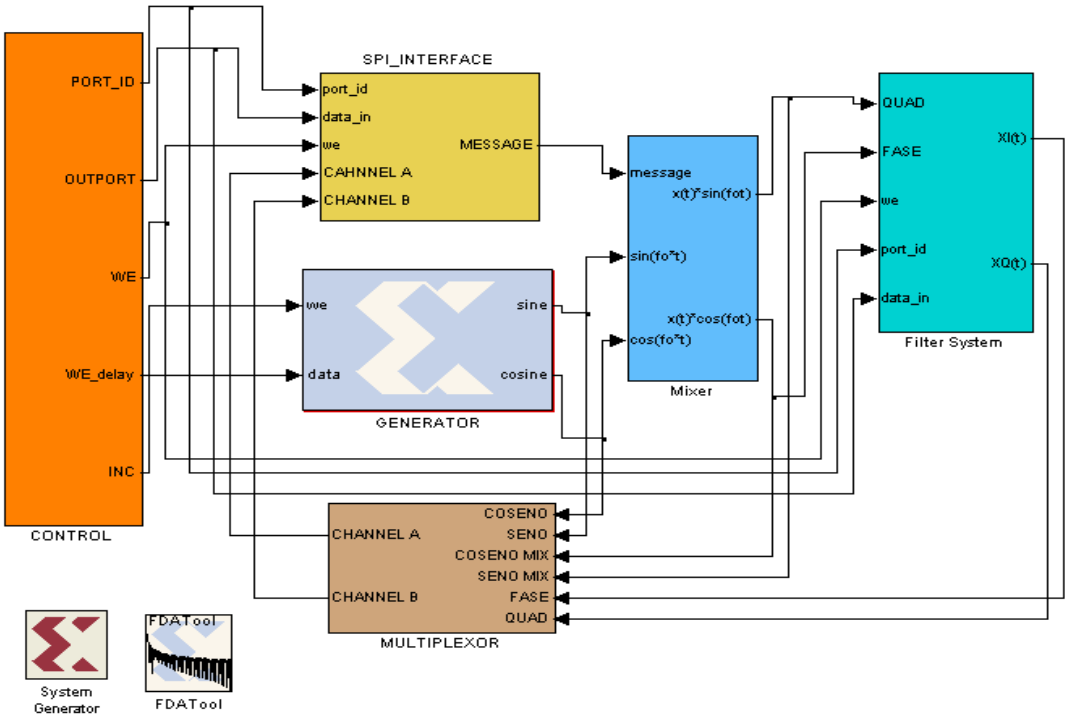


Figura 7. Esquema general del diseño en Simulink.

Se pueden observar los siguientes bloques:

A. *Control*

Este bloque corresponde a la unidad de control general del sistema. Contiene una interfaz para botones, pantalla, microcontrolador (PicoBlaze), una adecuación de la señal y un conversor binario-decimal, para poder visualizar los datos en la pantalla de la FPGA. Fig. 8

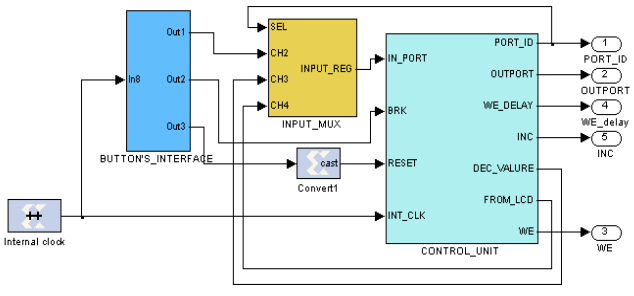


Figura 8. Bloques internos del subsistema Control.

Los bloques que conforman la unidad de control son:

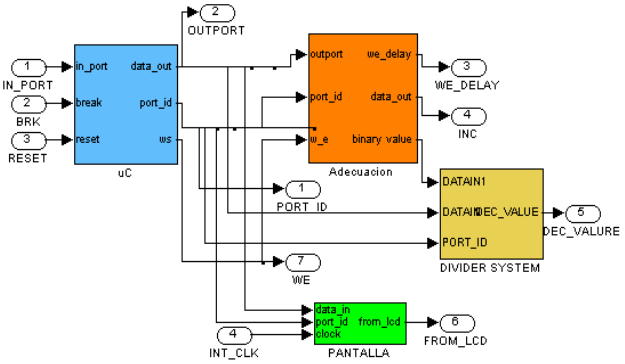


Figura 9. Bloques internos de la Unidad de Control.

B. *Interface SPI*

Mediante esta interface, se controlan tanto el pre-amplificador como el conversor A/D. Debido a que SysGen no tiene bloques específicos para la

configuración de éstos en la tarjeta Spartan 3A, su implementación se realiza mediante el bloque BlackBox. Fig 10. Este permite usar un proyecto realizado en VHDL e implementarlo en SysGen. En este caso es una máquina de estados que genera las señales de control para los dispositivos que manejan el protocolo SPI (Serial Peripheral Interface).

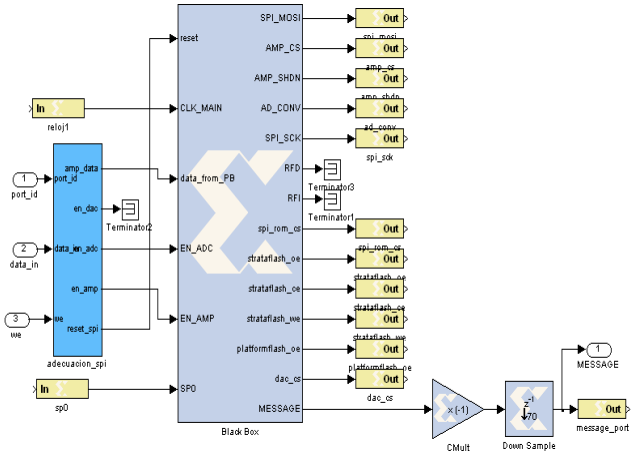


Figura 10. Bloques internos del subsistema SPI_INTERFACE

C. Generador

El DDS Compiler es el encargado de generar las señales portadoras (Seno y Coseno) a la frecuencia deseada por el usuario. La frecuencia máxima a generar es 3MHz. Fig. 11.

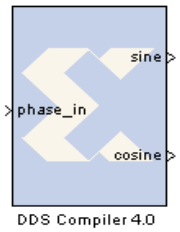


Figura 11. Bloque DDS Compiler

D. Mezclador

El mezclador se compone de 2 multiplicadores de 15bits cada uno y su función es mezclar el mensaje con cada una de las portadoras y de esta manera, pasar a banda base las señales entrantes.

E. Filtro FIR Compiler

El FIR Compiler (Respuesta al impulso finita), de alto rendimiento, implementa transformada de Hilbert, banco de filtros polifásicos, filtros interpolados, entre otros, además de la diponibilidad de 2 arquitecturas diferentes: *Distributed Arithmetic y Multiply-Accumulate (MAC).*

Entre sus características principales se encuentran:

- Soporta hasta 256 sets de coeficientes con 2 hasta 2048 coeficientes por set.
- Soporta hasta 64 canales de entrada
- Soporta múltiples datapaths paralelos con lógica de control compartida.
- Capaz de reconfigurar los coeficientes [2]

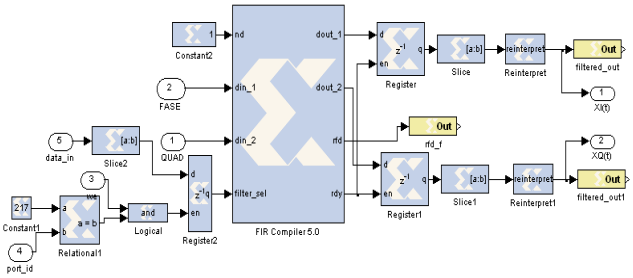


Figura 12. Bloque del FIR Compiler V5.0

Los módulos del sistema funcionan de la siguiente manera:

1. Módulo de adquisición

El módulo de adquisición de la señal consta de una adecuación de señal y un conversor A/D. La adecuación de señal consiste en una implementación externa de shifters level a través de amplificadores operacionales y resistencias, debido a que la señal de entrada al conversor debe estar sobre un nivel de DC de 1.65V.

El pre-ampificador permite modificar la amplitud de la señal analógica para adecuarla de acuerdo a la necesidad del usuario. El conversor A/D de 14 bits recibe dichas señales, realiza la conversión y las

transmite de manera serial. Este funciona a la frecuencia del reloj de 50MHz pero debido a los 34 bits que trabaja y que corresponden a los 2 canales, los datos sólo son fiables hasta una frecuencia de muestreo de aproximadamente 350KHz.

El protocolo de comunicación se encuentra detallado en la guía de usuario de la Spartan 3A [4]. Esta configuración, así como la del DAC se hace en el bloque de interfaz SPI.

2. Módulo de control y procesamiento

Este módulo es el control principal del diseño, conduce las señales de control y de dirección requeridas por los componentes. Realiza la configuración de los elementos externos, es decir, botones y pantalla. A nivel de software, configura y controla el DDS y el FIR Compiler con el microcontrolador integrado PicoBlaze.

El bloque DDS (Direct Digital Synthesizer) u oscilador controlado numéricamente, es un importante componente en sistemas de comunicaciones digitales. Éste genera fuentes sinusoidales de una frecuencia determinada, empleando el esquema de una LUT (Lookup table). Véase fig. 13.

La LUT guarda muestras de una sinusoidal y a partir de una fase acumulada se extraen las señales (seno, coseno) a una frecuencia deseada. Un integrador digital es usado para generar un argumento de fase adecuado que es mapeado por una lookup table a una frecuencia de salida deseada. [1]

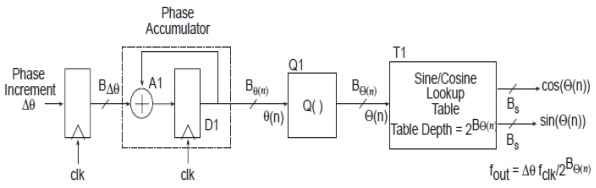


Figura 13. Vista simplificada del DDS core.

La frecuencia de salida f_{out} de la señales generadas por la DDS, depende de la frecuencia del reloj del sistema f_{clk}, el número de bits en el acumulador de fase $B_{\Theta(n)}$ y el valor del incremento de fase $\Delta\theta$ de acuerdo a la ecuación (5):

$$f_{out} = \frac{f_{clk}\,\Delta\theta}{2^{B(n)}}\ [\text{Hz}] \qquad \text{Ec. (5)}$$

El DDS utilizado en el proyecto genera señales a una frecuencia máxima de 3MHz.

Estas señales generadas por el DDS (portadoras), se mezclan con el mensaje de entrada, dando lugar a las componentes de fase y cuadratura. Para obtener el mensaje puro, es necesario el uso de filtros pasa bajas, los cuales eliminen las réplicas generadas por la multiplicación y dejen a la vista la frecuencia del mensaje exclusivamente.

Para esto, se hace uso del bloque FIR compiler, el cual proporciona una interfaz a los usuarios para generar filtros FIR parametrizables y de alto rendimiento.

Este bloque ofrece la posibilidad de tener coeficientes reconfigurables y diversos canales de entrada. Además permite extraer los coeficientes directamente de la herramienta FDATool, con la cual se pueden diseñar filtros según la necesidad. La arquitectura utilizada fue Transpose Multiply-Accumulate, la cual se ilustra en la figura 14. Se configuró para 12 bits y con datapath en paralelo para una mayor eficiencia.

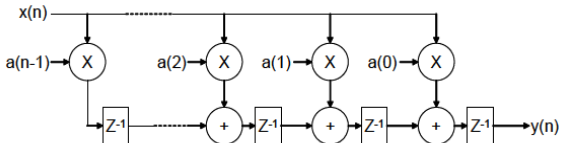


Figura 14. Forma directa de la Arquitectura Transpose Multiply-Accumulate. Fuente: Product Specification LogicCORE IP FIR Compiler V5.0

Módulo de visualización

La comunicación con el PC se realiza mediante el Chipscope pro a través del cable USB JTAG. Es necesario insertar núcleos al diseño para de esta forma, lograr capturar cualquier señal interna del sistema.

Luego de capturarlas, es posible exportar estas señales al directorio de Matlab, observarlas y realizar el análisis que se desee utilizando las herramientas que se brindan en el GUIDE diseñado.

Además de la visualización en el PC mediante ChipScope y el GUIDE, se pueden observar las señales en fase y cuadratura, antes y después de filtrar a través del DAC (Digital to Analog converter). Este está configurado para dos canales con entradas de 12 bits cada uno.

6. ANÁLISIS DE RESULTADOS

A. Pruebas de laboratorio.

A continuación un ejemplo de cómo funciona el dispositivo:

Se procede a capturar el mensaje en una modulación FM, el cual se modula con una portadora de 15KHz. La forma de onda se puede apreciar en la figura 15 y es capturada desde el entorno del ChipScope.

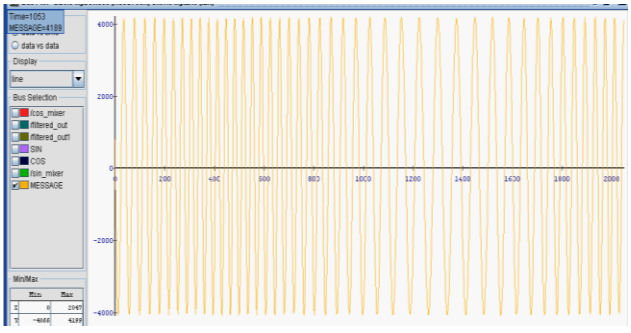


Figura 15. S(t), muestreada e impresa en el chipscope Pro Analyzer

El siguiente paso es seleccionar en el dispositivo una frecuencia de oscilación igual a la de la portadora. Al seleccionar la frecuencia deseada, ya se cuenta con una señal seno y coseno que serán multiplicados por la señal entrante. El resultado se aprecia en la figura 16

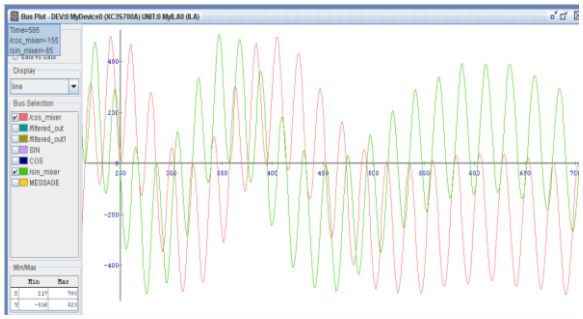


Figura 16 Señales del mezclador visualizadas en el entorno del Chipscope: salida del mezclador del seno (Verde), salida del mezclador del coseno (Rojo)

Luego es necesario filtrar la señal. Para este fin se usa un filtro digital con una frecuencia de corte de 20Khz, que permite recuperar solo la componente espectral que está situada en banda base y elimina la que queda ubicada alrededor de los 30Khz. En este momento ya se puede hablar de la Envolvente compleja de la señal S(t) que ingresó al sistema.

La figura 17 contiene las salidas de los dos filtros del sistema, comparados con sus respectivas entradas. En cada una de las gráficas se aprecia la aparición de la envolvente compleja una vez filtrada la señal.

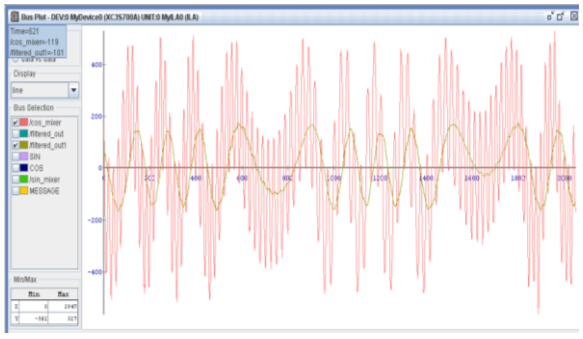
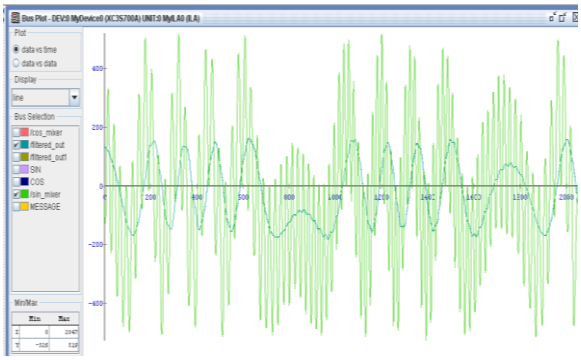


*Figura 17 a.) Señal de mezclador s(t)*cos(2*pi*fo*t)(rosado),Componente en fase (café*



b.) *Señal de mezclador s(t)*sin(2*pi*fo*t)(verde), Componente en cuadratura (azul)*

La figura 18 muestra el diagrama polar de la envolvente compleja de la señal. La gráfica es generada usando como eje "X" la componente en fase de la señal y como eje "Y" la componente en cuadratura. En el gráfico se aprecian la características de la modulación FM, es decir, la amplitud del mensaje no cambia y su fase sí, generando un punto que se mueve alrededor de una circunferencia en el plano polar.

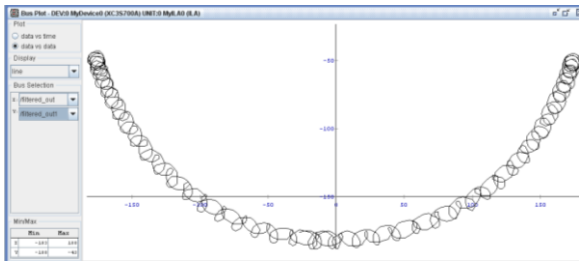


Figura18 Diagrama Polar de la envolvente compleja

El ChispScope permite exportar el contenido de las señales a un archivo ASCII. Dicho archivo se cargó en el entorno de Matlab. De este modo se pueden graficar las señales que se capturaron desde el dispositivo y es posible ver su espectro usando el comando FFT. En la figura 19 se aprecia además de un resumen de las señales, sus respectivos espectros, en donde se puede ver el cambio que sufrieron éstos. El resultado de dichos espectros concuerda con lo que se espera que haga el sistema, es decir, que se obtenga en banda base una componente espectral de la misma forma que la componente de la señal modulada y las demás componentes espectrales resultantes sean filtradas.

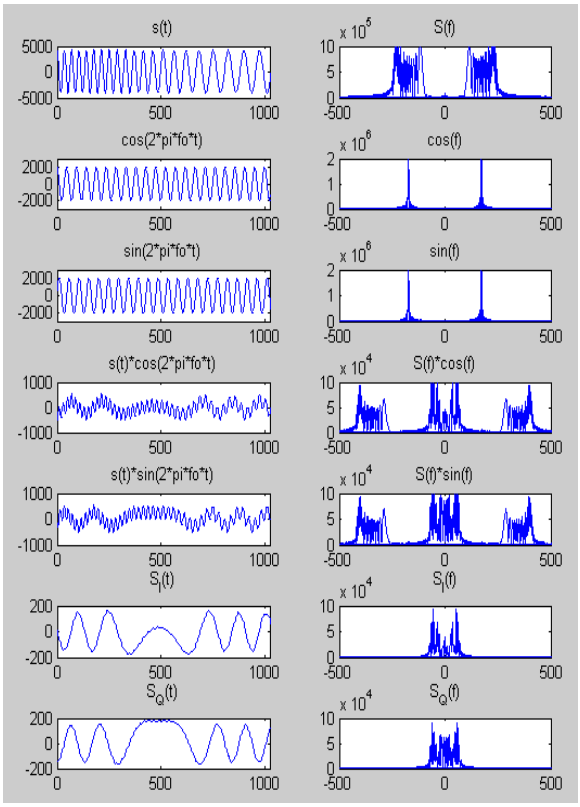


Fig.19 Resumen de señales y sus respectivos espectros

7. CONCLUSIONES Y CONTRIBUCIONES

- Se logró realizar un Downconverter con funciones implementadas en una FPGA y visualización a través de una interfaz en el pc. Se logró también que fuera de fines pedagógicos, pues la manipulación de sus señales y la gran variedad de herramientas utilizadas, permiten una formación en la comprensión del comportamiento de dichas señales tanto en el dominio del tiempo como en el de la frecuencia

- El Downconverter implementado en este proyecto tiene características sobresalientes tales como su bajo costo, su tamaño y que puede ser utilizado en cualquier laboratorio con la disposición de computadores u osciloscopios.

- Se aprovecharon y utilizaron correctamente los componentes y recursos de la FPGA gracias al uso de la herramienta System Generator, la cual optimiza los procesos.

- El ancho de banda manejado por el dispositivo está limitado por aspectos externos, como la velocidad de los dispositivos de conversión y la transmisión de datos al PC. Esta limitante de ancho de banda generada por los conversores se puede corregir usando componentes externos que funcionen más rápido y/o en paralelo.

- El trabajo presentado deja un precedente para futuras aplicaciones de SDR sobre FPGA's, y establece una línea para desarrollar nuevos proyectos pedagógicos en el campo de la Comunicaciones Digitales.

RECONOCIMIENTOS

Se hace especial reconocimiento al Grupo de Investigación RadioGIS y a la Universidad Industrial de Santander pues gracias a su apoyo y colaboración se pudo llevar a cabo exitosamente el presente trabajo.

REFERENCIAS

[1] 'LogiCORE IP DDS Compiler v4.0.' Product Specification. Xilinx, 2010

[2] 'LogiCORE IP FIR Compiler v5.0'. Product Specification. Xilinx, 2010

[3] Sklar, Bernard. Digital Communications: Fundamentals and Applications. Second Edition. Prentice Hall. University of California, Los Ángeles.

[4] Spartan-3A FPGA Starter Kit Board. User Guide (UG330) Xilinx, 2007

[5]'Using the ISE Design Tools for Spartan-3 FPGAs". Application note: Spartan-3 FPGA Family. Xilinx. 2003

[6] ChipScope Pro 12.1 Software and Cores. User Guide. (UG029) Xilinx, 2010

[7] System Generator for DSP. (UG639) Xilinx, 2010

ANEXOS

ANEXO A. Código de programación del microcontrolador PicoBlaze

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;
;;;---------DECLARACION DE LAS CONSTANTES DEL CODIGO ASCII DE LA PANTALLA--------;;;,
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;
        ; ASCII tabla
        CONSTANT character_a, 61
        CONSTANT character_b, 62
        CONSTANT character_c, 63
        CONSTANT character_d, 64
        CONSTANT character_e, 65
        CONSTANT character_f, 66
        CONSTANT character_g, 67
        CONSTANT character_h, 68
        CONSTANT character_i, 69
        CONSTANT character_j, 6A
        CONSTANT character_k, 6B
        CONSTANT character_l, 6C
        CONSTANT character_m, 6D
        CONSTANT character_n, 6E
        CONSTANT character_o, 6F
        CONSTANT character_p, 70
        CONSTANT character_q, 71
        CONSTANT character_r, 72
        CONSTANT character_s, 73
```

CONSTANT character_t, 74

CONSTANT character_u, 75

CONSTANT character_v, 76

CONSTANT character_w, 77

CONSTANT character_x, 78

CONSTANT character_y, 79

CONSTANT character_z, 7A

CONSTANT character_A, 41

CONSTANT character_B, 42

CONSTANT character_C, 43

CONSTANT character_D, 44

CONSTANT character_E, 45

CONSTANT character_F, 46

CONSTANT character_G, 47

CONSTANT character_H, 48

CONSTANT character_I, 49

CONSTANT character_J, 4A

CONSTANT character_K, 4B

CONSTANT character_L, 4C

CONSTANT character_M, 4D

CONSTANT character_N, 4E

CONSTANT character_O, 4F

CONSTANT character_P, 50

CONSTANT character_Q, 51

CONSTANT character_R, 52

CONSTANT character_S, 53

CONSTANT character_T, 54

```
CONSTANT character_U, 55

CONSTANT character_V, 56

CONSTANT character_W, 57

CONSTANT character_X, 58

CONSTANT character_Y, 59

CONSTANT character_Z, 5A

CONSTANT character_0, 30

CONSTANT character_1, 31

CONSTANT character_2, 32

CONSTANT character_3, 33

CONSTANT character_4, 34

CONSTANT character_5, 35

CONSTANT character_6, 36

CONSTANT character_7, 37

CONSTANT character_8, 38

CONSTANT character_9, 39

CONSTANT character_colon, 3A

CONSTANT character_stop, 2E

CONSTANT character_semi_colon, 3B

CONSTANT character_minus, 2D

CONSTANT character_divide, 2F       ;'/'

CONSTANT character_plus, 2B

CONSTANT character_comma, 2C

CONSTANT character_less_than, 3C

CONSTANT character_greater_than, 3E

CONSTANT character_equals, 3D

CONSTANT character_space, A0
```

```
CONSTANT character_CR, 0D          ;carriage return

CONSTANT character_question, 3F     ;'?'

CONSTANT character_dollar, 24

CONSTANT character_exclaim, 21      ;'!'

CONSTANT character_BS, 08          ;Back Space command character

CONSTANT character_RARROW, 7E          ;FLECHA HACIA LA DERECHA

CONSTANT character_LARROW, 7F          ;FLECHA HACIA LA IZQUIERDA

;;------DECLARACIONDE DIVERSOS PUERTOS DE ENTRADA Y SALIDA AL PICOBLAZE-----------
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;----------- PUERTOS DE ENTRADA--------------------------------------------------------------------------

CONSTANT BREAK_PORT,01                          ; PUERTO DE ENTRADA DEL REGISTRO DE INTERRUPCIONES

CONSTANT WEST_BUTTON,01                  ; BIT0: CORRESPONDIENTE AL BOTON OESTE

CONSTANT EAST_BUTTON,02                  ; BIT1: CORRESPONDIENTE AL BOTON ESTE

CONSTANT SOUTH_BUTTON,04                 ; BIT2: CORRESPONDIENTE AL BOTON SUR

CONSTANT NORTH_BUTTON,08                 ; BIT3: CORRESPONDIENTE AL BOTON NORTE

CONSTANT RLEFT,10                        ; BIT4: CORRESPONDIENTE AL RLEFT QUE INDICA EL SENTIDO DE LA ROTACION DEL ROTARY KNOB

CONSTANT DEC_VALUE_PORT,02                  ; REGISTRO QUE CONTIENE EL RESIDUO DE LA OPERACION DE LA DIVISION PROVENIENTE DEL NUCLEO DE DIVISION

CONSTANT RFI_PORT,00                           ; REGISTRO QUE CONTIENE LA BANDERA DE CUANDO SE TERMINÓ DE CONFIGURAR EL AMPLIFICADOR SPI

CONSTANT RFI_READY,01                      ; BIT0: SOLO TIENE UN BIT Y ES EL QUE INDICA LA BANDERA

;;------------PUERTOS DE SALIDA-----------------------------------------------------------------------------

CONSTANT REGOUT1_PORT,1A

CONSTANT REGOUT2_PORT,1B

CONSTANT REGOUT3_PORT,1C

CONSTANT DIV_CONTROL_PORT,1F                    ; PUERTO DE SALIDA PARA NUCLEO DE DIVISION  |X|X|X|X|X|EN_DAC|EN_ADC|EN_AMP|
```

```
CONSTANT MUX_SEL,01                ;BIT0

CONSTANT EN_QUOTIENT,02            ;BIT1

CONSTANT EN_DIVIDER,04             ;BIT2

CONSTANT SPI_CONTROL_PORT,FD       ; PUERTO DE SALIDA PARA EL CONTROL DEL BLOQUE DE INTERFAZ SPI  |X|X|X|X|X|EN_DAC|EN_ADC|EN_AMP|

CONSTANT EN_AMP,01                 ;BIT0

CONSTANT EN_ADC,02                 ;BIT1

CONSTANT EN_DAC,04                 ;BIT2

CONSTANT RST_SPI,08                ;BIT4


CONSTANT AMP_DATA,FE               ; PUERTO DE SALIDA CORRESPONDINETE A LAS GANANCIAS DEL AMPLIFICADOR ---------- |B3|B2|B1|B0|A3|A2|A1|A0|

CONSTANT SELECTOR_PORT,3E          ; PUERTO DE SELCCIOON DEL MUX DE ENTARDA DE LA DIVISION DE VALORES CONSECUTIVOS

CONSTANT AMP_DIV_PORT ,3F

CONSTANT FILTER_DEC,5B             ;PUERTO POR DONDE SE OBTIENE LA REPRESENTACION BINARIA DEL FILTRO.

CONSTANT NUMBER_FILTER_PORT,D9     ;PUERTO DE SALIDA PARA CONTROLAR EL SET DE COEFICIENTES QUE SE VA A USAR EN EL FILTRO

;;---------- PUERTOS DE INTERFACE CON LA PANTALLA LCD----------

CONSTANT LCD_output_port, 40       ;LCD character module output data

CONSTANT LCD_input_port, 03        ;LCD character module input data

CONSTANT LCD_DB0, 01               ;  8-bit        DB4 - bit0

CONSTANT LCD_DB1, 02               ;  interface    DB5 - bit1

CONSTANT LCD_DB2, 04               ;               DB6 - bit2

CONSTANT LCD_DB3, 08               ;               DB7 - bit3

CONSTANT LCD_DB4, 10               ;               DB4 - bit4

CONSTANT LCD_DB5, 20               ;               DB5 - bit5

CONSTANT LCD_DB6, 40               ;               DB6 - bit6
```

```
CONSTANT LCD_DB7, 80              ;              DB7 - bit7
CONSTANT LCD_control_port, 20     ;LCD character module control signals
CONSTANT LCD_E, 01                ;   active High Enable       E - bit0
CONSTANT LCD_RW, 02               ;   Read=1 Write=0           RW - bit1
CONSTANT LCD_RS, 04               ;   Instruction=0 Data=1     RS - bit2

; delay_1us_constant =  (clock_rate - 6)/4       Where 'clock_rate' is in MHz
CONSTANT delay_1us_constant, 0B
        ;;-----DECLARACION DE LAS CONSTANTES DEL SCRATCHPAD-----------;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
CONSTANT DELTHA11,00
CONSTANT DELTHA12,00
CONSTANT DELTHA13,05
CONSTANT DELTHA21,00
CONSTANT DELTHA22,00
CONSTANT DELTHA23,36
CONSTANT DELTHA31,00
CONSTANT DELTHA32,02
CONSTANT DELTHA33,19
CONSTANT DELTHA41,00
CONSTANT DELTHA42,14
CONSTANT DELTHA43,F9
CONSTANT DELTHA51,00
CONSTANT DELTHA52,D1
CONSTANT DELTHA53,B7
CONSTANT DELTHA61,08
CONSTANT DELTHA62,31
```

```
CONSTANT DELTHA63,27

CONSTANT INIT,       00              ; configura el valor por defecto del deltha de fase.

CONSTANT LUGAR_CONT,  12          ;posicion donde va a quedar almacenado el valor del contador

CONSTANT PLACE1,13       ; señala los lugares del scartchpad donde se ubicaran los valores decimales de la frecuencia

CONSTANT PLACE2,14       ; señala los lugares del scartchpad donde se ubicaran los valores decimales de la frecuencia

CONSTANT PLACE3,15       ; señala los lugares del scartchpad donde se ubicaran los valores decimales de la frecuencia

CONSTANT PLACE4,16       ; señala los lugares del scartchpad donde se ubicaran los valores decimales de la frecuencia

CONSTANT PLACE5,17       ; señala los lugares del scartchpad donde se ubicaran los valores decimales de la frecuencia

CONSTANT PLACE6,18       ; señala los lugares del scartchpad donde se ubicaran los valores decimales de la frecuencia

CONSTANT PLACE7,19       ; señala los lugares del scartchpad donde se ubicaran los valores decimales de la frecuencia

;;------------------LUGARES EN EL SP PARA ALMACENAR LAS POSIBLES GANANCIAS DEL AMPLIFICADOR SPI----------;;;;

CONSTANT GAIN0,   1A

CONSTANT GAIN1,   1B

CONSTANT GAIN2,   1C

CONSTANT GAIN5,   1D

CONSTANT GAIN10,  1E

CONSTANT GAIN20,  1F

CONSTANT GAIN50,  20

CONSTANT GAIN100, 21


CONSTANT CURRENT_GAIN,22          ; POSICION DEL SP QUE CONTIENE EL VALOR PRESENTE DE LA GANANCIA DEL AMPLIFICADOR
```

CONSTANT CURRENT_GAIN_POSITION,23 ; POSICION QUE ALMACENA LA UBICAION ACTUAL DE DONDE SE LEYÓ EL DATO DEL AMPLIFICADOR

CONSTANT SELECT, 24 ; POSICION DEL SP DONDE SE ALMACENA EL VALOR DE LA SELECCION DE LA SEÑAL

CONSTANT BYNARY_GAIN,25

CONSTANT CURRENT_FILTER,26 ;VALOR QUE TIENE RL NUMERO DEL FILTRO QUE SE VA A IMPLEMENTAR (0-1KHZ 1-5KHZ 2-10KHZ 3-20KHZ 4-50KHZ)

;;-----------------Nombre de los registros Especificos--;

;;

NAMEREG SE, DELTHA1 ; registro destinado para el incremento de los valores de deltha.

NAMEREG SD, DELTHA2 ; registro destinado para el incremento de los valores de deltha.

NAMEREG SC, DELTHA3 ; registro destinado para el incremento de los valores de deltha.

NAMEREG S9, REGOUT1

NAMEREG SA, REGOUT2

NAMEREG SB, REGOUT3

NAMEREG SF, CONT ; registro que va a servir de contador en distintos procesos.

;;--------INICIO DE LA PROGRAMACION----------;;

;;

ENABLE INTERRUPT

cold_start:

LOAD S0,00

OUTPUT S0,SPI_CONTROL_PORT

CALL initreg ; load all the registers

CALL STOREKTES ; carga los valores que se van a utilizar en el scatchpad para los incrementos.

CALL LCD_reset

CALL WELCOME

CALL WAIT2S

```
CALL CONF_AMP


start1:

        COMPARE s8, 01              ; if s8 equals 01 set the zero flag to 1 (carry flag remains 0)

        CALL Z,main_routine         ; ADD CODE HERE TO CALL THE ROUTINE output_s0 when s2 contains the value 01.

        JUMP start1

;-------------------------------MAIN_ROUTINE----------------------------

main_routine:

DISABLE INTERRUPT

FETCH S1,SELECT

COMPARE S1,00

CALL Z,RUTINA_OSCILADOR

FETCH S1,SELECT

COMPARE S1,01

CALL Z,RUTINA_AMP

FETCH S1,SELECT

COMPARE S1,02

CALL Z,RUTINA_FILTRO

LOAD S8,00

ENABLE INTERRUPT

RETURN


RUTINA_OSCILADOR:

CALL output_s0

LOAD S0,00

OUTPUT S0,SELECTOR_PORT
```

```
CALL LOAD_DEC_VALUE
CALL LCD_clear
CALL PRINT_LCD_FREC
CALL PRINT_LCD_DELTHA
RETURN

RUTINA_AMP:
CALL CONF_AMP
FETCH S6,CURRENT_GAIN
OUTPUT S6,AMP_DIV_PORT
LOAD S0,01
OUTPUT S0,SELECTOR_PORT
CALL LOAD_DEC_VALUE
CALL LCD_clear
CALL PRINT_AMP
CALL PRE_AMPLIFIER
RETURN

RUTINA_FILTRO:
    FETCH S0,CURRENT_FILTER
    OUTPUT S0,NUMBER_FILTER_PORT

    COMPARE S0, 00
    JUMP Z,IF_0
    COMPARE S0, 01
    JUMP Z,IF_1
    COMPARE S0, 02
```

```
    JUMP Z,IF_2
    COMPARE S0, 03
    JUMP Z,IF_3
    COMPARE S0, 04
    JUMP Z,IF_4

IF_0: LOAD S1,01
    JUMP FIN_IF
IF_1: LOAD S1,05
    JUMP FIN_IF
IF_2: LOAD S1,0A
    JUMP FIN_IF
IF_3: LOAD S1,14
    JUMP FIN_IF
IF_4: LOAD S1,32

FIN_IF: OUTPUT S1,FILTER_DEC
    LOAD S2,02
    OUTPUT S2,SELECTOR_PORT
    CALL LOAD_DEC_VALUE
    CALL LCD_clear
    CALL LETRERO_FILTER
RETURN

CONF_AMP:
FETCH S5,BYNARY_GAIN
OUTPUT S5,AMP_DATA
```

```
LOAD S5,RST_SPI
OUTPUT S5,SPI_CONTROL_PORT
CALL wait_15
XOR S5,RST_SPI
OUTPUT S5,SPI_CONTROL_PORT
LOAD S5,EN_AMP
OUTPUT S5,SPI_CONTROL_PORT
CALL wait_1ms
XOR S5,EN_AMP
OUTPUT S5,SPI_CONTROL_PORT
call wait_1ms
LOAD S0,06
OUTPUT S0,SPI_CONTROL_PORT
RETURN
; ---------- End Main routine ----------
STOREKTES:
   LOAD DELTHA3,DELTHA63
   STORE DELTHA3,11
   LOAD DELTHA2,DELTHA62
   STORE DELTHA2,10
   LOAD DELTHA1,DELTHA61
   STORE DELTHA1,0F
   LOAD DELTHA3,DELTHA53
   STORE DELTHA3,0E
   LOAD DELTHA2,DELTHA52
   STORE DELTHA2,0D
   LOAD DELTHA1,DELTHA51
```

```
STORE DELTHA1,0C
LOAD DELTHA3,DELTHA43
STORE DELTHA3,0B
LOAD DELTHA2,DELTHA42
STORE DELTHA2,0A
LOAD DELTHA1,DELTHA41
STORE DELTHA1,09
LOAD DELTHA3,DELTHA33
STORE DELTHA3,08
LOAD DELTHA2,DELTHA32
STORE DELTHA2,07
LOAD DELTHA1,DELTHA31
STORE DELTHA1,06

LOAD DELTHA3,DELTHA23
STORE DELTHA3,05
LOAD DELTHA2,DELTHA22
STORE DELTHA2,04
LOAD DELTHA1,DELTHA21
STORE DELTHA1,03
LOAD DELTHA3,DELTHA13
STORE DELTHA3,02
LOAD DELTHA2,DELTHA12
STORE DELTHA2,01
LOAD DELTHA1,DELTHA11
STORE DELTHA1,00
LOAD S0,00
```

```
    STORE S0,GAIN0
    LOAD S0,01
    STORE S0,GAIN1
    STORE S0,CURRENT_GAIN
    LOAD S0,02
    STORE S0,GAIN2
    LOAD S0,05
    STORE S0,GAIN5
    LOAD S0,0A
    STORE S0,GAIN10
    LOAD S0,14
    STORE S0,GAIN20
    LOAD S0,32
    STORE S0,GAIN50
    LOAD S0,64
    STORE S0,GAIN100
    LOAD S0,GAIN1
    STORE S0,CURRENT_GAIN_POSITION
    LOAD S0,00
    STORE S0,SELECT
    LOAD S0,01
    STORE S0,BYNARY_GAIN
    LOAD S0,00
    STORE S0,CURRENT_FILTER
    RETURN

initreg:                    ; initialize the registers
```

```
   LOAD REGOUT1,INIT              ; carga el valor inicial de deltha
   LOAD REGOUT2,INIT              ; carga el valor inicial de deltha
   LOAD REGOUT3,INIT              ; carga el valor inicial de deltha
        LOAD s8, 01               ; esta es la condicion para que salga de la iteracion del ciclo infinito
   LOAD CONT,03                   ; inizializa el registro contador en cero.
   STORE CONT, LUGAR_CONT         ; guarda el valor actual del contador para hacer reuso del registro CONT
   RETURN

output_s0:          ;output all the register values
        OUTPUT REGOUT1,REGOUT1_PORT     ;  OUTPUT REGISTER  PORT_ID 1A
   OUTPUT REGOUT2,REGOUT2_PORT     ;  OUTPUT REGISTER  PORT_ID 1B
   OUTPUT REGOUT3,REGOUT3_PORT     ;  OUTPUT REGISTER  PORT_ID 1C
        RETURN          ; return to program location before last jump

DECREMENTO:             ; establece  la rutina en caso de giro hacia la izquierda
   LOAD  S3,FF
   SUB   REGOUT3,DELTHA3
   SUBCY REGOUT2,DELTHA2
   SUBCY REGOUT1,DELTHA1
   CALL C,SATURAR
   RETURN

INCREMENTO:             ; estable la rutina en caso de giro hacia la derecha
   LOAD S3,00
   ADD   REGOUT3,DELTHA3
   ADDCY REGOUT2,DELTHA2
```

```
   ADDCY REGOUT1,DELTHA1
   CALL C,SATURAR
   RETURN

SATURAR:                    ; retorna el valor de salida  a su valor anteror cuando se da un carry o borrow dependiendo de la operación
LOAD S2,FF
XOR S2,S3
LOAD REGOUT3,S2
LOAD REGOUT2,S2
LOAD REGOUT1,S2
RETURN

RESETEAR_ALTO:              ; retoma al valor de registro del contador a cero
LOAD CONT,12
RETURN

RESETEAR_BAJO:              ; retoma al valor de registro del contador a cero
LOAD CONT,03
RETURN

SUBIR:
ADD CONT,03
COMPARE CONT,15
CALL Z,RESETEAR_ALTO
RETURN
```

```
BAJAR:
SUB CONT,03
COMPARE CONT,00
CALL Z,RESETEAR_BAJO
RETURN

 PLUSDEL:                    ;establece la rutina en caso de que la interrucion sea por un aunmento del deltha
    COMPARE S2,01
    CALL Z,SUBIR
    COMPARE S2,02
    CALL Z,BAJAR
    LOAD S4,CONT
    SUB S4,01
    FETCH DELTHA3,(S4)
    SUB S4,01
    FETCH DELTHA2,(S4)
    SUB S4,01
    FETCH DELTHA1,(S4)
    RETURN
;;;;;;;;;;;;;;;;;;----------PANTALLAZO DE BIENVENIDA----------
WELCOME:
      LOAD S5, 11     ; LINEA 1 POSICION 2
      CALL LCD_cursor
      LOAD S5, character_R
      CALL LCD_write_data
      LOAD S5, character_a
```

```
CALL LCD_write_data
LOAD S5, character_d
CALL LCD_write_data
LOAD S5, character_i
CALL LCD_write_data
LOAD S5, character_o
CALL LCD_write_data
LOAD S5, character_G
CALL LCD_write_data
LOAD S5, character_I
CALL LCD_write_data
LOAD S5, character_S
CALL LCD_write_data
CALL PRINT_SPACE
CALL PRINT_SPACE
LOAD S5, character_U
CALL LCD_write_data
LOAD S5, character_I
CALL LCD_write_data
LOAD S5, character_S
CALL LCD_write_data
LOAD S5, 21      ; LINEA 1 POSICION 2
CALL LCD_cursor
LOAD S5, character_D
CALL LCD_write_data
LOAD S5, character_o
CALL LCD_write_data
```

```
        LOAD S5, character_w
        CALL LCD_write_data
        LOAD S5, character_n
        CALL LCD_write_data
        LOAD S5, character_C
        CALL LCD_write_data
        LOAD S5, character_o
        CALL LCD_write_data
        LOAD S5, character_n
        CALL LCD_write_data
        LOAD S5, character_v
        CALL LCD_write_data
        LOAD S5, character_e
        CALL LCD_write_data
        LOAD S5, character_r
        CALL LCD_write_data
        LOAD S5, character_t
        CALL LCD_write_data
        LOAD S5, character_e
        CALL LCD_write_data
        LOAD S5, character_r
        CALL LCD_write_data
RETURN

WAIT2S:LOAD S6,C8    ;ESPERA 200 VECES 20 MS
POINT:CALL delay_20ms
      SUB S6,01
```

```
    JUMP NZ,POINT
    RETURN

;;;;;-RUTINA PARA IMPRIMIR EN PANTALLA EL RESPECTIVO VALOR DEL AMPLIFICADOR----;;;
PRINT_AMP:
        LOAD S5, 22             ;LINE 1 POSICION 1
        CALL LCD_cursor
        LOAD s5, character_G        ;carga la letra I
        CALL LCD_write_data
        LOAD s5, character_a        ;carga la letra n
        CALL LCD_write_data
        LOAD s5, character_i        ;carga la letra c
        CALL LCD_write_data
        LOAD s5, character_n        ;carga la letra c
        CALL LCD_write_data
        LOAD s5, character_equals    ;carga la letra =
        CALL LCD_write_data
        LOAD S5,28                  ; linea :1 posicion: 6
        CALL LCD_cursor             ;configura la linea
        CALL IMPRIMIR
RETURN

PRINT_FLECHA_IZQ:
    LOAD S5, 10     ; LINEA 1 POSICION 2
    CALL LCD_cursor
    LOAD s5, character_LARROW       ;carga la letra c
    CALL LCD_write_data
```

```
    CALL PRINT_SPACE
RETURN

PRE_AMPLIFIER:
    CALL PRINT_FLECHA_IZQ
    LOAD S5, character_P
    CALL LCD_write_data
    LOAD S5, character_r
    CALL LCD_write_data
    LOAD S5, character_e
    CALL LCD_write_data
    LOAD S5, character_minus
    CALL LCD_write_data
    LOAD S5, character_A
    CALL LCD_write_data
    LOAD S5, character_m
    CALL LCD_write_data
    LOAD S5, character_p
    CALL LCD_write_data
    LOAD S5, character_l
    CALL LCD_write_data
    LOAD S5, character_i
    CALL LCD_write_data
    LOAD S5, character_f
    CALL LCD_write_data
    LOAD S5, character_i
    CALL LCD_write_data
```

```
    LOAD S5, character_e
    CALL LCD_write_data
    LOAD S5, character_r
    CALL LCD_write_data
    LOAD s5, character_RARROW         ;carga la letra c
    CALL LCD_write_data
RETURN
```

;;;;;;;;;;;;;;;;;;;-----------------------RUTINA PARA IMPRIMIR EN PANTALLA EL VALOR DE LA FRECUENCIA--------------;;

```
PRINT_LCD_FREC:
        LOAD s5,11
        CALL LCD_cursor
        LOAD s5, character_F          ;carga la letra F
        CALL LCD_write_data
        LOAD s5, character_o          ;carga la letra F
        CALL LCD_write_data
        LOAD s5, character_equals     ;carga la letra =
        CALL LCD_write_data
        CALL PRINT_SPACE
        CALL IMPRIMIR
```

```
        CALL PRINT_SPACE
        LOAD s5, character_H        ;carga la letra H
        CALL LCD_write_data
        LOAD s5, character_z        ;carga la letra z
        CALL LCD_write_data
        LOAD s5, character_RARROW        ;carga la letra c
        CALL LCD_write_data
    RETURN



  IMPRIMIR:
        LOAD S6,00
        LOAD S7,PLACE1                ; indica cuantas veces se va a hacer la impresion de los valores a la pantalla
    ITERE:
        FETCH S5,(S7)
        COMPARE S5,30
        CALL NZ,CAMBIAR_INDICADOR
        COMPARE S6,00
        CALL Z,PRINT_SPACE
        COMPARE S6,00
        CALL NZ, LCD_write_data
        ADD S7,01
        COMPARE S7,PLACE7
        JUMP NZ,ITERE
```

```
            FETCH S5,PLACE7
            CALL LCD_write_data
        RETURN

CAMBIAR_INDICADOR:
  LOAD S6,01
  RETURN
;;;;;;;;;;;;;;;;;;;;;;;;----------RUTINA QUE IMPRIME EN PANTALLA EL VALOR CORRESPONDIENTE DEL INCREMENTO;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PRINT_LCD_DELTHA:
            CALL BORRAR_DEC_VALUES
            CALL UBICAR_UNO
            LOAD S5,21              ; linea :1 posicion: 6
            CALL LCD_cursor
            LOAD s5, character_plus     ;carga la letra H
            CALL LCD_write_data
            LOAD s5, character_divide     ;carga la letra H
            CALL LCD_write_data
            LOAD s5, character_minus     ;carga la letra H
            CALL LCD_write_data
            LOAD S5,25              ; linea :1 posicion: 6
            CALL LCD_cursor         ;configura la linea
            CALL IMPRIMIR
            CALL PRINT_SPACE
            LOAD s5, character_H     ;carga la letra H
            CALL LCD_write_data
```

```
        LOAD s5, character_z        ;carga la letra z
        CALL LCD_write_data
RETURN

BORRAR_DEC_VALUES:                  ; hace un barrido en las 7  posiciones del scratch destinadas a almacenar los valores decimales y los llena con ceros(30)
        LOAD S1,07                  ; un ciclo de 7 repeticiones
        LOAD S2,PLACE1              ; carga el primer valor del scratch
        LOAD S3,30                  ; constante que corresponde al número cero el la pantalla LCD
    ERASE: STORE S3,(S2)
        ADD S2,01
        SUB S1,01
        JUMP NZ, ERASE
RETURN

UBICAR_UNO:
        LOAD S6,00
        LOAD S2,00
    ITERAR: ADD S2,03
        COMPARE S2,CONT
        JUMP Z,UBICAR
        ADD S6,01
        JUMP ITERAR
    UBICAR: LOAD S3,PLACE7
        SUB  S3,S6
        LOAD S4,31
        STORE S4,(S3)
```

```
    RETURN

;;RUTINA PARA LA PANTALLA QUE MANEJA EL NUMERO DE MUESTRAS TOMADAS EN LA FFT--

LETRERO_FILTER:

    CALL PRINT_FLECHA_IZQ

    LOAD S5, character_L

    CALL LCD_write_data

    LOAD S5, character_o

    CALL LCD_write_data

    LOAD S5, character_w

    CALL LCD_write_data

    LOAD S5, character_P

    CALL LCD_write_data

    LOAD S5, character_a

    CALL LCD_write_data

    LOAD S5, character_s

    CALL LCD_write_data

    LOAD S5, character_s

    CALL LCD_write_data

    CALL PRINT_SPACE

    LOAD S5, character_F

    CALL LCD_write_data

    LOAD S5, character_i

    CALL LCD_write_data

    LOAD S5, character_l

    CALL LCD_write_data

    LOAD S5, character_t
```

```
        CALL LCD_write_data
        LOAD S5, character_e
        CALL LCD_write_data
        LOAD S5, character_r
        CALL LCD_write_data
       LOAD S5, 22                ; LINEA 1 POSICION 2
        CALL LCD_cursor            ;CONFIGURA LA LINEA
        LOAD S5, character_F
        CALL LCD_write_data
        LOAD S5, character_c
        CALL LCD_write_data
        LOAD S5, character_equals
        CALL LCD_write_data
        CALL IMPRIMIR
        CALL PRINT_SPACE
        LOAD S5, character_K
        CALL LCD_write_data
        LOAD S5, character_H
        CALL LCD_write_data
        LOAD S5, character_z
        CALL LCD_write_data
RETURN
;;;;;;--RUTINA DE LA DIVISION DE LOS VALORES CONSECUTIVOS--;;;;;;;;;;;;;;
ROUTINE:
        OUTPUT S0,DIV_CONTROL_PORT      ; en este caso el registro s7 contiene el dato de control, el valor por defecto es 00 que hace que se escoja el dividendo original y que no se cargue el registro de salida
        load S2,15                      ; contador que espera 15 instrucciones del picoblaze mientras el módulo divisor ejecuta su operación.
```

```
        CALL wait_15                    ; rutina que espera las 15 instrucciones

        INPUT S3,DEC_VALUE_PORT         ; lee el residuo de la división y lo almacena en s3

        STORE S3,(S4)                   ; gurda el actual residuo en la posicion correspondiente del scratch

        SUB S4,01                       ; modifica la ubica la ubicacion del scrathch para ubicar un nuevo
residuo

        LOAD S0,07                      ; bit3: mantiene el divisor   bit2:activa el registro del cociente   bit1:
carga el cociente como nuevo dividendo

        OUTPUT S0,DIV_CONTROL_PORT      ; recarga lo anterior

        XOR S0,EN_QUOTIENT              ; deshabilita el registro del cociente

        SUB S1,01                       ; indica que se ha cumplido una iteración.

        JUMP NZ,ROUTINE

RETURN

wait_15:

          SUB S2,01

          JUMP NZ,wait_15

RETURN


LOAD_DEC_VALUE:

    LOAD S4,PLACE7                  ; carga el valor en donde se debe guardar el primer de los valores

    LOAD S0,EN_DIVIDER              ; habilita el divisor para poder hacer la operacion

    LOAD S1,07                      ; carga un contador que lleva el numero de instrucciones que se van a llevar
a cabo

    CALL ROUTINE

    LOAD S0,00                      ; bit 3: deshabilita el divisor bit2: deshabilita el registro del cociente  bit1:
reubica el mux en la posicion 0

    OUTPUT S0,DIV_CONTROL_PORT      ; confirma la orden

RETURN

;;;;;;;;;;;;;;;------RUTINAS PROPIAS DEL MANEJO DE LA PANTALLA LCD;;;;;;;;;;;;;;;;;;;;;--------

       LCD_reset: CALL delay_20ms       ;wait more that 15ms for display to be ready
```

```
        LOAD s5, 38             ;Function set
        CALL LCD_write_inst         ;write '38'
        CALL delay_20ms         ;wait >4.1ms
        CALL LCD_write_inst         ;write '38'
        CALL delay_1ms          ;wait >100us
        CALL LCD_write_inst         ;write '38' and wait >40us
        CALL LCD_write_inst         ;write 'Function Set' and wait >40us
        LOAD s5, 06             ;Entry mode
        CALL LCD_write_inst         ;write 'Entry mode' and wait >40us
        LOAD s5, 0C             ;Display control
        CALL LCD_write_inst         ;write 'Display control' and wait >40us
LCD_clear: LOAD s5, 01          ;Display clear
        CALL LCD_write_inst
        CALL delay_1ms          ;wait >1.64ms for display to clear
        CALL delay_1ms
        RETURN

delay_20ms:
        LOAD s3, 14             ;20 x 1ms = 20ms
wait_20ms: CALL delay_1ms
        SUB s3, 01
        JUMP NZ, wait_20ms
        RETURN

delay_1ms:
        LOAD s2, 19             ;25 x 40us = 1ms
 wait_1ms: CALL delay_40us
```

```
        SUB s2, 01
        JUMP NZ, wait_1ms
        RETURN

delay_40us:
        LOAD s1, 28             ;40 x 1us = 40us
wait_40us: CALL delay_1us
        SUB s1, 01
        JUMP NZ, wait_40us
        RETURN

delay_1us:
        LOAD s0, delay_1us_constant
wait_1us: SUB s0, 01
        JUMP NZ, wait_1us
        RETURN

LCD_write_inst: OUTPUT s5, LCD_output_port      ;data output
        LOAD s4, 00             ;RS=0 Instruction, RW=0 Write, E=0
        OUTPUT s4, LCD_control_port
        CALL LCD_pulse_E
        CALL delay_40us         ;wait >40us
        RETURN

LCD_pulse_E:
        XOR s4, LCD_E           ;E=1
        OUTPUT s4, LCD_control_port
```

```
        CALL delay_1us
        XOR s4, LCD_E               ;E=0
        OUTPUT s4, LCD_control_port
        RETURN

LCD_cursor:
        TEST s5, 10                 ;test for line 1
        JUMP Z, set_line2
        AND s5, 0F                  ;make address in range 80 to 8F for line 1
        OR s5, 80
        CALL LCD_write_inst         ;instruction write to set cursor
        RETURN
set_line2: AND s5, 0F               ;make address in range C0 to CF for line 2
        OR s5, C0
        CALL LCD_write_inst         ;instruction write to set cursor
        RETURN

LCD_write_data:
        OUTPUT s5, LCD_output_port  ;data output
        LOAD s4, 04                 ;RS=1 Data, RW=0 Write, E=0
        OUTPUT s4, LCD_control_port
        CALL LCD_pulse_E
        CALL delay_40us             ;wait >40us
        RETURN


PRINT_SPACE:
```

```
            LOAD S5,character_space
            CALL LCD_write_data
        RETURN
;;;;;;---RUTINA PARA CARGAR LOS VALORES GENERADOR EN EL NUCLEO DE DIVISION;;;---
  int_routine:
            DISABLE INTERRUPT
INPUT    S1,BREAK_PORT                ;lee el puerto de interrupciones
TEST     S1,EAST_BUTTON
CALL NZ, CHANGE_SELECT_RIGHT
TEST     S1,WEST_BUTTON
CALL NZ, CHANGE_SELECT_LEFT
FETCH S7,SELECT
COMPARE  S7,00                        ;CONDICION NECESARIA PARA QUE HAGA LA RUTINA DEL OSCILADOR Y NO LA DEL AMPLIFICADOR
JUMP Z, OSCILADOR
COMPARE  S7,01                        ;CONDICION NECESARIA PARA HACER LA RUTINA DEL AMPLIFICADOR
JUMP Z,  AMPLIFICADOR
COMPARE  S7,02                        ;CONDICION NECESARIA PARA QUE HAGA LA RUTINA DEL OSCILADOR Y NO LA DEL AMPLIFICADOR
JUMP Z, FILTER

OSCILADOR: INPUT S1,BREAK_PORT
        COMPARE  S1,00          ;revisa si  la interrupcion de dio por giro y el giro es hacia la izquierda
        CALL Z,  DECREMENTO     ; en caso afirmativo envia a la rutina que disminuye el valor del registro S0.
        COMPARE  S1,RLEFT       ;revisa si  la interrupcion de dio por giro y si el giro es hacia la derecha.
```

```
        CALL Z,  INCREMENTO          ; en caso afirmativo envia a la rutina que aumenta el valor del
registro S0.

        AND S1,0F

        LOAD S2,01

        TEST S1, NORTH_BUTTON        ; Revisa si la interrupcion se dio debido a que se solicita un
aumento del cambio del deltha.

        CALL NZ, PLUSDEL             ; en caso afirmativo envia a la rutina que aumenta el valor del
registro S0.

        LOAD S2,02

        TEST S1, SOUTH_BUTTON        ; Revisa si la interrupcion se dio debido a que se solicita un
aumento del cambio del deltha.

        CALL NZ, PLUSDEL

        LOAD S8,01

        RETURNI ENABLE       ; return from interrupt and enable interrupts


AMPLIFICADOR:

        INPUT S1,BREAK_PORT

        COMPARE S1,RLEFT

        CALL Z, PLUSGANANCIA

        COMPARE S1,00

        CALL Z, MINUSGANANCIA

        LOAD S8,01

        RETURNI ENABLE


      FILTER:

        INPUT S1,BREAK_PORT

        COMPARE S1,RLEFT

        CALL Z, PLUS_FC

        COMPARE S1,00
```

```
        CALL Z, MINUS_FC
        LOAD S8,01
        RETURNI ENABLE

CHANGE_SELECT_RIGHT:
FETCH S7,SELECT
COMPARE S7,02
RETURN Z
ADD S7,01
STORE S7,SELECT
RETURN
CHANGE_SELECT_LEFT:
FETCH S7,SELECT
COMPARE S7,00
RETURN Z
SUB S7,01
STORE S7,SELECT
RETURN

PLUSGANANCIA:
   FETCH S5,CURRENT_GAIN_POSITION
   COMPARE S5,GAIN100
   RETURN Z
   ADD S5,01
   STORE S5,CURRENT_GAIN_POSITION
   FETCH S6,(S5)
   STORE S6,CURRENT_GAIN
```

```
   FETCH S5, BYNARY_GAIN
   ADD S5,01
   STORE S5,BYNARY_GAIN
RETURN

MINUSGANANCIA:
   FETCH S5,CURRENT_GAIN_POSITION
   COMPARE S5,GAIN0
   RETURN Z
   SUB S5,01
   STORE S5,CURRENT_GAIN_POSITION
   FETCH S6,(S5)
   STORE S6,CURRENT_GAIN
   FETCH S5, BYNARY_GAIN
   SUB S5,01
   STORE S5,BYNARY_GAIN
RETURN

PLUS_FC:
FETCH S1,CURRENT_FILTER
COMPARE S1,04              ;MAXIMO VALOR CORRESPONDIENTE AL FILTRO DE 50KHZ
RETURN Z
ADD S1,01
STORE S1,CURRENT_FILTER
RETURN

MINUS_FC:
```

```
FETCH S1,CURRENT_FILTER
COMPARE S1,00                ;MINIMO VALOR CORRESPONDIENTE AL FILTRO DE 1 KHZ
RETURN Z
SUB S1,01
STORE S1,CURRENT_FILTER
RETURN

ADDRESS 3FF ; set interrupt vector
        JUMP int_routine     ;jump to int_routing
```