UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
ELÉCTRICA – PROCESAMIENTO DIGITAL DE SEÑALES

DESARROLLO DE UN SISTEMA ULTRASÓNICO BASADO EN FPGA

T E S I S
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
ING. MARCO GUSTAVO SERNA ESTRADA

TUTOR PRINCIPAL
DRA LUCIA MEDINA GOMEZ, FACULTAD DE CIENCIAS
COMITÉ TUTOR
DR. PABLO ROBERTO PÉREZ ALCÁZAR, FACULTAD DE INGENIERÍA
DR. SERGIO ENRIQUE SOLÍS NÁJERA, FACULTAD DE CIENCIAS
DR. BOHUMIL PSENICKA, FACULTAD DE INGENIERÍA
DR. JAIME FABIÁN VÁZQUEZ DE LA ROSA, FACULTAD DE CIENCIAS

MÉXICO, D. F. OCTUBRE 2014

JURADO ASIGNADO:

Presidente:	DR. PABLO ROBERTO PÉREZ ALCÁZAR
Secretario:	DR. SERGIO ENRIQUE SOLÍS NÁJERA
Vocal:	DRA. LUCIA MEDINA GOMEZ
1er. Suplente:	DR. BOHUMIL PSENICKA
2do. Suplente:	DR. JAIME FABIÁN VÁZQUEZ DE LA ROSA

Lugar o lugares donde se realizó la tesis: FACULTAD DE CIENCIAS.

TUTOR DE TESIS:

ING. MARCO GUSTAVO SERNA ESTRADA

_____

FIRMA

A mi Madre…

Laura Estrada Monroy

Agradecimientos

Esta tesis se la dedico a la persona que más admiro en la vida, la persona que me dio la oportunidad de existir y la que me ha llenado de las enseñanzas más trascendentales para formar la persona que soy, a mi Madre Laura Estrada Monroy. Su fuerza, ganas de hacer las cosas bien no es comparable con una persona normal, extraordinaria y excepcional las palabras más precisas para definirla, gracias por todo y por tanto Ma.

Agradezco a mi abuelita Josefina Monroy Sánchez, mujer fuerte, llena de amor y energía, el pilar de nuestra familia, su vida es, el ejemplo a seguir, sin lugar a dudas. Su generosidad y atención hacia el prójimo así como para sus seres queridos no tiene límites, persona como pocas en este mundo, un privilegiado y muy agradecido de ser su nieto. Gracias mi Yeyi.

Agradecimiento especial a la Doctora Lucia Medina Gómez por la libertad y apoyo brindado en la realización de este proyecto de vida. Agradezco a mis sinodales Dr. Pablo Roberto Pérez Alcázar, Dr. Sergio Enrique Solís Nájera, Dr. Jaime Fabián Vázquez de la Rosa y al Dr. Bohumil Psenicka, por sus comentarios y correcciones a este trabajo.

Agradezco a la Vida por darme la oportunidad invaluable de disfrutarla, crecer como persona y aprender día a día. Agradezco por todas las ausencias y presencias que me ha brindado la vida, que sin ellas definitivamente no sería la persona que soy.

Finalmente agradezco a PAPIIT-DGAPA proyecto IT118811-3 Diseño de un sistema básico de tomografía ultrasónica en transmisión para materiales altamente heterogéneos, por el apoyo económico brindado. Por ultimo agradezco a CONACYT por la beca que me proporciono para realizar los estudios de posgrado.

I | ÍNDICE GENERAL

RESUMEN.

Un sistema de ultrasonido para caracterización de materiales y tejidos biológicos está compuesto por un dispositivo capaz de generar pulsos de excitación y detectar energía reflejada por el medio a analizar mediante transductores ultrasónicos. Existen diferentes plataformas en las cuales se desarrollan estos sistemas, con en las basadas en: computadoras personales, microprocesadores, procesadores digitales de señales (DSP's), o arreglos de lógica programable en campo (FPGA's).

En esta tesis se presenta el desarrollo de un sistema de ultrasonido basado en una arquitectura configurable, tal como lo son los Arreglos de Lógica Programable en Campo (FPGA's). La ventajas principal respecto a sistemas comerciales, radica en la versatilidad de diseñar pulsos de excitación para los transductores, así como tener un sistema abierto que permita adecuar tanto la transmisión como la recepción dependiendo de la aplicación específica.

El desarrollo de este sistema se sustenta en la necesidad de adquirir datos en tiempo real, para lo cual es necesario el uso de convertidores tanto Analógico-Digital (ADC), como Digital-Analógico (DAC); además se requiere de transductores ultrasónicos, una etapa de electrónica analógica para el acondicionamiento de las señales trasmitidas y recibidas, al mismo tiempo del FPGA, en el cual se realizará todo el sistema de comunicación.

El FPGA utilizado en este proyecto es un Artix7, de Xilinx, el cual cuenta con las siguientes características: Convertidor Analógico Digital embebido, a 1Msps

de 17 canales a 12 bits, 16K de CLB, 215K celdas lógicas y 13 MB de RAM; además, este FPGA está montado sobre una tarjeta de desarrollo llamada AC701 que cuenta con distintos puertos tales como FMC, HDMI, SD, un reloj de 200MHz y 1GB de memoria RAM DDR3, que proporciona suficiente cantidad de almacenamiento.

La Implementación del sistema se realiza mediante la generación de señales en Matlab, 256 muestras con formato q12, para su posterior carga en el FPGA mediante descripción de hardware VHDL. La transmisión de las señales es activada con ayuda de los *pushbottons* de la tarjeta de desarrollo AC701 de *Xilinx*. El canal de comunicación entre el FPGA y los transductores está compuesto, por el puerto FMC, un adaptador FMC-DAC, y DAC3162EMV de *Texas Instruments*.

Los puntos a destacar de los resultados obtenidos, comparándolos con el sistema comercial *Ultratek USB-UT350T,* son: mayor flexibilidad en cuanto a trasmisión de señales, ya que cualquier señal que cumpla con 256 muestras y q12, tiene la capacidad de ser transmitida mediante nuestro sistema; cabe destacar que se cuenta con la capacidad de recepción de señales con el convertidor ADC embebido en el FPGA, otro de los principales beneficios es el bajo costo del sistema aquí desarrollado cuyo valor fue de $1,200 dólares, en contra parte el *Ultratek USB-UT350T* tiene un costo de $10,000 dólares.

INTRODUCCIÓN

INTRODUCCIÓN

A lo largo de los años el ultrasonido ha sido parte fundamental para la caracterización de cualquier tipo de materiales [1], tejidos biológicos, concreto, metal, incluso pinturas murales debido a que es un método no invasivo y no destructivo para dichos materiales. La necesidad de realizar un sistema de ultrasonido, que trabaje en tiempo real y sea portable, así como la reducción de costos, lo cual es crucial el día de hoy en cualquier sistema de ingeniería, esto es sustentado por la gran cantidad de desarrollos de ultrasonido en la investigación e industria [2].

Los sistemas de prueba de ultrasonido están compuestos por: software, etapa de amplificación, sistema de comunicación así como uno o varios transductores de ultrasonido. En cuanto al software, se utilizan distintas plataformas para realizar este proceso, computadoras personales, microprocesadores, procesadores digitales de señales (*DSP*) y arreglos de lógica programable en campo *(FPGA)*; cabe señalar que son bastantes elementos de software y hardware necesarios para el correcto funcionamiento y sincronización de todo el sistema ultrasónico.

El objetivo principal de esta tesis consiste en desarrollar un sistema de emisión/recepción ultrasónico sobre la tarjeta de Xilinx AC701, cuyo FPGA es un Artix7 XC7A200T-2FBG676C. Para esto tendremos objetivos puntuales: creación de carta ASM (Máquina de estados algorítmica), con la cual se controlarán todas las señales del sistema de comunicación, generación de señales de prueba para la excitación de transductores ultrasónicos y su posterior almacenamiento en memoria RAM del FPGA mediante VHDL; implementación del diseño sobre la tarjeta de

desarrollo AC701, acondicionamiento de las señales mediante ADC, DAC y amplificadores, finalmente la sincronización del conjunto de elementos (FPGA, DAC, Amplificadores, Transductores, ADC, FPGA, memoria RAM) para la emisión y recepción de las ondas de ultrasonido con la finalidad de que el sistema funcione en tiempo real.

En el capítulo 1 se mencionan los diferentes sistemas ultrasónicos y se aborda el *pulser* ultrasónico.

En el capítulo 2, se hace una breve descripción de las representaciones numéricas en punto fijo y punto flotante las cuales son fundamentales para el manejo digital de señales, así como la descripción de los diferentes tipos de hardware usados para el procesamiento digital de señales.

En el capítulo 3 se describe con detalle el diseño del sistema de ultrasonido desarrollado, las descripciones de hardware y software requeridos para el desarrollo del sistema.

En el capítulo 4 se exponen las simulaciones, la implementación, las comparaciones del sistema desarrollado frente a sistemas comerciales y los resultados obtenidos en la investigación. Finalmente las conclusiones que se obtuvieron en la realización de este proyecto y el trabajo futuro.

Capítulo 1

SISTEMAS DE ULTRASONIDO

CAPÍTULO 1

SISTEMAS DE ULTRASONIDO

En este capítulo se describirán diversos sistemas de ultrasonido.

El ultrasonido es una señal mecánica cuya frecuencia es mayor al rango audible por el ser humano, es decir, superior a 20 KHz. Su uso médico fue desarrollado por primera vez en 1949 por el Dr. George D Ludwing realizando estudios en perros y cerdos [1]. Este estudio dio origen al desarrollo de ingeniería basada en ultrasonido para la salud [2]. Desde su uso para el tratamiento de tejidos o lesiones hasta la generación de imágenes médicas. El uso del ultrasonido como ensayo no destructivo no se limita a la medicina, también tiene distintas aplicaciones en la caracterización, monitoreo y control de calidad en procesos industriales, incluso como técnica de detección y localización de daños de bienes e inmuebles para su posterior restauración, debido a que se pueden evaluar materiales sin afectar las propiedades mecánicas de los mismos.

La estructura general de un sistema de ultrasonido está compuesta por tres grandes módulos (ver fig 1.1).

El modulo Digital es el objetivo principal de este trabajo, por lo que será descrito a detalle en el capítulo 2.

- Transductores
- Pulsador/Receptor
- Modulo Digital

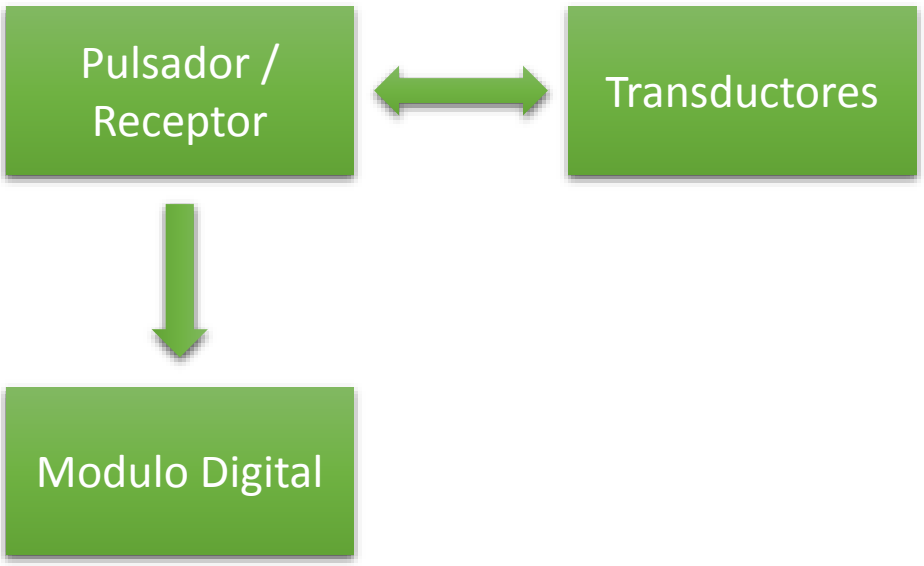


Fig. 1.1 Estructura básica de Pulsador y Receptor Ultrasónico

1.1 | PULSADOR/RECEPTOR

Es el encargado de la generación y recepción de señales eléctricas hacia y desde los transductores. Las tareas a su cargo se pueden realizar considerando tres etapas:

- Conversión digital analógica
- Acondicionamiento.
- Conversión analógica digital

1.1.1 Conversión Digital Analógica.

Es una etapa de suma importancia debido a que es la encargada de convertir

la señal digital (bits), en una señal analógica, con ayuda del circuito integrado llamado Convertidor Digital Analógico. En la fig. 1.2 se muestra la Característica del DAC ideal.

En la fig. 1.2 podemos apreciar el comportamiento de un DAC ideal, el cual a toda respuesta binaria da una salida de voltaje o corriente. En la práctica existen algunas desviaciones de la ideal debido a errores de offset, errores de ganancia o no linealidades de la característica de entrada-salida [6]. Para corregir este tipo de desviaciones es necesario agregar dos bloques: el primero consta de un muestreo-retención, el segundo de un filtro paso bajo suavizado.

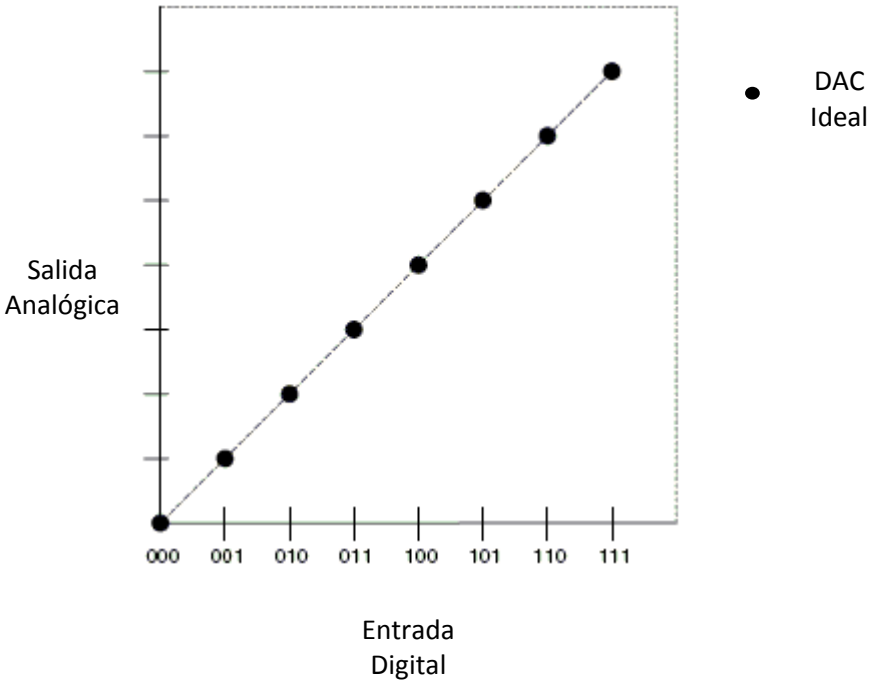


Fig. 1.2 Característica del DAC ideal.

El bloque de muestreo y retención (S/H) se necesita debido a una característica fundamental del DAC, llamada tiempo de asentamiento. Este tiempo es el necesario para que el circuito integrado mantenga el dato una vez que el valor binario es mandado a la entrada del DAC.

El fenómeno conocido como *glitch*, es generado por un cambio abrupto en los bits de los valores de entrada, con respecto al valor anterior. Este problema es resuelto por el bloque S/H (Fig. 1.3), y por ello se le conoce como *"deglitcher"*. La tarea del S/H es mantener la salida del DAC constante hasta que la nueva muestra se encuentre en la salida del DAC [6].

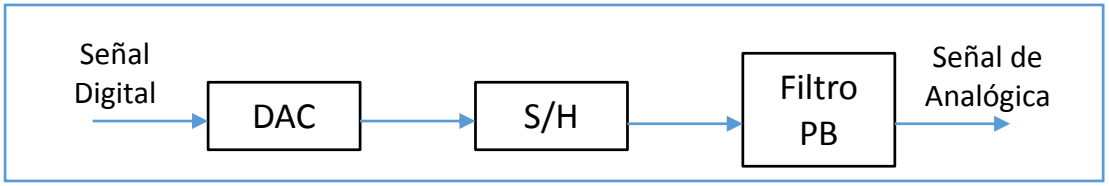


Fig. 1.3 Bloques DAC real.

Este proceso de Muestreo-Retención genera altas frecuencias a la salida del bloque S/H, por lo que es necesario implementar un Filtro Paso Bajas para eliminar el *aliasing* de componentes frecuenciales no deseadas.

1.1.2 Acondicionamiento

La etapa de acondicionamiento de señales en los sistemas de ultrasonido consta principalmente de dos partes; la primera se encuentra a la salida del convertidor

digital analógico la cual consiste en elevar el voltaje para la correcta excitación de los transductores de ultrasonido.

La segunda de acondicionamiento se coloca a la entrada del convertidor Analógico Digital, la cual es de suma importancia debido a que para el caso de la modalidad pulso eco se requiere de un *switcheo*.

1.1.3 Conversión Analógica Digital

Una vez que la señal de ultrasonido produce un eco, después de golpear con una superficie, es necesario captar toda esa información analógica, es por eso que debemos utilizar circuitos integrados de conversión analógica digital *ADC*. El objetivo de estos circuitos es convertir una señal analógica, por consiguiente continua en el tiempo, en una señal digital representada por un número finito de bits.

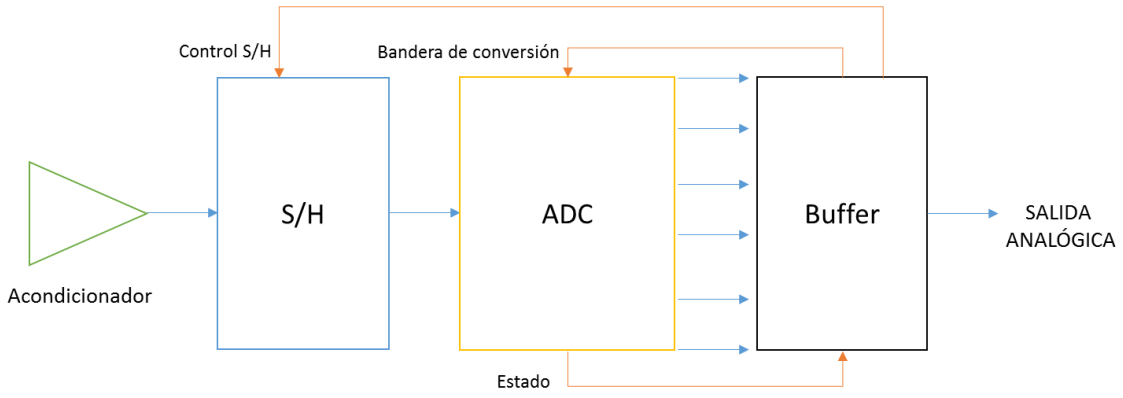


Fig. 1.4 Diagrama a bloques de un Convertidor Analógico Digital

Como se muestra en la fig. 1.4, para el funcionamiento de un ADC será necesario el uso de un circuito de muestro y retención, S/H, muy similar al descrito en el punto 1.3, solo cambia la posición en la cual se utiliza, ya que es un circuito que trabaja únicamente con señales analógicas y se coloca a la entrada del ADC. Su control es realizado mediante una señal digital, su principal objetivo es muestrear y retener la señal que sale del bloque acondicionador hasta que el ADC lo procesa para obtener su representación digital.

1.5.1 Cuantificación y Codificación

La función principal del ADC es convertir un rango continuo de amplitudes en un grupo de palabras representadas con bits, la cual involucra un proceso de cuantificación y uno de codificación. La cuantificación es un proceso no lineal y no invertible que asigna a una amplitud determinada $x(n) = x(nT)$, en el instante $t = nT$, una amplitud x_k, tomada de un conjunto finito de valores [6].

La función de codificación de un ADC consta en asignar un valor binario a cada nivel de cuantificación, existen errores en cuanto a la codificación se refiere, uno de ellos es el error debido al offset, otro se puede presentar debido a un error en el factor de escala o un error debido a la linealidad, dado por las diferencias entre los valores de transición que no son iguales o varian de manera uniforme [6].

1.2 | TRANSDUCTORES

Son los encargados de la conversión de energía eléctrica a energía mecánica y viceversa [19]. Para realizar dicha conversión, la construcción de estos se basa, generalmente en dos principios físicos por los cuales son fabricados los transductores de ultrasonido: el efecto piezoeléctrico y el magneto-resistivo. A continuación se explican estos dos efectos

Por otro lado, para obtener la información del medio en estudio existen dos modalidades principales de radiación y adquisición: el pulso eco y la transmisión directa.

1.2.1 Efecto Piezoeléctrico

El efecto piezoeléctrico, consiste en convertir deformaciones mecánicas en pulsos eléctricos fue descubierto en 1880, y en 1881 el efecto inverso fig. 1.5.

Las propiedades de un cristal dependerán de cómo sea cortado, debido a esto, es que los cristales vibran a cierta frecuencia, la cual estará definida por la relación que existe entre la longitud de onda y el espesor.

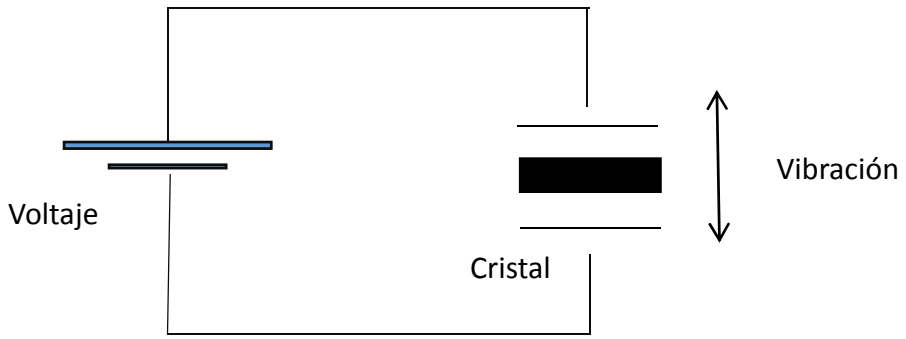


Fig. 1.5 Efecto Piezoeléctrico

1.2.2 Efecto Magneto-restrictivo

Como su definición lo dice, es un material que al momento de aplicar voltaje sufre una deformación mecánica, este tipo de materiales se llaman ferro eléctricos [18], cuyas moléculas se encuentran al azar en su estado de reposo, una vez que se aplica una diferencia de potencial se ordenan en dirección del campo magnético, esto hace que el espesor de dicho material aumente y disminuye al mismo tiempo que el voltaje lo hace, al realizar este proceso varias ocasiones se generan las vibraciones.

Los materiales ferro eléctricos no alcanzan frecuencias tan altas por sí mismos, es por esto que se debe realizar un proceso químico para que se logren comportar como piezoeléctricos.

1.2.3 Pulso Eco

El pulso eco, como se mencionó a grandes rasgos anteriormente, consiste en la trasmisión y recepción de señales de ultrasonido utilizando sólo un transductor. La medición consistirá en la amplitud de la señal recibida y el tiempo de vuelo entre la onda enviada y recibida, como se muestra en la fig. 1.6. El despliegue grafico muestra dos ecos, uno referente a la presencia de una fractura y otro a la superficie opuesta al transductor, indicando el espesor del material inspeccionado [24].

Cabe mencionar que la técnica de pulso eco es utilizada cuando se tiene acceso a solo una cara del material.

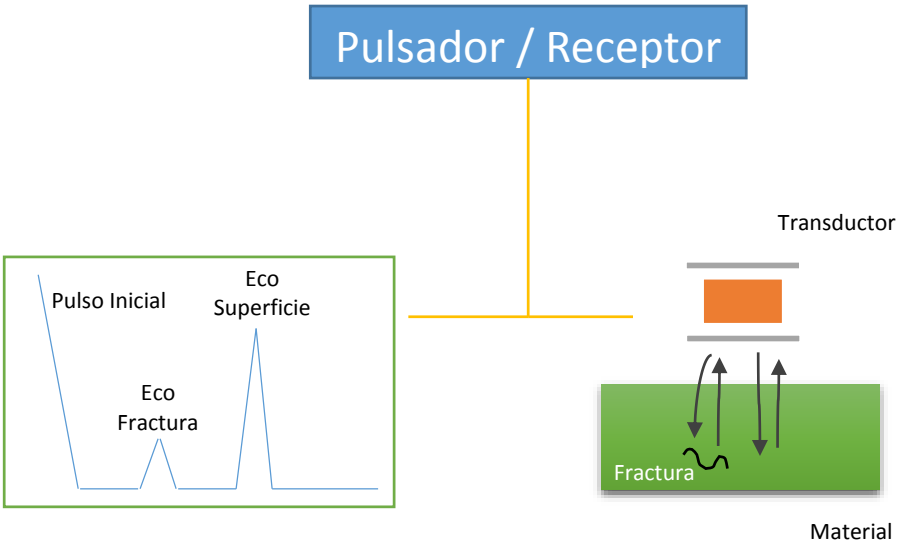


Fig. 1.6 Pulso Eco

1.2.4 Transmisión Directa.

La transmisión directa consiste en usar dos transductores (ver fig.1.7): uno que fungirá como transmisor y el otro como receptor. Esta técnica nos proporciona facilidad en cuanto a la etapa de acondicionamiento de las señales, debido a que no hay que preocuparse por la conmutación de la electrónica analógica, correspondiente a la emisión y recepción.

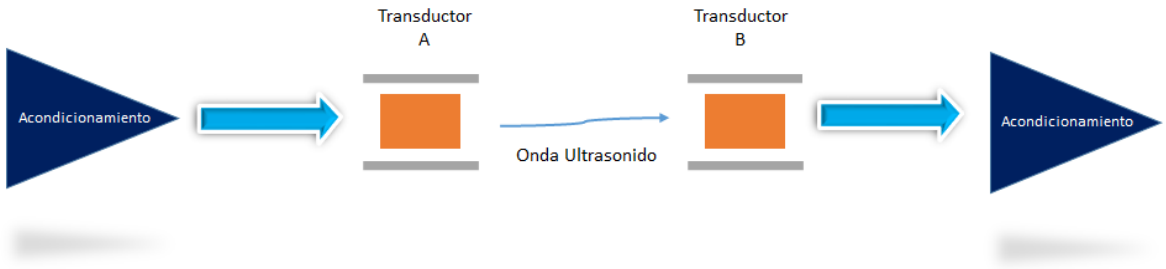


Fig. 1.7 Configuración de Transmisión Directa en transductores.

Capítulo 2

HARDWARE PARA PROCESAMIENTO DIGITAL DE SEÑALES

CAPÍTULO 2

HARDWARE PARA PROCESAMIENTO DIGITAL DE SEÑALES

En este capítulo se describirán los diferentes tipos de hardware usados para el procesamiento digital de señales.

2.1 | HARDWARE PARA PROCESAMIENTO DIGITAL DE SEÑALES

Existen diferentes opciones a elegir para realizar el procesamiento de señales en cuanto al aspecto digital se refiere, desde una PC pasando por Microprocesadores, Procesadores Digital de Señales (DSP's), Circuitos Integrados de Aplicación Especifica (ASIC's), hasta terminar con los Arreglos de Lógica Programable en Campo (FPGA's).

2.1.1. Formatos Numéricos

Es de suma importancia tener presente las opciones que existen para realizar el procesamiento más adecuado de la señal, ya sea mediante punto fijo o punto flotante en sus diferentes estándares.

Los números, en el aspecto digital, se representan con grupos de bits los cuales sólo pueden tomar valores 0 o 1. El conjunto de bits que se necesitan pare representar un valor se le conoce como palabra binaria y si la longitud estará definida por el número de bits que se empleen. Es necesario poder expresar valores tanto positivos como negativos, para lo cual se utiliza el formato de *magnitud signada*. Este usa bit más significativo, *MSB*, de la palabra para establecer el signo, es decir si, el *MSB* de una palabra es 0 implicará un número positivo, por el contrario al ser 1 sabremos que se trata de un número negativo.

2.1.1.1 Punto Fijo.

El formato de magnitud signada no es el más adecuado cuando se trata de realizar operaciones, es por esto que es conveniente el uso de un formato conocido como *complemento a dos* [3], dicho formato es más ventajoso para realizar sumas y restas en hardware de una manera menos compleja.

El rango de valores que se puede representar con este formato está definido por el número de bits con el que se forma el valor, denotado con la letra N, va desde $-2^{(N-1)}$ hasta $2^{(N-1)} - 1$, donde para obtener la representación se tomaran los valores que tienen 1 y se sumaran en potencia de dos de acuerdo a la posición en la que se encuentran. En caso de que sea un número negativo, se complementa a 2 y se suma a 1 con el mismo número de bits.

Es fundamental la representación de números decimales, y para esto se puede utilizar el formato en punto fijo, en el cual ciertos bits de la longitud de palabra se asignan para la parte decimal y el restante para la parte entera; esto nos limitara

en cuanto a la resolución de los valores a representar, es decir, entre más bits para la parte entera tendremos valores menos exactos en la parte decimal, esta elección debe realizarse de acuerdo a la aplicación a desarrollar tomando en cuenta la importancia en la precisión de los datos.

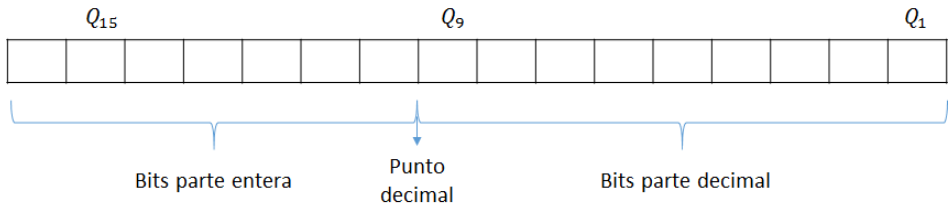


Fig. 2.1 Representación Punto Fijo en Q_9

En la fig. 2.1 se establece un ejemplo de la representación en punto fijo, incluyendo parte entera y parte decimal, en este caso se le llama Q_9, debido a que se asignan 9 bits para la parte decimal y el resto queda para la parte entera.

2.1.1.2 Punto Flotante.

En cuanto al punto flotante, debemos decir que tiene como objetivo la representación de un rango de valores mucho más grande y preciso que el punto fijo.

En punto flotante existen diversos tipos de formato numérico, los cuales son desarrollados por distintas compañías de hardware y software, tales como Microsoft y Texas Instruments; sin embargo, existe un estándar que es el establecido por la IEEE [4] y es el que se usa de manera general en las operaciones de punto flotante.

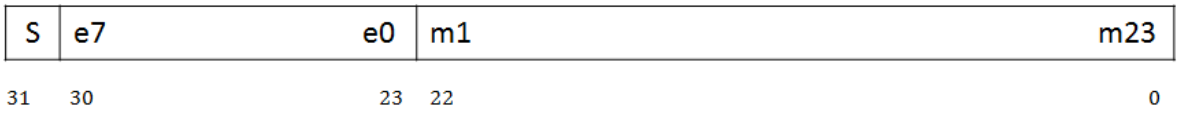


Fig. 2.2 Representación Punto Flotante

Donde:

Bit 31: El signo de la mantisa, s=0 para positivo, s=1 para negativo.

Bits 30–23: El exponente de 8 bits binario $0 \leq e \leq 255$

Bits 22–0: La mantisa de 23 bits, m, expresada como una fracción binaria, incluye un 1 binario como el bit más significativo: m=1 . $m_1, m_2,..., m_{23}$

A continuación describiremos un ejemplo para un mayor entendimiento de este formato. Se desea representar el número -423.17, que es un numero negativo, por lo tanto el primer bit de signo será 1. A continuación convertimos el numero 423.17 a binario sacamos el exponente 1000 1001 y finalmente la mantisa de 23 bits será 101 001 1001 0101 1100 0010. El valor al final de la conversión queda como se muestra en la fig 2.3.

1	1000 1001	101 001 1001 0101 1100 0010

Fig. 2.3 Conversión a punto Flotante

Una ventaja muy importante que genera el uso de este formato numérico, es mayor precisión a la hora de realizar operaciones; sin embargo, tendrá un alto costo

en tiempo de procesamiento. Al momento de realizar sumas o restas con punto flotante existe una gran desventaja, ya que hay que realizar varias operaciones complejas para obtener el resultado, lo que nos hará considerar el uso de punto fijo, en el cual, la implementación a nivel de computo es más sencilla; además la complejidad para implementar funciones simples u operaciones es mayor en punto flotante, así que, para la elección de este formato numérico será importante evaluar la exactitud requerida en los datos a trabajar.

2.2 | MICROPROCESADORES

Los microprocesadores son circuitos integrados surgidos al inicio de la década de los 70´s, por la necesidad de procesar gran cantidad de tareas que de manera manual, el ser humano no puede realizar.

En la fig. 2.4 se presenta la arquitectura Von Neumann de un Microprocesador. Se distinguen 4 pasos para lograr ejecutar una sola operación, lo cual nos habla que mínimo requerimos 4 ciclos de reloj para obtener un solo resultado, generando una larga fila de instrucciones, que para poder ser ejecutadas deberemos esperar a que finalice la que está siendo procesada.

El funcionamiento de un microprocesador con arquitectura Von Neumann se describe a continuación:

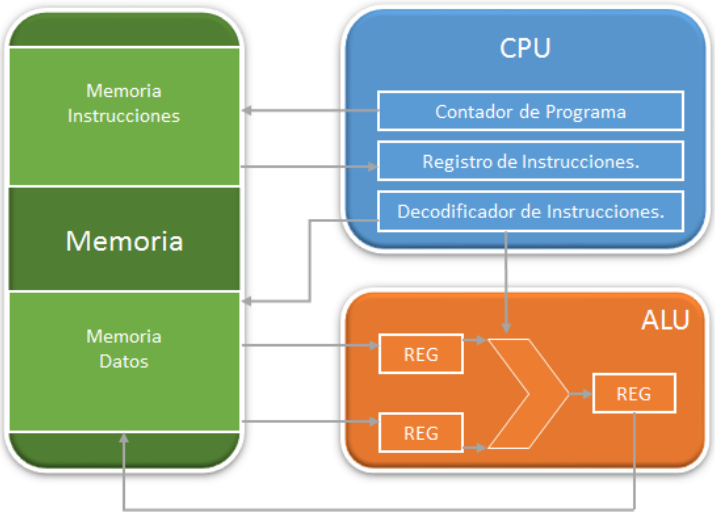


Fig. 2.4 Arquitectura Von Neumann Microprocesador

- ✓ Inicialización de contador de programa y búsqueda de la instrucción solicitada en la memoria de instrucciones.
- ✓ La Instrucción llega al registro de instrucciones en la Unidad Central de Procesos (CPU), para su decodificación.
- ✓ Se toman los operandos requeridos desde la memoria de datos y subsecuentemente se mandan a los registros en la Unidad Aritmética Lógica (ALU) para su operación.
- ✓ Finalmente el resultado se guardará en un registro y se enviará a la memoria de datos.

La programación de este tipo de circuitos integrados se puede realizar con lenguaje de alto nivel, por ejemplo C o C++, lo cual hace necesario conocer la arquitectura o la lista de instrucciones a detalle para poder implementar algoritmos o funciones; sin embargo, al querer realizar un proceso lo más eficiente posible, será necesario el uso del lenguaje ensamblador, por lo que se deberá aprender todas las

instrucciones por cada microprocesador que se desee trabajar.

2.3 | PROCESADORES DIGITALES DE SEÑALES

A raíz de la necesidad de implementar una mayor cantidad de operaciones en el menor tiempo posible e incluso buscando el procesamiento en tiempo real, surgieron en los años 80`s arquitecturas con mejor rendimiento que los microprocesadores, a lo que se les llamo procesadores digitales de señales o DSP's, por sus siglas en inglés. Estos constan de bloques de hardware específico que ayudan a una implementación más sencilla de algoritmos propios del procesamiento digital de señales, como la función MAC (Multiplica y Acumula).

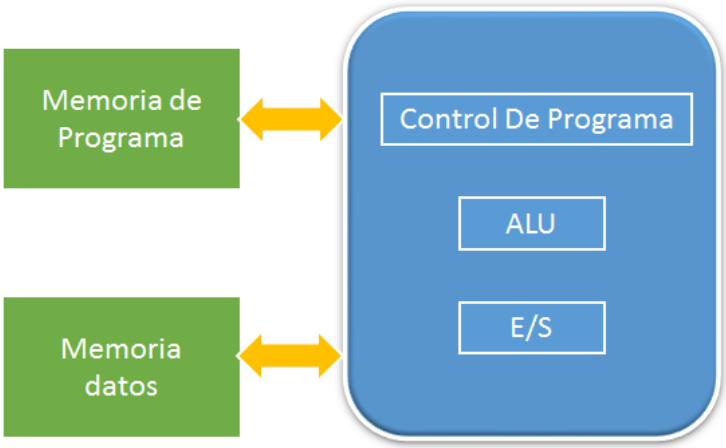


Fig. 2.5 Arquitectura Harvard DSP's.

Los DSP's tienen consigo mejoras significativas con respecto a los microprocesadores, algunas de ellas se presentan enseguida:

- ✓ Aumento del tamaño de los buses, fig. 2.7, lo que permite mayor transferencia de información en menor tiempo.

- ✓ Direccionamiento indirecto, es decir, podemos direccionar varias localidades de memoria al mismo tiempo, sin la necesidad de indicar la dirección exacta del dato que requerimos.

- ✓ Otra gran ventaja que tenemos con los DSP's es la inclusión de instrucciones especiales para diferentes tipos de algoritmos de Procesamiento, ya sea de filtrado, comunicaciones, corrección de errores, etc.

- ✓ La ventaja más sustancial en los DSP's, es que se efectúa una técnica llamada Pipeline, la cual, separa en más instrucciones el ciclo básico de ejecución (busca instrucción, codifica instrucción, busca operados, ejecuta). Existen arquitecturas con niveles de pipeline superiores a 20, esto es una gran ventaja a partir del ciclo de reloj número 20, ya que nos arrojara un resultado cada ciclo de reloj y no cada 4 como con los microprocesadores en el mejor de los casos.

Algunas limitantes que podemos tener en el uso de los DSP's es que tienen un arquitectura rígida y hay que adaptar los algoritmos para cada estructura dada, esto hace que nuestra implementación no sea flexible y dependamos del DSP que estemos usando.

2.4 | HARDWARE CON ARQUITECTURA CONFIGURABLE

2.4.1 Circuitos Integrados de aplicación específica (ASIC's).

Este tipo de circuitos integrados cumplen con la máxima eficiencia, en velocidad, como en consumo mínimo de silicio y por lo tanto de energía, son ideales para procesamiento en tiempo real y portabilidad, capaces de realizar una gran cantidad de procesos con el mejor resultado.

Podemos encontrar dos tipos de ASIC's, el primero denominado de *celda estándar*, el cual ofrece librerías de funciones para poder implementar de una manera eficiente los algoritmos deseados, los hay con microprocesadores embebidos, memorias, etc. Por otra parte, existen los llamados *estructurados,* estos contienen memoria RAM, registros y se basan en multiplexores, a diferencia de los de celda estándar, estos son más grandes y tienen mayor consumo de potencia.

Hasta el momento podrían parecer la mejor opción para desarrollar un sistema digital; sin embargo tienen limitaciones de peso, por las cuales solo se eligen para aplicaciones sumamente conocidas o una vez que se tenga el diseño final al cual ya no se le harán pruebas y modificaciones para que así se puedan producir en serie. El proceso de elaboración de los ASIC's es altamente costoso en comparación con las otras opciones, debido a que es la construcción de un circuito integrado única y exclusivamente para cierta aplicación, es decir, que una vez configurados ya no pueden ser reprogramados en un laboratorio cualquiera y por consiguiente es una

limitante de peso para realizar pruebas o mejoras de diversos diseños.

2.4.2 Dispositivos Lógicos Programables Complejos (CPLD's).

Circuitos integrados que están formados por compuertas lógicas y matrices de interconexiones entre ellas, con la finalidad de un uso más eficiente del silicio, derivando en un ahorro de energía significativo comparado con otros circuitos integrados. Son de bajo costo, una ventaja importante sobre otras opciones, reconfigurables, debido a que tienen celdas EPROM o EEPROM. Los CPLD's más complejos tienen incorporados bloques de memoria RAM, sin embargo, a pesar de que tienen buen número de compuertas lógicas, es casi insignificante para su uso en algoritmos o funciones complejas propias del Procesamiento Digital de Señales.

2.5| ARREGLOS DE LÓGICA PROGRAMABLE EN CAMPO (FPGA)

Los Arreglos de Lógica Programable en Campo o FPGA por sus siglas en inglés (Field Programmable Gate Array) son circuitos integrados formados por compuertas lógicas, las cuales son configuradas y pueden ser reconfiguradas posteriormente, mediante descripción de hardware, con los lenguajes VHDL y Verilog. Los FPGA's surgieron con la idea de mejorar las opciones existentes, tomando lo mejor de los ASIC's y los CPLD's; en otras palabras, la capacidad de realizar operaciones complejas y tener la facultad de ser reconfigurarlos de una forma flexible.

En el año 2000 comenzaron a aparecer FPGA's con alto desempeño para el procesamiento de funciones complejas, lo cual creo un auge para su uso en el procesamiento de gran cantidad de información. Es posible darle un enfoque completamente diferente al de los microprocesadores y DSP's, esto quiere decir, que en vez de tener una unidad ejecutando filas de procesos, podemos generar pequeñas unidades de procesos, como consecuencia nos introduce a procesamiento en paralelo, aquí radica una importante ventaja sobre las arquitecturas complejas de multinucleo y multibuses, además de reducción de costo y potencia, obteniendo una mayor eficiencia siempre y cuando el diseño sea el adecuado.

A diferencia de los CPLD's, los FPGA's están formados por una gran cantidad de bloques lógicos rodeados de conexiones fig. 2.6, dichos bloques tienen un comportamiento similar a interruptores, permitiendo la realización de cualquier función que cubra las necesidades del diseñador. Esta cualidad se convierte en una ventaja debido a que no existe limitantes de hardware como las opciones expuestas anteriormente, sin embargo, esto conlleva en un problema: alto tiempo de diseño. La gran ventaja de los FPGA's es que el diseñador es libre de implementar cualquier función que requiera para plantear soluciones a diferentes algoritmos, también esto puede llegar a ser una desventaja debido a que hay que implementar cada una de las funciones que se requieran 'a mano'; multiplicar y acumular, hacer desplazamientos, filtros, ciclos , etc.

2.5.1. Tecnologías de Configuración en FPGA's

Existen diferentes tecnologías para la configuración del FPGA, dependiendo de la tecnología de cada uno de ellos. La primera que se abordará es la que se conoce

como *antifusible*, tecnología que está compuesta de celdas que muestran alta impedancia cuando no se están programadas, al introducir alto voltaje en estas celdas tienden a comportarse como circuito cerrado y por lo tanto conducen la señal ya que la impedancia se hace nula, la desventaja de este método es que solo pueden ser usados una sola vez, por esta razón hace que sea una técnica no apta para pruebas y reconfiguraciones.

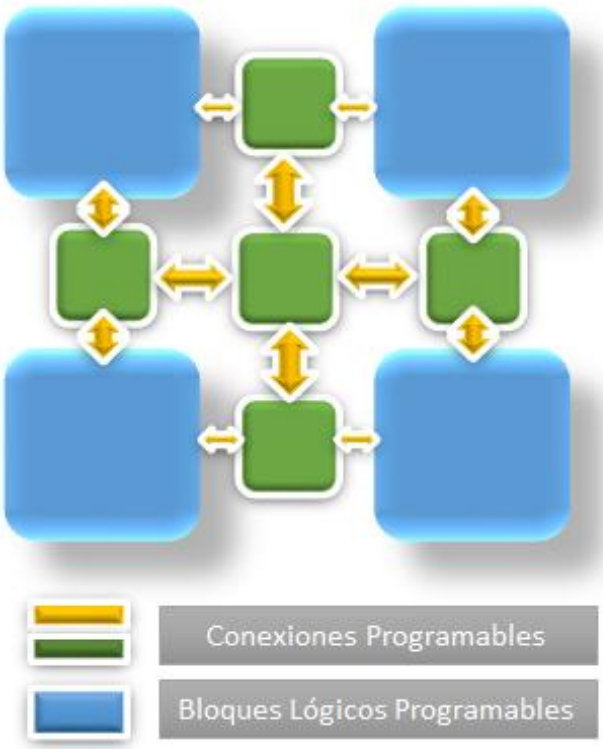


Fig. 2.6 Arquitectura Básica FPGA

Por otro lado, los FPGA's basados en memoria RAM específicamente en la memoria estática, poseen una gran ventaja debido a que se pueden configurar las veces que sea necesario, evitando la limitante que se presenta en la los FPGA's basados en el *antifusible*, sin embargo, es sabido que la memoria RAM es volátil, es decir, una vez que permanezca sin voltaje se perderá el valor o configuración cargada,

significa la necesidad de volver cargar la configuración cada que se desconecte.

Actualmente existen FPGA's con memorias EEPROM y FLASH lo cual nos da la versatilidad de las SRAM pero con la posibilidad de no perder los datos almacenados en ellas una vez que se queden sin voltaje, a su vez las celdas son más pequeñas, por consiguiente generan dispositivos más reducidos en tamaño.

2.5.2. Arquitecturas FPGA's

Existen dos tipos de arquitecturas de FPGA's, una de ellas es mediante multiplexores, como ventaja principal de estos se puede decir que son veloces para la transferencia de señales, debido a que el retardo entre bloques es pequeño, a pesar de ello, pierden eficiencia en funciones complejas, la razón: los retrasos entre bloques se van sumando a lo largo del proceso, hasta crear un tiempo largo que incluso puede llegar a ser de uno o varios ciclos de reloj, por lo cual, será un problema grave en aplicaciones de tiempo real.

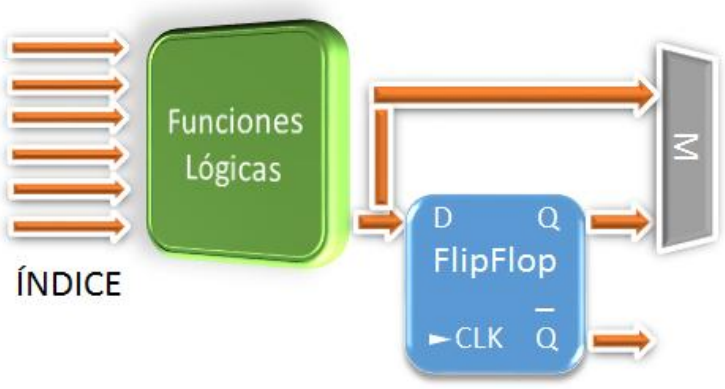


Fig. 2.7 Configuración básica LUT

La otra alternativa que existe es con tablas de consulta o LUT's fig 2.7, por sus siglas en inglés (Look-up Tables), son sumamente eficientes en cuanto operaciones aritméticas se refiere, contienen un índice como señal de entrada, lo cual optimiza el acceso a la función lógica deseada. Esta alternativa es altamente recomendada para diseño de funciones complejas, ya que los tiempos de retardo son mucho más cortos que si se trabajara con la arquitectura de multiplexores.

2.5.3. Hardware para Procesamiento Digital de Señales

Con el paso del tiempo a los FPGA's se les ha ido incorporando hardware para un mejor procesamiento de señales sobre todo con la finalidad de reducir tiempos de diseño y realizar funciones de manera óptima [9]. A continuación se enlista algunos de ellos:

- Acarreo Rápido

Son elementos que nos permiten calcular los acarreos en operaciones aritméticas, sin la necesidad de esperar el resultado de las operaciones, esto nos permite realizar operaciones en el mismo tiempo sin que la longitud de los operando sea una limitante.

- Hardware de Multiplicación o MAC

Como sabemos las multiplicaciones en el procesamiento de señales son fundamentales, es por esto que se han ido incorporando multiplicadores de diferente longitud de bits esto ha sido una gran ventaja debido a que el diseño de los

multiplicadores en VHDL o Verilog implicaba tiempo crucial en el diseño, también se han incluido bloques de Multiplicación y acumulación, de gran ayuda para la implementación de filtros.

- Microprocesadores

Cada vez es más común encontrar microprocesadores embebidos en FPGAs, esto por la necesidad de realizar operaciones complejas, sin embargo el costo de este tipo de encapsulados es elevado.

2.5.4. VHDL y Verilog

Existen diferentes lenguajes para la descripción de hardware, los más usados son VHDL y Verilog, estos lenguajes tienen un enfoque diferente a los lenguajes de programación que conocemos para sistemas digitales, tales como C, C++ o ensamblador que son secuenciales en contraste a las sentencias en HDL que son inherentes y concurrentes, en otras palabras existe paralelismo en los procesos.

Los lenguajes de descripción de hardware constan de un conjunto de expresiones que ayudan a describir el comportamiento del circuito, la ventaja que da su uso, es que se depende de una arquitectura fija para poder implementar una función o algoritmo, a diferencia de trabajar con microprocesadores o DSP's donde el conocimiento de cada parte del circuito integrado así como su set de instrucciones es fundamental para el desarrollo de un sistema.

El VHDL se fundamenta un estándar realizado por la IEEE (Institute of Electrical and Electronic Engineers) llamado IEEE 1076, el cual está continuamente actualizado con nuevas sentencias y mejoras para facilitar su uso, es un lenguaje que constantemente se está mejorando. Verilog utiliza una estructura similar al lenguaje de programación C.

2.5.5. Limitantes

El mejor desempeño siempre será dado por los ASIC's ya que minimizan el consumo de potencia y optimizan el uso de recursos de hardware, es aquí donde valdrá realizar una evaluación sobre costo beneficio en la elección de alguna de estas dos opciones. En cuanto a la necesidad de implementar algoritmos con alta complejidad los FPGA's requerirán un mayor tiempo de diseño, mientras que en DSP's el tiempo de diseño estará en función del conocimiento de la arquitectura.

Capítulo 3

DISEÑO DEL SISTEMA DE ULTRASONIDO.

CAPÍTULO 3

DISEÑO DEL SISTEMA DE ULTRASONIDO

En este capítulo se planteará el diseño del sistema de ultrasonido, descripción del hardware y software requerido para el desarrollo del mismo.

3.1 | DISEÑO DEL SISTEMA ULTRASONICO

El desarrollo consiste en realizar una Sistema completo de Ultrasonido, en el cual se realizara el envío y recepción de las señales sobre un FPGA, es decir un sistema de comunicación, esto mediante software y hardware, para el proceso se utilizará la tarjeta de desarrollo AC701 de *Xilinx*, que cuenta con convertidor Analógico-Digital (ADC) embebido, compuesto por 17 canales de 12 bits a 1 MSPs, además de un convertidor Digital-Analógico (DAC) 3162EVM de *Texas Instruments*, electrónica analógica para etapas de acondicionamiento de las señales, así como de diversas herramientas para medición en laboratorio, por lo que la metodología a seguir es la que se describe a continuación.

- Creación de carta ASM (Máquina de estados algorítmica), con la cual se controlarán todas las señales del sistema de comunicación.
- Simulación en el software de *Xilinx ISM* para la verificación del funcionamiento correcto de cada una de las señales de control correspondientes para la trasmisión y recepción.

- Generación de señales de prueba para la excitación de transductores ultrasónicos y su posterior almacenamiento en memoria RAM del FPGA mediante VHDL.
- Ruteo de Señales a través del canal de comunicación.
- Implementación del diseño sobre la tarjeta de desarrollo AC701.
- Acondicionamiento de las señales mediante ADC, DAC y amplificadores.
- Sincronización del conjunto de elementos (FPGA, DAC, Amplificadores, Transductores, ADC, FPGA, memoria RAM) para la emisión y recepción de las ondas de ultrasonido con la finalidad de que el sistema funcione en tiempo real.

El diagrama del diseño se muestra en la fig. 3.1.

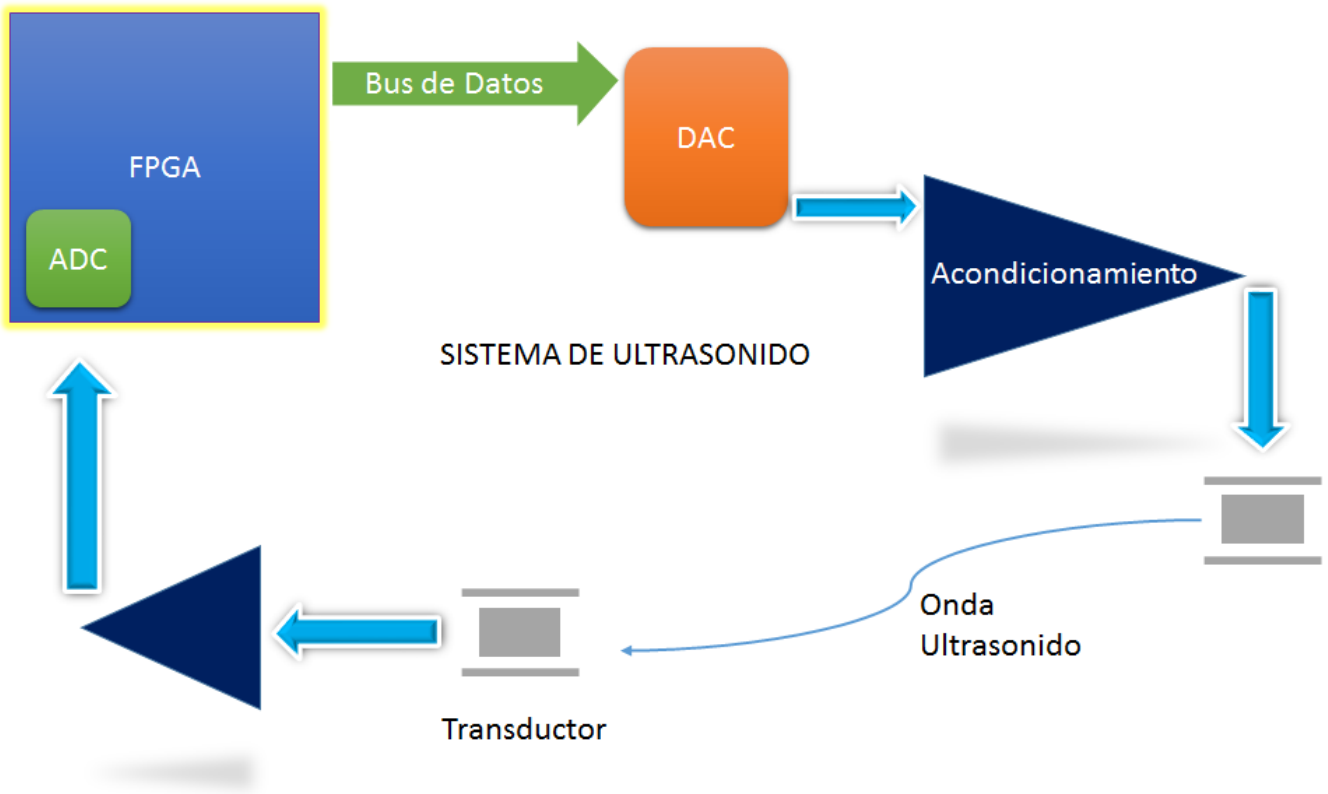


Fig. 3.1 Diseño Sistema Ultrasonido.

En la fig. 3.1 podemos observar que a través del FPGA se transmitirán las señales en forma digital con un bus de 12 bits, los cuales llegarán al DAC con sus respectivas señales de reloj para realizar la conversión analógica, posteriormente la señal analógica arrojada por el DAC, llegará a una etapa de acondicionamiento, con el objetivo de una excitación adecuada del transductor de emisión. Posteriormente se transmitirá la señal en el medio, aquí se debe tomar en cuenta el tiempo de vuelo de la señal de ultrasonido, el cual consiste en el tiempo que tarda en llegar la señal del transductor de emisión al de recepción, una vez que la señal llegue a este último, el cual fungirá como receptor, esta será enviada a otra etapa de acondicionamiento para finalmente introducir nuestra información para su digitalización en el ADC embebido en el FPGA.

3.2 | TARJETA DE DESARROLLO XILINX AC701

Los FPGA's son circuitos integrados con una gran cantidad de compuertas lógicas y con buena cantidad de pines de entrada y salida, por esta razón son circuitos de montaje superficial para reducir tamaño y mejor uso del espacio, los fabricantes como *Xilinx Altera Lattice*, etc, montan sus encapsulados en tarjetas de desarrollo donde se incluyen distintos puertos. Hay una variedad de elementos útiles en las tarjetas para realizar diferentes tipos de pruebas, desde LED's para verificar el funcionamiento adecuado pasando por *display*, botones, *dipswitch*, puertos para conexiones de otros periféricos como convertidores digitales, puertos de comunicación como Ethernet, USB-UART FMC, PMOD, SMA,HDMI, PS/2, incluso puertos para memorias SD.

A continuación se describirá el FPGA con el que se trabajara, número de compuertas, velocidad de operación, puertos entrada/salida, memoria interna y demás elementos.

El FPGA con el que se trabajara tiene por nombre Artix7 cuyo modelo es XC7A200T-2FBG676C que se puede observar en el número 1 de la fig. 3.2, cuenta con las siguientes características:

- ✓ 16825 Bloques Lógicos Configurables o Bloques de Arreglo Lógicos (CLB/LAB)
- ✓ 215360 Celdas Lógicas
- ✓ 13 Mb de RAM embebida
- ✓ 400 puertos entrada-salida
- ✓ 0.95[V] – 1.05[V] Voltaje de operación
- ✓ 0 [°C] – 85 [°C] Temperatura de operación

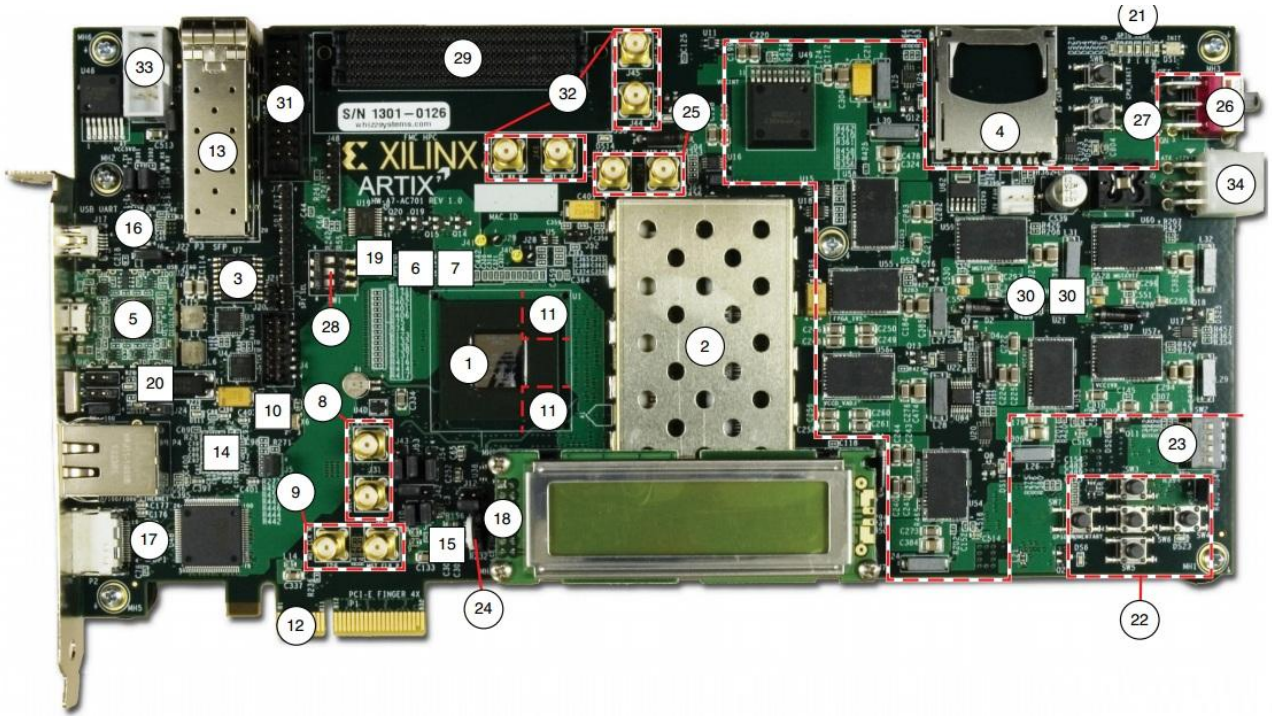


Fig. 3.2 Tarjeta Desarrollo Xilinx Artix7 AC701

En la tabla 3.1 se describen cada uno de los componentes y elementos con que se cuentan en la tarjeta AC701 de *Xilinx*.

1	FPGA Artix-7	Xilinx XC7A200T-2FBG676C
2	Memoria 1 GB DDR3 SODIMM	Micron MT8JT12864HZ-1G6G1
3	Memoria Quad-SPI Flash	Micron N25Q256A13ESF40G
4	Conector para tarjeta SD	Molex 67840-8001
5	MUSB-JTAG	Digilent USB JTAG Module
6	Reloj 200Mhz	SiTime SIT9102AI-243N25E200.0000
7	Reloj Programable 10MHz-810MHz	Silicon Labs SI570BAB000544DG
8	SMA	Rosenberger 32K10K-400L5
9	SMA GTP	Rosenberger 32K10K-400L5
10	Atenuador de Reloj	Silicon Labs SI5324-C-GM
11	GTP Transceivers	Embedded within FPGA U1

12	PCI Express Edge Connector	4-lane card edge connector
13	SFP/SFP+ Connector	Molex 74441-0010
14	10/100/1000 Tri-Speed Ethernet PHY	Marvell 88E1116RA0-NNC1C000
15	GTP Transceiver Reloj 125MHz	ICS ICS84402IAGI-01LF
16	Puente USB-to-UART	Silicon Labs CP2103GM
17	HDMI Conetor y Dispositive	Molex 500254-1927, Analog Devices ADV7511KSTZ-P
18	Conector LCD Character Display	2 x 7 0.1 in male pin header
19	I2C Bus Switch	TI PCA9548ARGER
20	Ethernet PHY Status LEDs, Green	Lumex SML-LX0603GW
21	User GPIO LEDs, Green	Lumex SML-LX0603GW
22	User Pushbuttons E-Switch	E-Switch TL3301EF100QG
23	GPIO DIP Switch, 4-pole	C&K SDA04H1SBD
24	User Rotary Switch	Panasonic EVQ-WK4001
25	SMA User GPIO	Rosenberger 32K10K-400L5
26	Power On/Off Slide Switch	C&K 1201M2S3AQE2
27	FPGA_PROG_B Pushbutton Switch	E-Switch TL3301EF100QG
28	Configuration Mode DIP Switch, 3-pole	C&K SDA03H1SBD
29	Conector FMC HPC	Samtec ASP_134486_01
30	Reguladores y controladores de voltaje	TI UCD90120ARGC
31	XADC Header	2X10 0.1 in. male header
32	MGT TX, RX SMA	Rosenberger 32K10K-400L5
33	Conector PMBus	Assmann AWHW10G-0202-T-R
34	6 pin Molex Mini-fit Jr. 12V	Molex 39-30-1060

Tabla 3.1 Componentes Tarjeta Xilinx Artix7 AC701

3.3 | CONVERTIDOR DIGITAL ANALÓGICO DAC3162EVM

Es necesario el uso de un convertidor digital analógico, para esto se eligió el DAC3162, de *Texas Instruments* Fig 3.3, debido a su alta frecuencia de muestreo y su bajo costo, trabaja a 500 MSPs con 12 bits de longitud de palabra, esta tarjeta cuenta con un puerto de comunicación DAC.

Fig. 3.3 DAC3162EVM Texas Instruments

Para realizar su conexión con el FPGA se utilizó el puerto FMC incorporado en la AC701 de Xilinx y para el acoplamiento de las tarjetas se utilizará un adaptador FMC-DAC (ver Fig. 3.4). Es necesario realizar un mapeo de las señales, desde el FPGA estableciendo los puertos de entrada o salida, acoplándolos con el puerto

FMC, posteriormente realizar el seguimiento de las señales mediante el adaptador FMC-DAC para finalmente llegar al circuito integrado DAC3162 y realizar la conversión digital analógica.

Fig. 3.4 Adaptador FMC-DAC Texas Instruments

En la fig. 3.5 se observa el diagrama de pines del DAC3162 podemos distinguir los 12 pares diferenciales correspondientes a la palabra binaria a convertir DxP y DxN es decir , que es necesario para un correcto funcionamiento del DAC cargar el dato positivo y el dato negado a través de dichos pines, además las señales de reloj DACCLKP y DACCLKN. Más adelante se abordara el tema de envió de bits mediante el mapeo de señales por el canal de comunicación

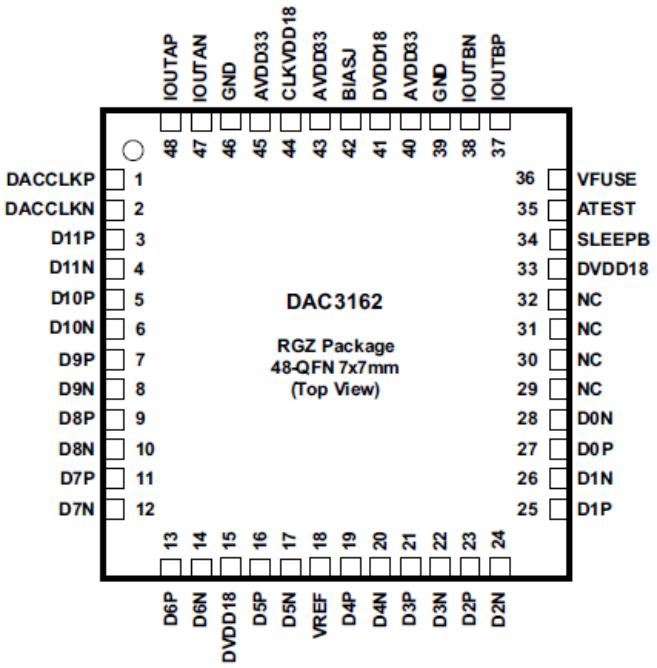


Fig. 3.5 DAC3162 diagrama de Pines

3.4 | TARJETA AMS 101

A su vez la Tarjeta de desarrollo AC701 cuenta con una tarjeta extra, la cual lleva por nombre AMS 101 ver (Fig. 3.6), en la tabla 3.2 se mencionan algunas características.

Fig. 3.6 Tarjeta AMS 101

Esta tarjeta permite realizar la conexión e interacción con el convertidor Analógico Digital embebido en el FPGA, mediante el puerto XADC propio de *Xilinx*.

1	Conector XADC
2	ADC 12 bits , 17 canales
3	1Msps
4	DAC 16 bits
5	Mini BNC

Tabla. 3.2 Elementos Tarjeta AMS 101

3.5 | CONVERTIDOR ANALÓGICO DIGITAL EMBEBIDO

La tarjeta AMS 101 es la que nos permitirá interactuar con los canales del Convertidor Analógico Digital embebido en el FPGA

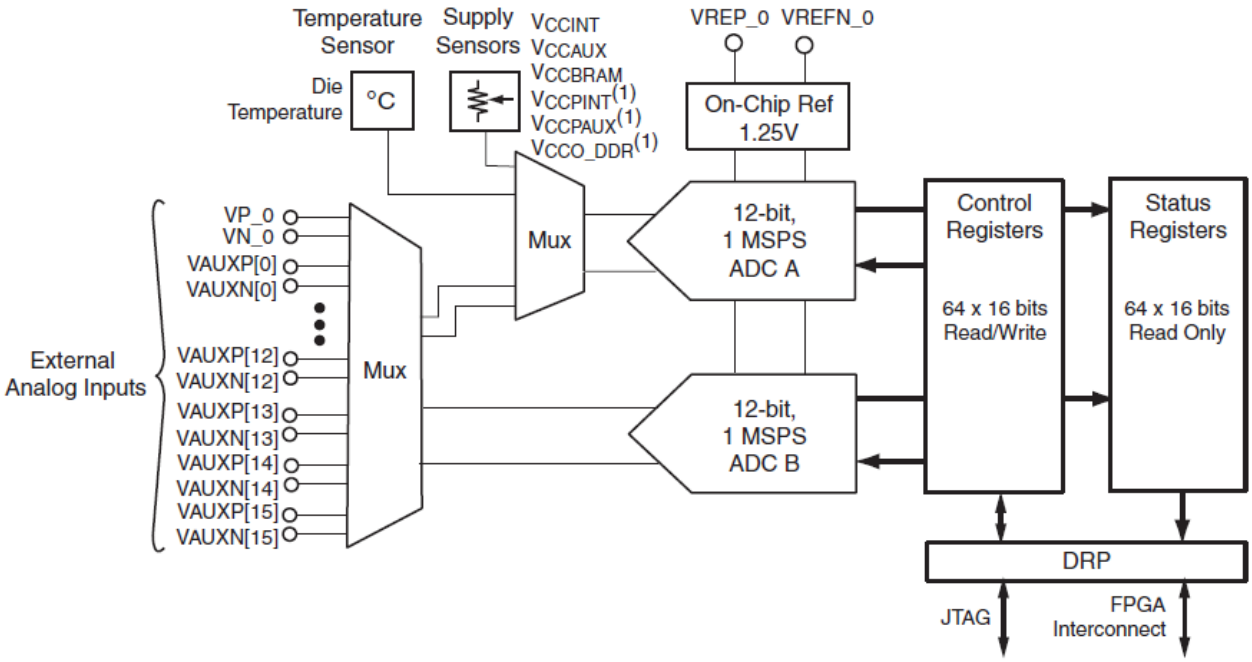


Fig. 3.7 Diagrama ADC embebido en FPGA Artix7 de Xilinx

En la Fig. 3.7 se puede observar el funcionamiento del ADC. Se cuenta con 16 canales de conversión, señales de monitoreo de voltaje y temperatura. Además está conformado por dos convertidores ADC, mismos que se usaran dependiendo de la función o canales que se elijan; la conversión se guardará en los registros de control, a los que se puede acceder mediante el puerto de reconfiguración dinámica DRP, o por JTAG (Joint Test Action Group por sus siglas en ingles).

La fig. 3.8, muestra las señales de control y puertos del XADC necesarias para su configuración y funcionamiento, las señales analógicas serán introducidas a través de los puertos llamados VP-VN y VAUXP [15:0]-VAUXN [15-0]. Las señales de reloj y control que darán inicio a la conversión analógica digital son CONVST y CONVSTCLK. El dato convertido esta en la señal llamada DO, un vector de 16 bits, el cual deber ser almacenado en memoria RAM, ya que el valor convertido únicamente estará en ese registro por un ciclo de reloj, debido a que en el siguiente ciclo se tendrá el siguiente valor de conversión.

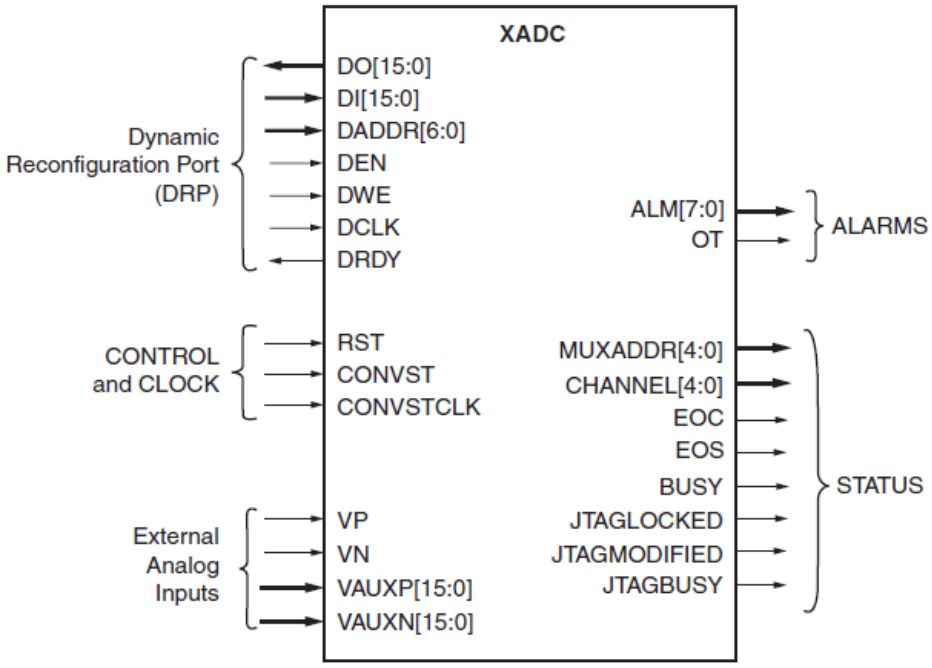
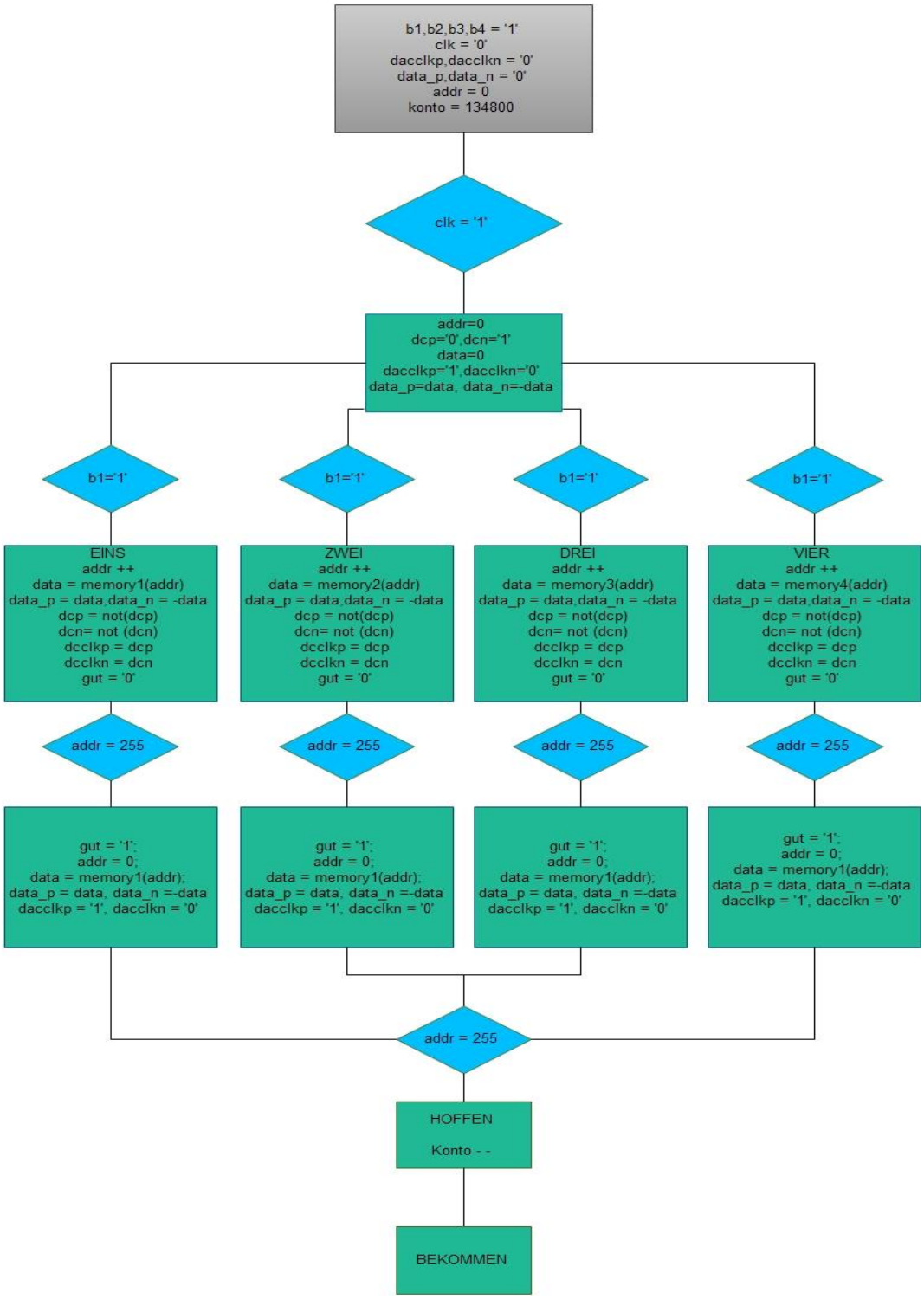


Fig. 3.8 Puertos XADC

3.6 | CARTA ASM


b1,b2,b3,b4 = '1'
clk = '0'
dacclkp,dacclkn = '0'
data_p,data_n = '0'
addr = 0
konto = 134800
clk = '1'
addr=0
dcp='0',dcn='1'
data=0
dacclkp='1',dacclkn='0'
data_p=data, data_n=-data
b1='1'
b1='1'
b1='1'
b1='1'
EINS
addr ++
data = memory1(addr)
data_p = data,data_n = -data
dcp = not(dcp)
dcn= not (dcn)
dcclkp = dcp
dcclkn = dcn
gut = '0'
ZWEI
addr ++
data = memory2(addr)
data_p = data,data_n = -data
dcp = not(dcp)
dcn= not (dcn)
dcclkp = dcp
dcclkn = dcn
gut = '0'
DREI
addr ++
data = memory3(addr)
data_p = data,data_n = -data
dcp = not(dcp)
dcn= not (dcn)
dcclkp = dcp
dcclkn = dcn
gut = '0'
VIER
addr ++
data = memory4(addr)
data_p = data,data_n = -data
dcp = not(dcp)
dcn= not (dcn)
dcclkp = dcp
dcclkn = dcn
gut = '0'
addr = 255
addr = 255
addr = 255
addr = 255
gut = '1';
addr = 0;
data = memory1(addr);
data_p = data, data_n =-data
dacclkp = '1', dacclkn = '0'
gut = '1';
addr = 0;
data = memory1(addr);
data_p = data, data_n =-data
dacclkp = '1', dacclkn = '0'
gut = '1';
addr = 0;
data = memory1(addr);
data_p = data, data_n =-data
dacclkp = '1', dacclkn = '0'
gut = '1';
addr = 0;
data = memory1(addr);
data_p = data, data_n =-data
dacclkp = '1', dacclkn = '0'
addr = 255
HOFFEN
Konto - -
BEKOMMEN

3.7 | SIMULACIÓN DEL DISEÑO EN *XILINX ISE*

En la fig. 3.9 se presenta el esquemático generado por el software *ISE* de *Xilinx* después de la implementación de la carta ASM en VHDL y su correcta síntesis. Este esquemático muestra la representación de los puertos de entrada y de salida requeridos para nuestro diseño.

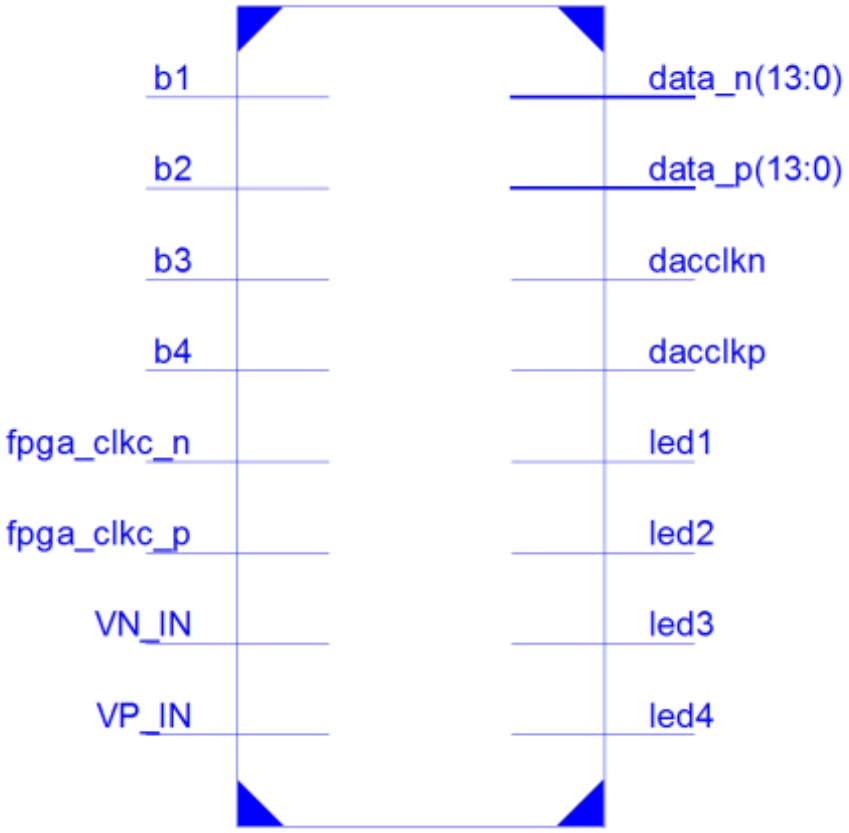


Fig. 3.9 Esquemático RLT post-síntesis

Con la ayuda del esquemático post-síntesis es posible corroborar de manera gráfica la correcta asignación de puertos de entrada y salida requeridos para el funcionamiento del sistema.

3.8 | MAPEO DE SEÑALES E IMPLEMENTACION

3.7.1 Mapeo de Señales para Transmisión de Datos al DAC

Como se mencionó anteriormente, es necesario, realizar un mapeo o ruteo de las señales a través de todo el canal de comunicación FPGA, Puerto FMC, Adaptador FMC, puerto DAC, DAC. A continuación se muestra una tabla del mapeo empleado para la correcta transmisión de las señales.

Señal DAC	Pin DAC	Puerto DAC	DAC-FMC	FMC	FPGA
DCLKP	1	96	FPGA_CLKOUTP	H4	D19
DCLKN	2	98	FPGA_CLKOUTN	H5	C19
D0P	32	131	IO_D2P	H22	M14
D0N	31	133	IO_D2N	H23	L14
D1P	30	125	IO_D3P	H19	B22
D1N	29	127	IO_D3N	H20	A22
D2P	28	119	IO_D4P	H16	B19
D2N	27	121	IO_D4N	H17	A19
D3P	25	113	IO_D5P	H13	H16
D3N	26	115	IO_D5N	H14	G16
D4P	23	107	IO_D6P	H10	F18
D4N	24	109	IO_D6N	H11	F19
D5P	21	101	IO_D7P	H7	H14

D5N	22	103	IO_D7N	H8	H15
D6P	19	89	IO_D8P	G36	G25
D6N	20	91	IO_D8N	G37	F25
D7P	16	83	IO_D9P	G33	E26
D7N	17	85	IO_D9N	G34	D26
D8P	13	77	IO_D10P	G30	G24
D8N	14	79	IO_D10N	G31	F24
D9P	11	71	IO_D11P	G27	G22
D9N	12	73	IO_D11N	G28	F22
D10P	9	65	IO_D12P	G24	L17
D10N	10	67	IO_D12N	G25	L18
D11P	7	59	IO_D13P	G21	M16
D11N	8	61	IO_D13N	G22	M17
D12P	5	53	IO_D14P	G18	E21
D12N	6	55	IO_D14N	G19	D21
D13P	3	47	IO_D15P	G15	E20
D13N	4	49	IO_D15N	G16	D20

TABLA 3.3 MAPEO DE SEÑALES PARA TRANSMISIÓN

3.7.2 IMPLEMENTACIÓN

Al hablar de implementación del diseño sobre el FPGA, se refiere a configurar los puertos de entrada y salida que se utilizaran en el desarrollo de acuerdo al FPGA, es decir indicar la localidad del puerto deseado en el encapsulado, dichas localidades será diferentes de acuerdo al circuito integrado.

Además de establecer las localidades necesarias en el FPGA , también es necesario establecer otras características de las señales tales como el voltaje de salida, para los que tenemos diferentes opciones; 1.2, 1.5, 1.8, 2.5, 3.3 o 5[V], impedancia, corriente, si será una señal diferencial o no, o si serán puertos usados para señales de reloj.

Dichas configuraciones se harán en un archivo llamado UCF *(User Constraint File*, por sus siglas en inglés), archivo propio de *Xilinx*, es en el cual el usuario establece, cada una de las condiciones, localidades y características necesarias para el uso adecuado de los pines de entrada y salida del FPGA. Este tipo de configuraciones son sumamente importantes debido a que si no son definidas de una forma adecuada, se pueden dañar los puertos del FPGA, así como las tarjetas auxiliares.

Capítulo 4

SIMULACION IMPLEMENTACION Y RESULTADOS

CAPÍTULO 4

SIIMULACIONES, IMPLEMENTACIÓN Y RESULTADOS

4.1| SIMULACIONES

El funcionamiento de nuestro sistema se describirá a continuación con ayuda de las simulaciones realizadas en ISIM, software que se puede usar posterior a la síntesis del código en VHDL, perteneciente a la suite ISE de Xilinx.

En la simulación mostrada en la fig. 4.1 podemos observar como en el tiempo $10^{\wedge 6}$ [ps], comienza a trabajar el sistema con la señal llamada *CLK*, que simula el reloj, el cual regirá el comportamiento de nuestro sistema, esta señal de reloj será constante, ya que todos los procesos están sincronizados con ella. La transmisión de las señales se elegidas por el usuario con ayuda de los *pushbotton b1,b2,b3,b4*, asignados a cada una de las señales anteriormente descritas que están almacenadas en memoria RAM del FPGA, el *b1* corresponde a la señal Seno, el *b2* a la Sinc, *b3* a la rampa y *b4* al pulso cuadrado.

En la simulación de la fig. 4.1 podemos observar la activación del *b1*, la cual inicializa todos los movimientos de la señales para realizar la transmisión de la señal correspondiente, en este caso es la señal senoidal, la cual enviará un paquete de 12 bits, positivo y negado, que equivalen a la primera de las 256 muestras de la senoidal. Al mismo tiempo se generan dos señales de reloj alternas llamadas: *dacclkp* y *dacclkn*, uno el negado del otro, estos relojes tienen una frecuencia de 1172 [Hz] y sincronizan

la conversión de dato enviado al DAC.

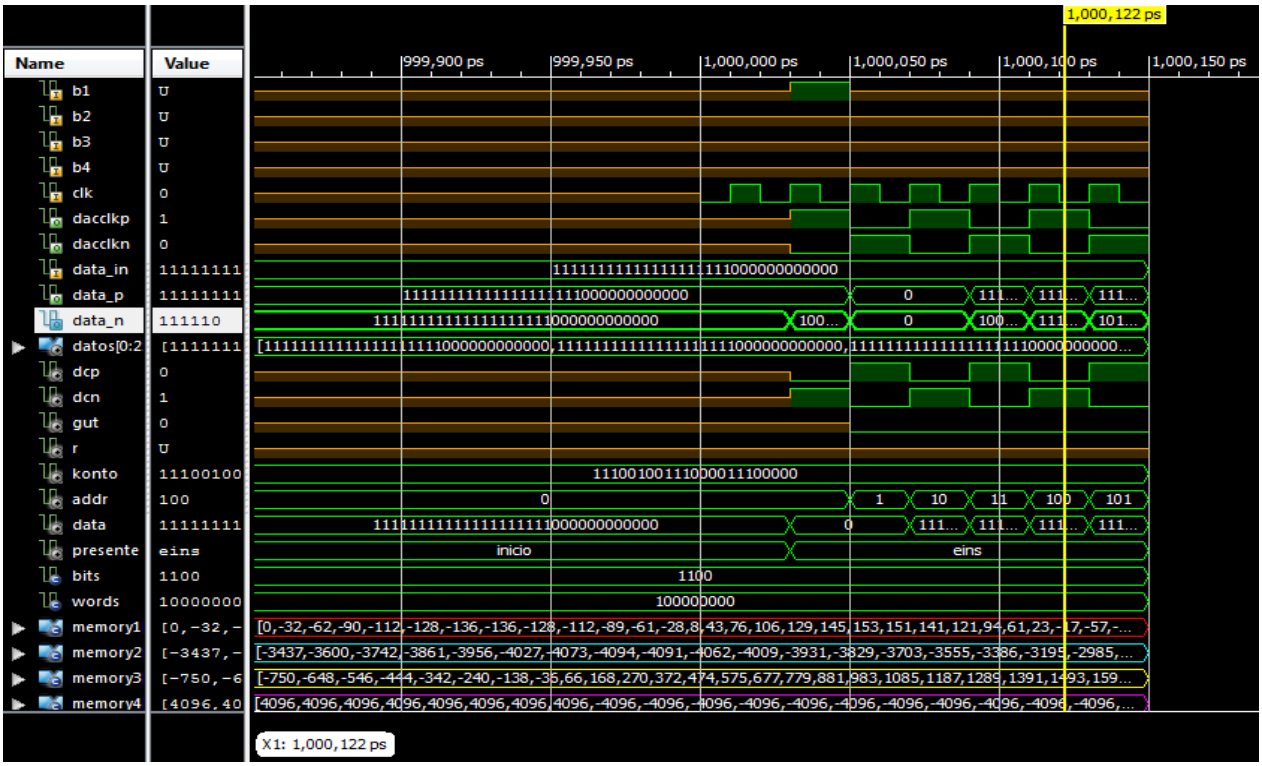


Fig. 4.1 Análisis de tiempos del Sistema en ISIM. Inicio

La señal *addr*, continente el número de muestras enviadas al DAC, los valores que van al convertidor son enviados mediante las señales *data_p* y *data_n,* una vez terminadas la transferencia de las 256 muestras obtendremos el periodo de nuestra señal seno igual a 3.333 [µs], la frecuencia necesaria para la excitación del cristal del transductor de 300 [KHz].

La línea amarilla mostrada en la fig. 4.2 indica el momento en el que el número de muestra de la señal *addr*, llega a su máximo valor 255. En ese instante las señales de reloj del DAC se apagan y comienza el tiempo de espera, relacionado con el tiempo

de vuelo necesario para que el pulso viaje cierta distancia. El tiempo de vuelo se calcula tomando en cuenta la velocidad del ultrasonido en agua y considerando 1 metro de distancia entre transductores, esto nos da un valor el cual será una constante que nos ayudara a llevar el contado, esta señal lleva el nombre de *Konto*, es la encargada de realizar la espera a la recepción y su valor inicial es 134800 y va en decremento, esto para dar inicio a las señales que corresponden a la activación del ADC para la recepción de la señal, como es muestra en la fig. 4.3.

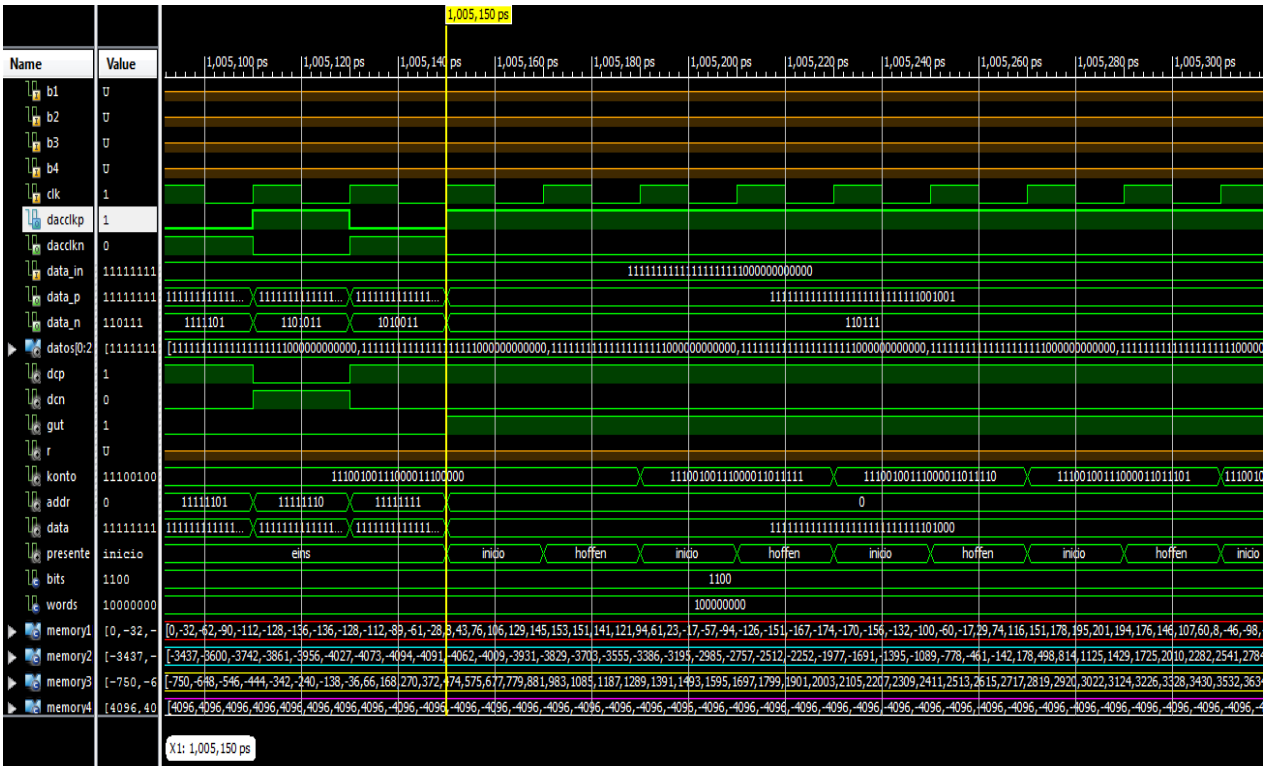


Fig. 4.2 Finaliza transmisión señal seno.

La Fig. 4.3 muestra el momento en el que la señal *Konto* termina la cuenta llegando al valor cero, una vez ahí comienza la configuración y activación del ADC.

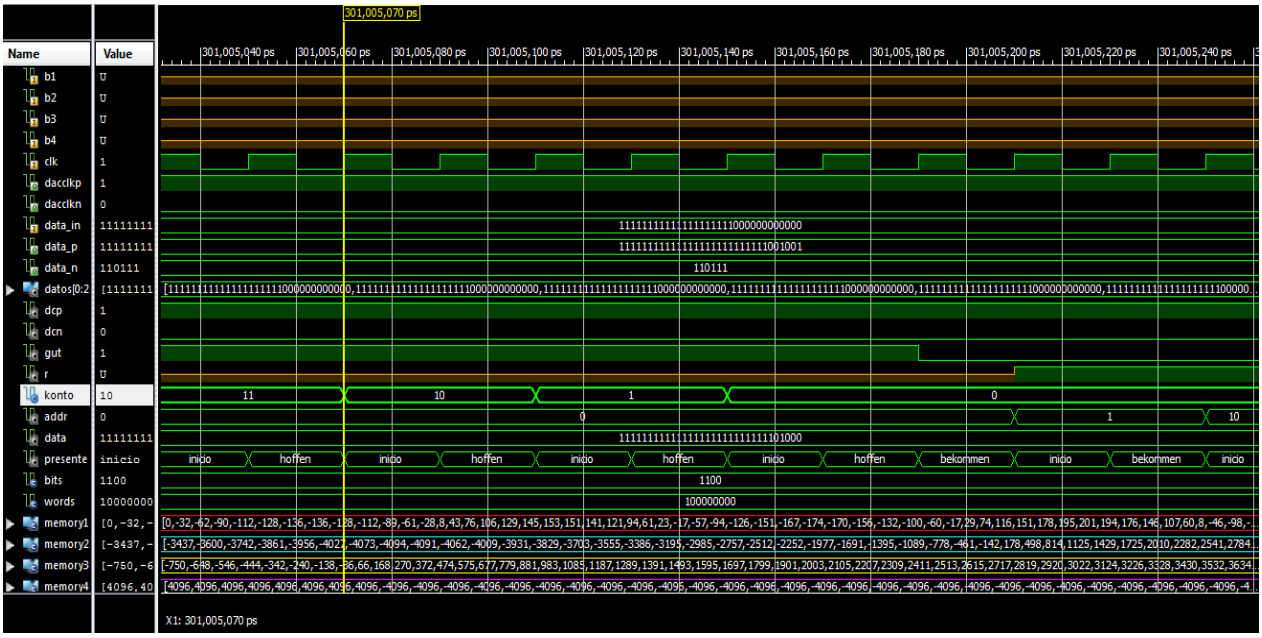


Fig. 4.3 Termina tiempo de espera relativo al tiempo de vuelo.

4.2| IMPLEMENTACIÓN Y RESULTADOS

Para la implementación del sistema se generaron señales de excitación del transductor del ultrasonido, se eligieron 4 funciones básicas generadas en la plataforma Matlab y cargadas en RAM del FPGA mediante VHDL, que permitirán analizar los transductores.

Las señales elegidas son: sinc (fig 4.4), senoidal (fig 4.5), diente de sierra (fig 4.6) y pulso cuadrado negativo (fig 4.7).Estas fueron generadas con 256 muestras en punto fijo y formato q12, para facilitar la operación en nuestro hardware digital.

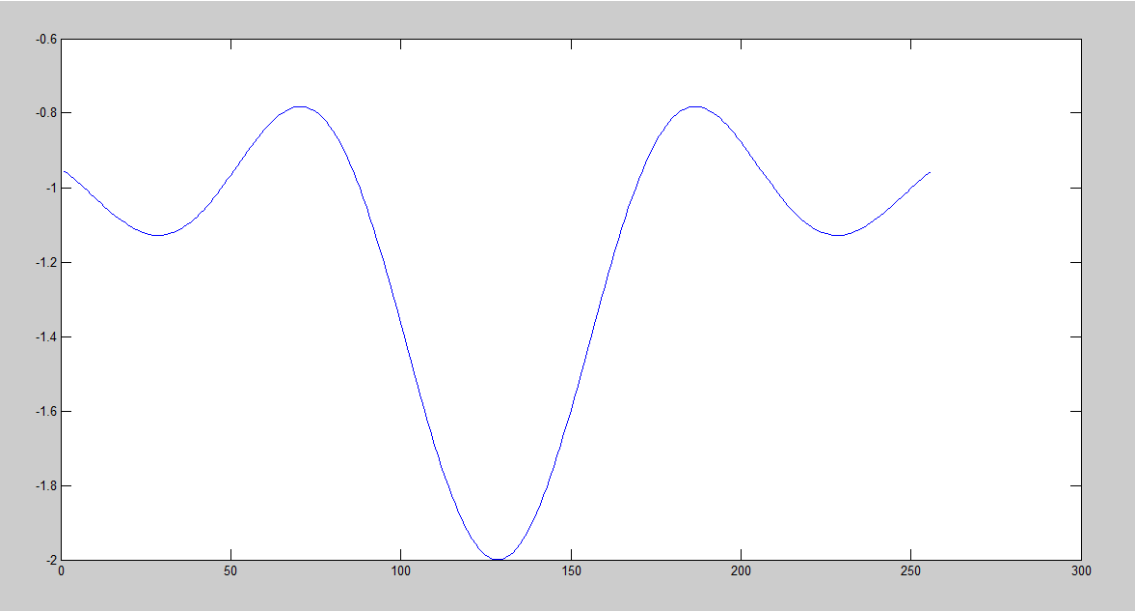


Fig. 4.4 Señal Sinc, 256 muestras generada en Matlab Punto Fijo q12.

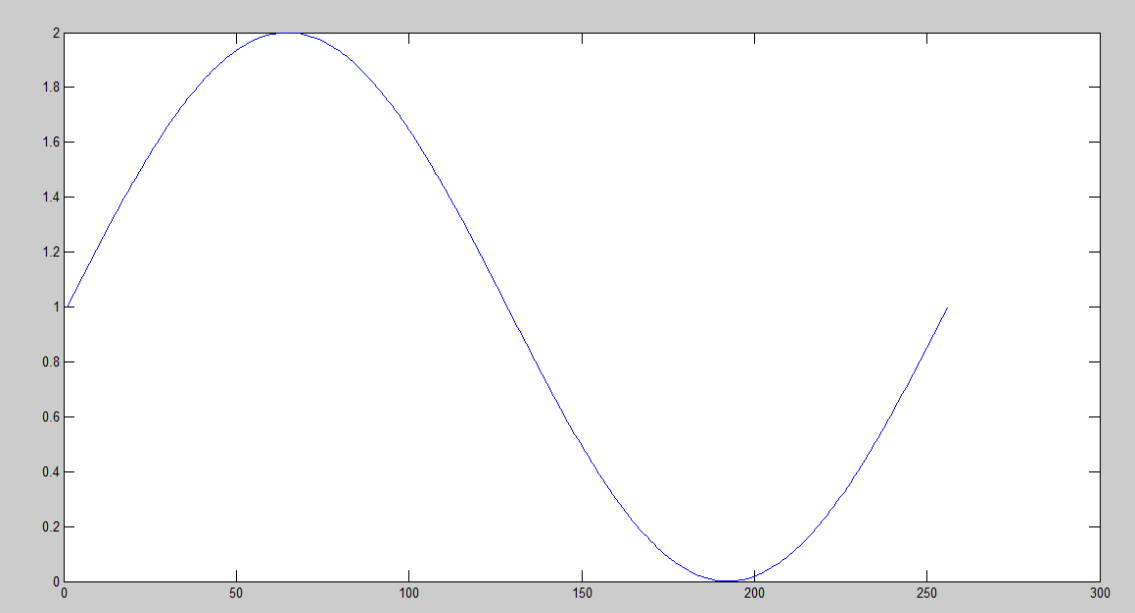


Fig. 4.5 Señal Seno, 256 muestras generada en Matlab Punto Fijo q12.

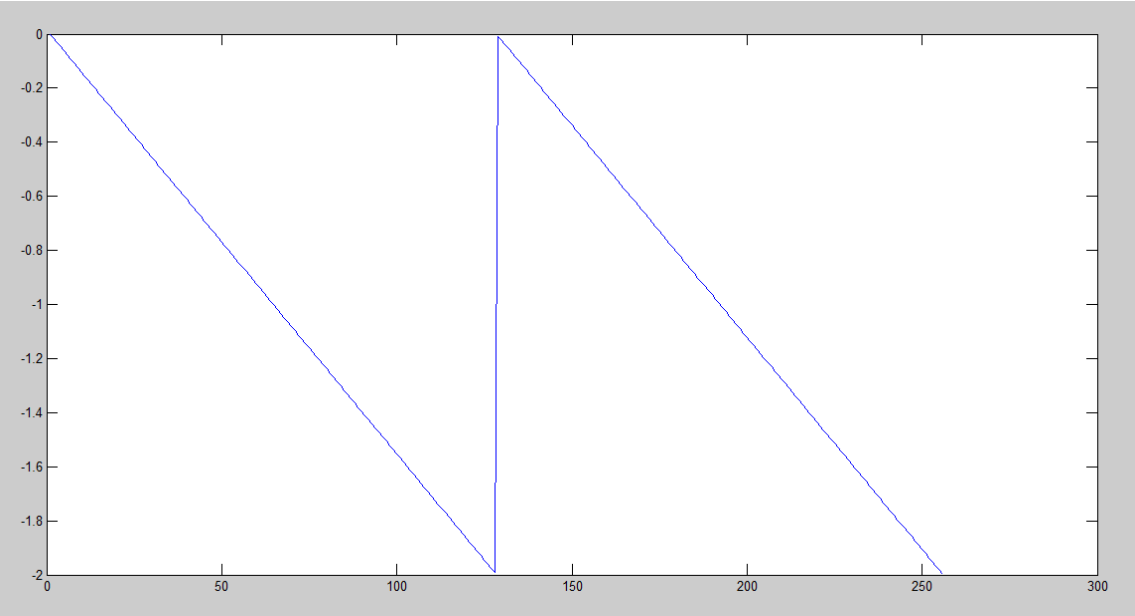


Fig. 4.6 Señal Diente de Sierra, 256 muestras generada en Matlab Punto Fijo q12.

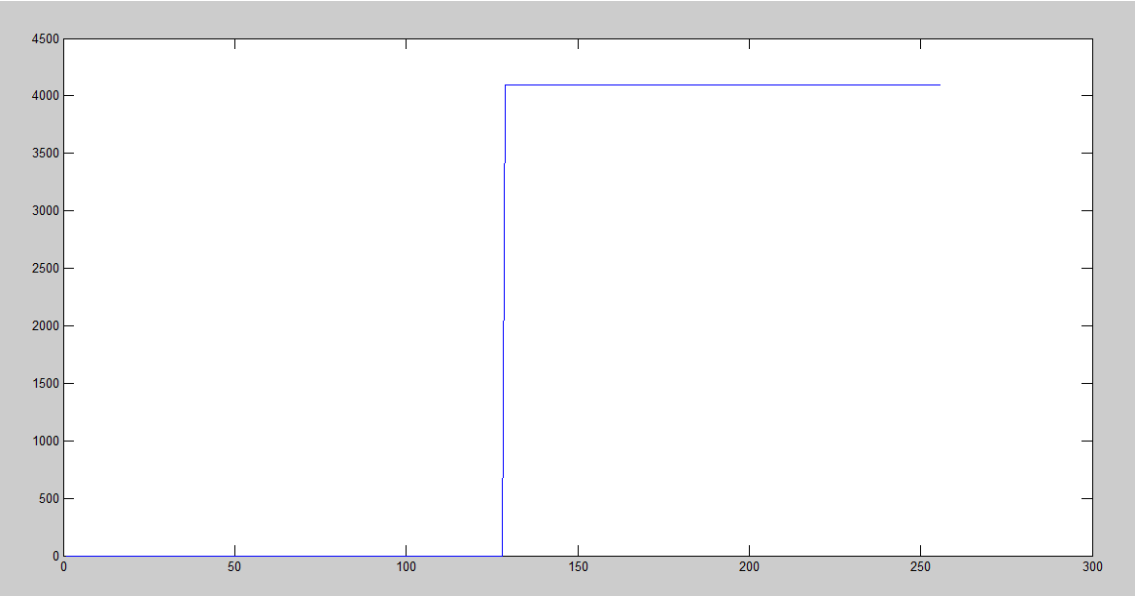


Fig. 4.7 Señal cuadrada Negativa, 256 muestras generada en Matlab Punto Fijo q12.

Antes de la implementación y medición en el FPGA, es necesario realizar un análisis de señales a través del circuito integrado. Este proceso se realiza con el software llamado *ChipScope* de Xilinx fig (4.8), programa que permite captar las señales en los buses del FPGA para analizarlas antes de ser enviadas al convertidor Digital Analógico DAC.

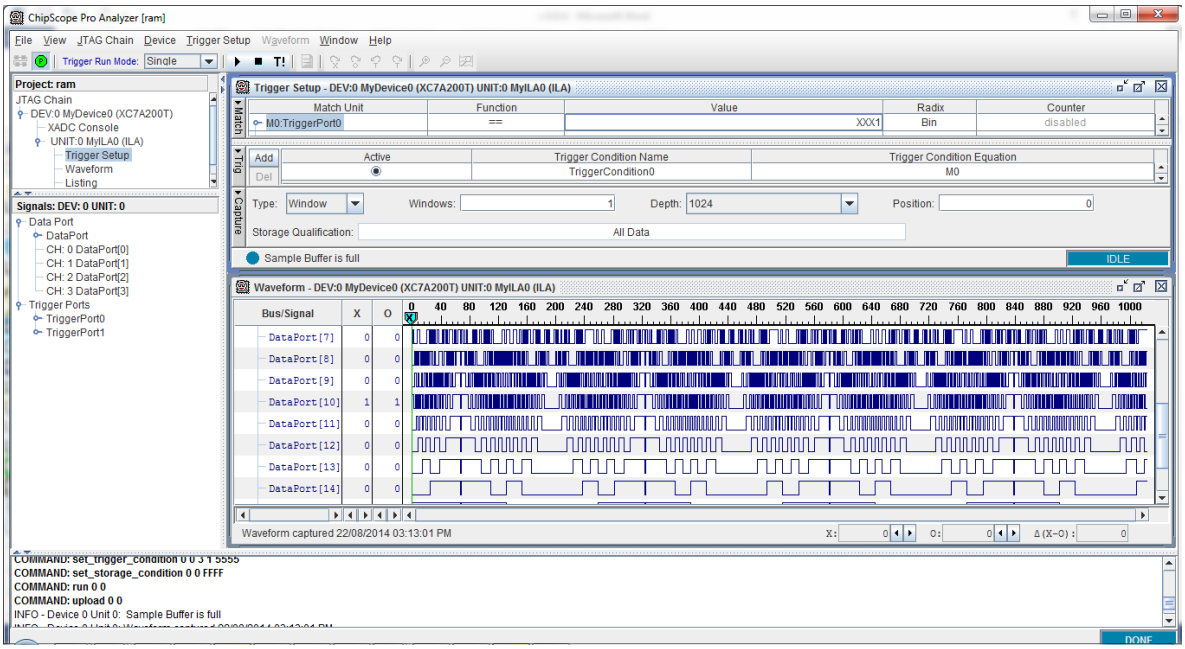


Fig. 4.8 Análisis de Señales en *Chipscope de Xilinx.*

Una vez comprobado el funcionamiento de buses a través del FPGA, se procede a la conexión del sistema completo FPGA, FMC, DAC. La alimentación de 5V se realiza con la fuente Agilent E3630A mostrada en la fig. 4.9.

Fig. 4.9 Agilent E3630A

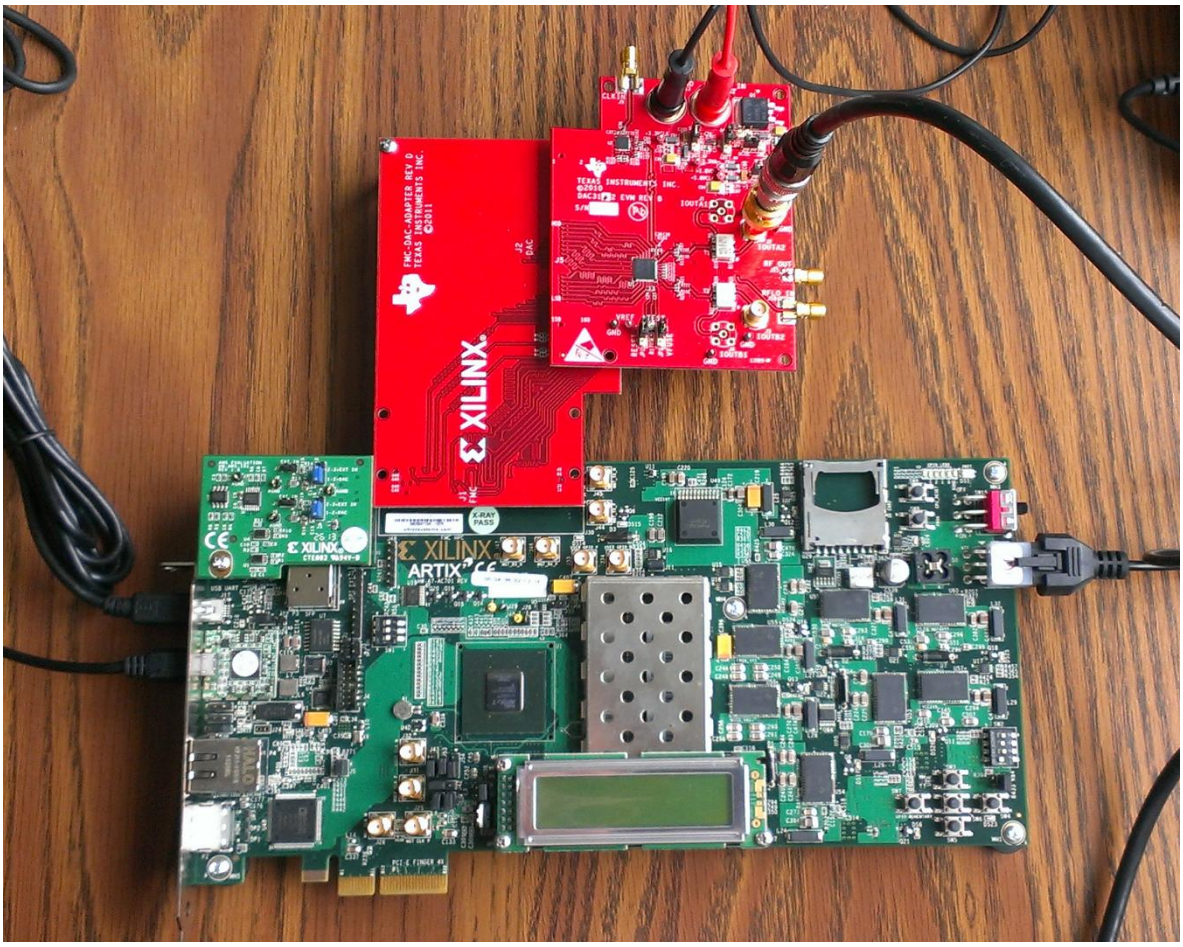


Fig. 4.10 Conexión de Tarjetas AC701, FMC-DAC, DAC3262EVM

En la fig. 4.10 se observa la conexión de las Tarjetas necesarias para este desarrollo la AC701 de Xilinx, el Adaptador FMC-DAC y DAC3262 EVM de Texas Instruments.

Las señales fueron capturadas con un Osciloscopio (fig. 4.11) mediante su función USB, Tektronix TDS 1012B, cuya máxima frecuencia para analizar señales es de 100 MHz y una frecuencia de muestreo de 1 Gs/s.

Fig. 4.11 Osciloscopio Tektronix TDS 1012B

Como se explicó en las simulaciones la activación, del sistema es realizada por los *pushbottons* presentes en la tarjeta de desarrollo AC701. El *b1* activa la transmisión de la señal senoidal. En la fig. 4.12. Esta señal es capturada por el osciloscopio a la salida del sistema de ultrasonido.

El *pushbotton* *b2* permite transmitir la señal Sinc mostrada en la fig. 4.13, *b3* señal cuadrada (ver fig. 4.14) y finalmente la señal diente de sierra que es activada por *b4*, fig. 4.15.

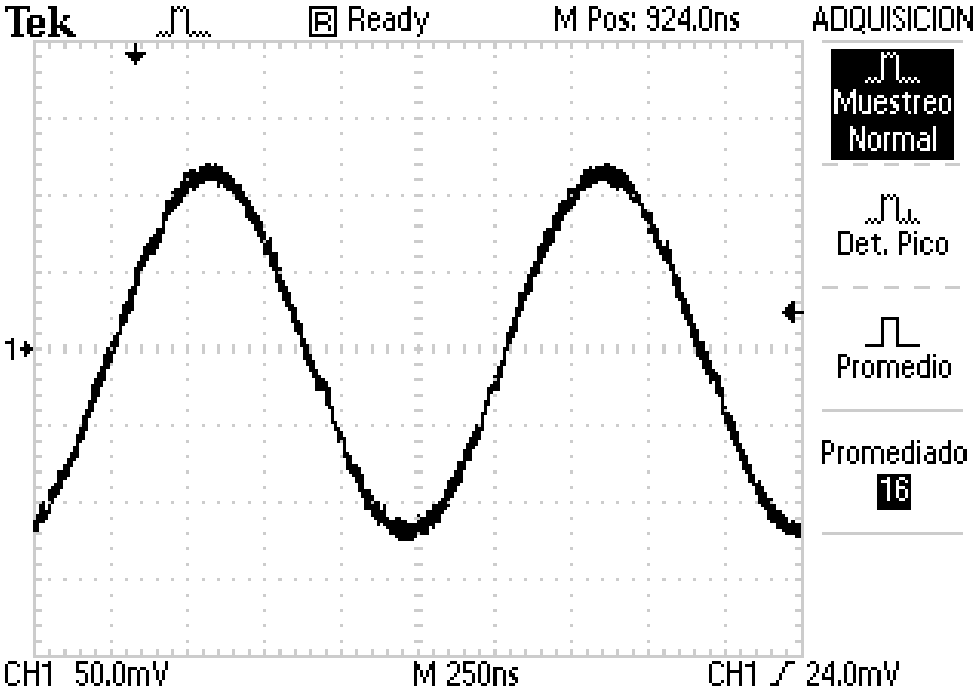


Fig. 4.12 Señal Senoidal generada por el sistema de Ultrasonido.

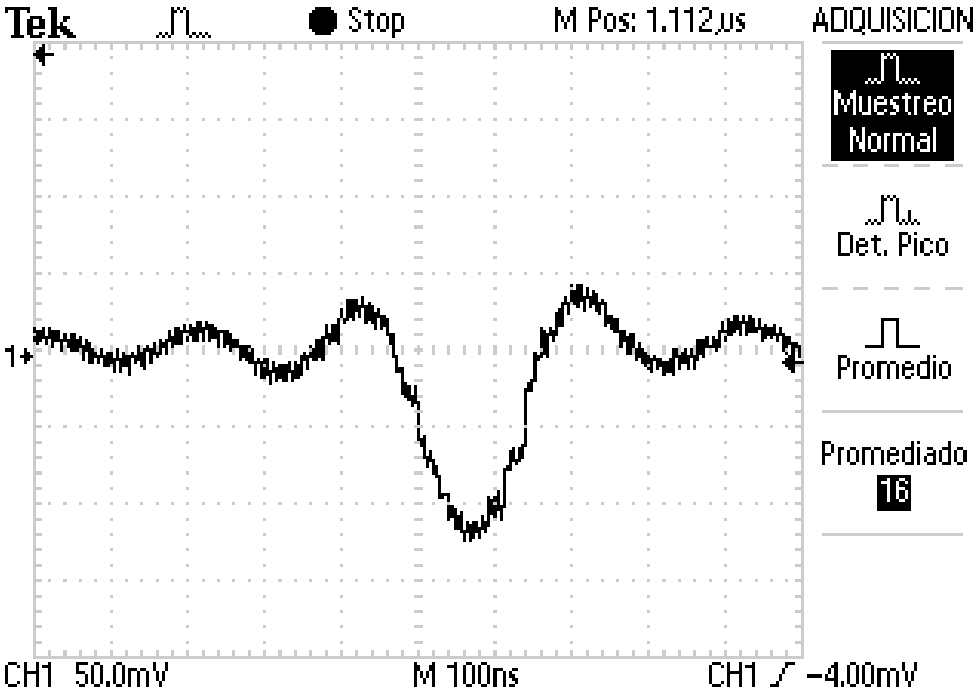


Fig. 4.13 Señal Sinc generada por el sistema de Ultrasonido.

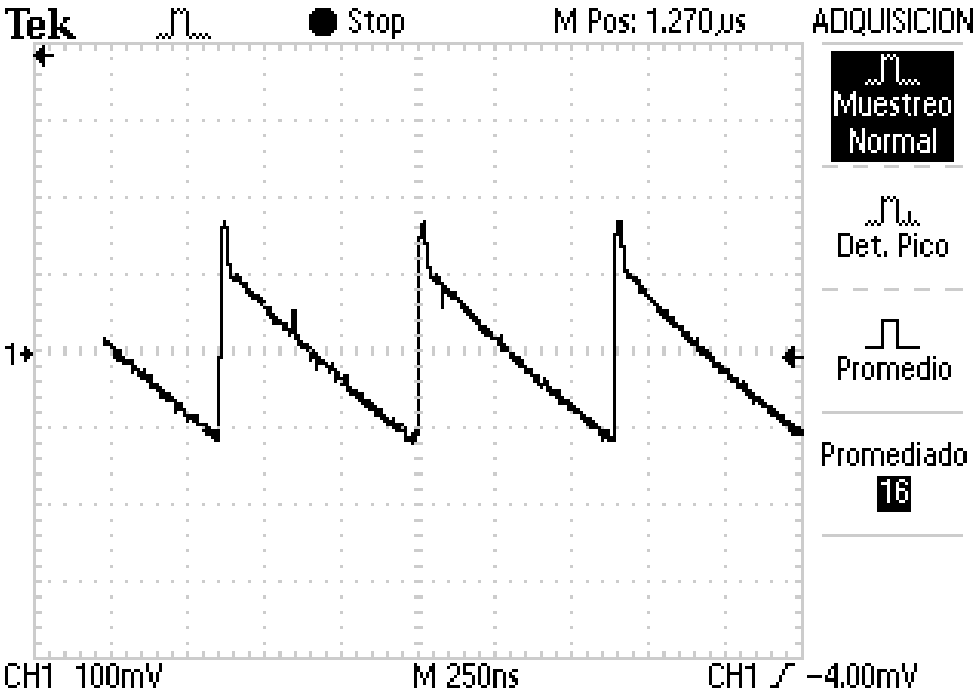


Fig. 4.14 Señal Diente de Sierra generada por el sistema de Ultrasonido.

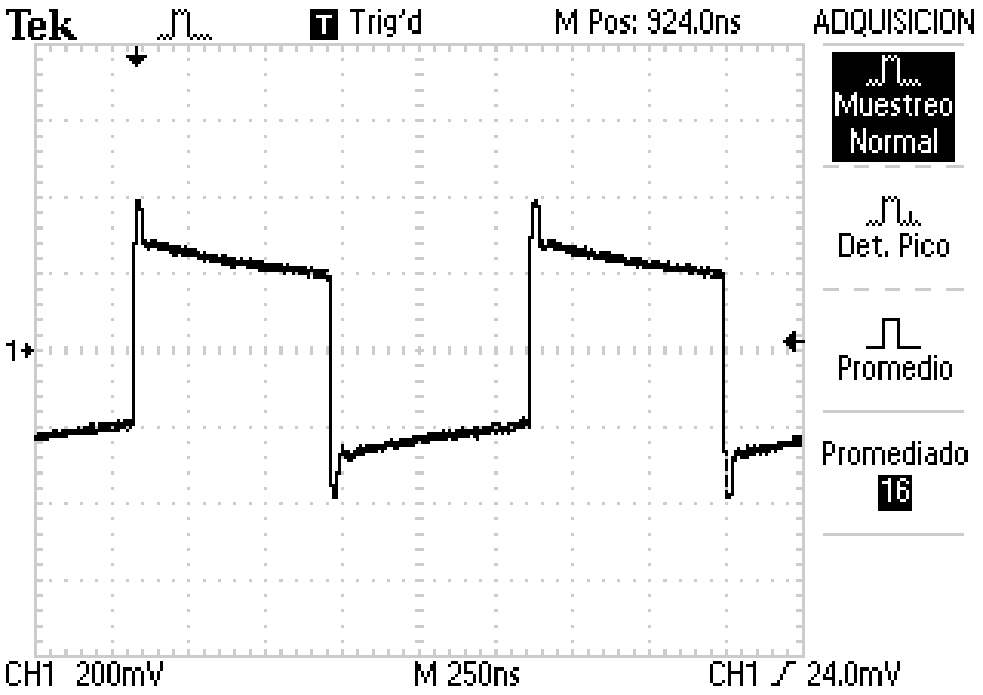


Fig. 4.15 Señal Cuadrada generada por el sistema de Ultrasonido.

4.3| COMPARACIÓN CON EQUIPO COMERCIAL ULTRATEK USB-UT350T

El equipo comercial *Ultratek USB-UT350T* es un pulsador-receptor cuyas características fundamentales son: 4 canales de recepción y transmisión, generación únicamente de un pulso cuadrado de hasta 300V y su costo es de $10,000 dólares. A continuación se mostrará el pulso generado por este dispositivo, cuando es capturado con el mismo osciloscopio *Tektronix TDS 1012B* utilizado anteriormente.

En la fig. 4.16 se puede observar el pulso de la señal cuadrada generado por el *Ultratek USB-UT350T*, en la cual se pueden observar oscilaciones muy marcadas en la parte baja del pulso, así como dos picos en las transiciones debidas al fenómeno de Gibbs.

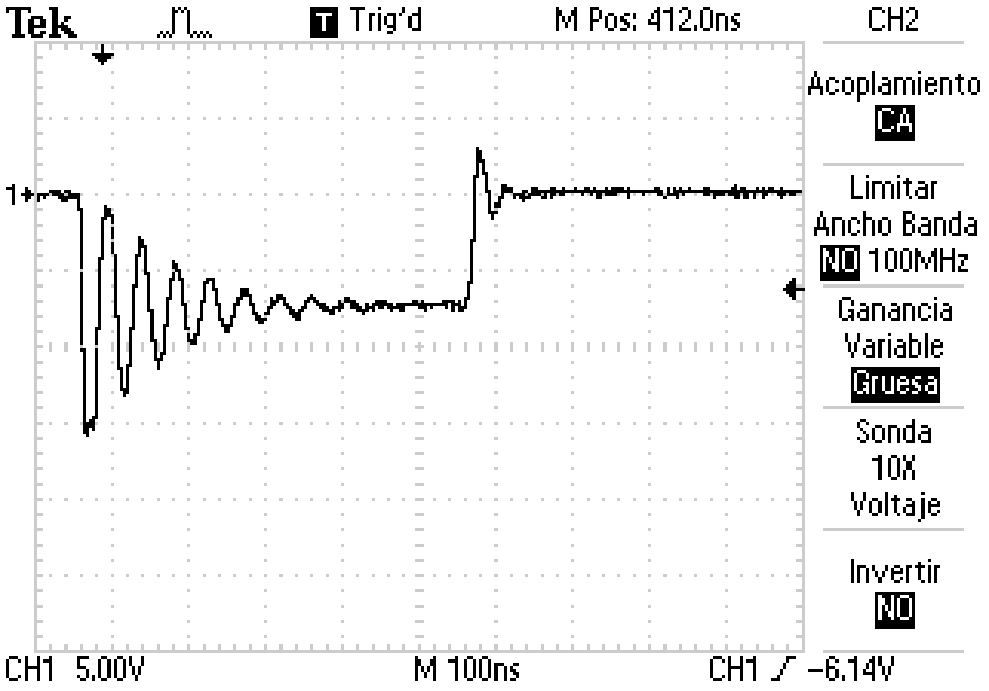


Fig. 4.16 Señal Cuadrada generada por *Ultratek USB-UT350T.*

En fig. 4.17 se observa la señal generada por el sistema de ultrasonido basado en FPGA, se puede apreciar los mismos picos en las transiciones de parte baja y alta del pulso cuadrado, sin embargo no existen oscilaciones como con el equipo comercial, esto es deseado para la excitación de un transductor.

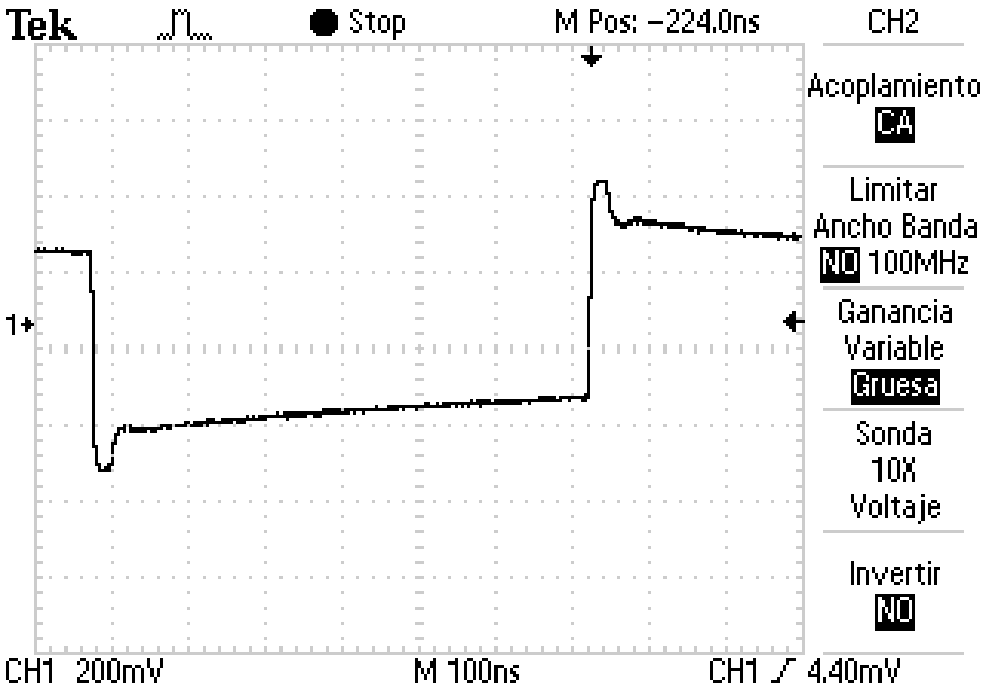


Fig. 4.17 Señal Cuadrada con Sistema ultrasonido basado en FPGA.

En las figuras mostradas a continuación se observa con mayor detalle, debido al acercamiento del pulso, las oscilaciones tan abruptas en la señal generada por el equipo comercial, fig. 4.18, y las cuales no aparecen en el sistema desarrollado en este trabajo,fig. 4.19.

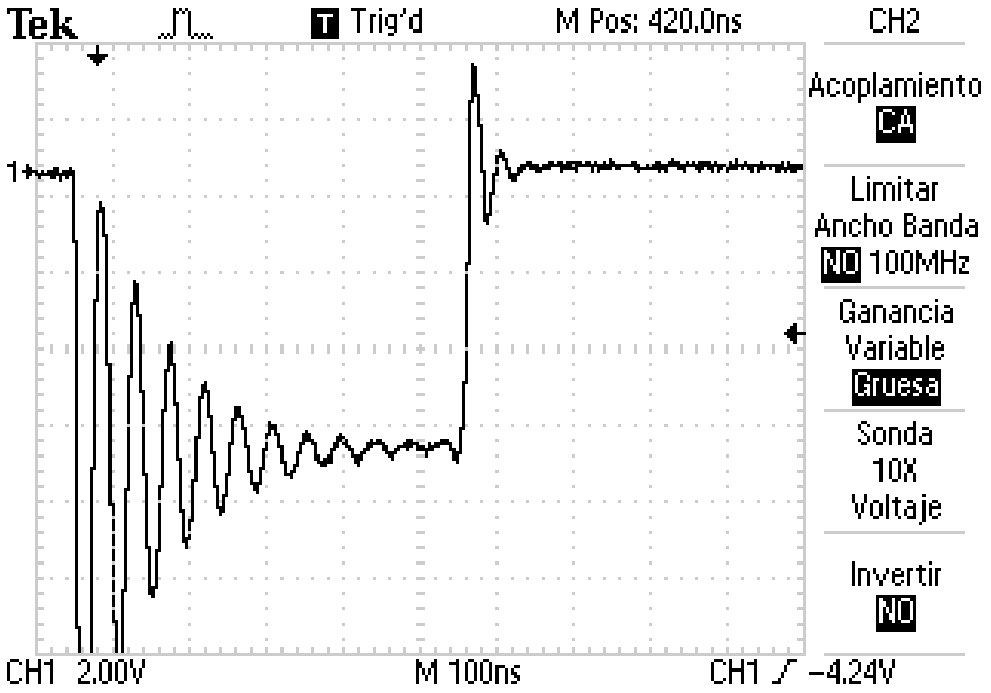


Fig. 4.18 Señal Cuadrada generada por *Ultratek USB-UT350T.*

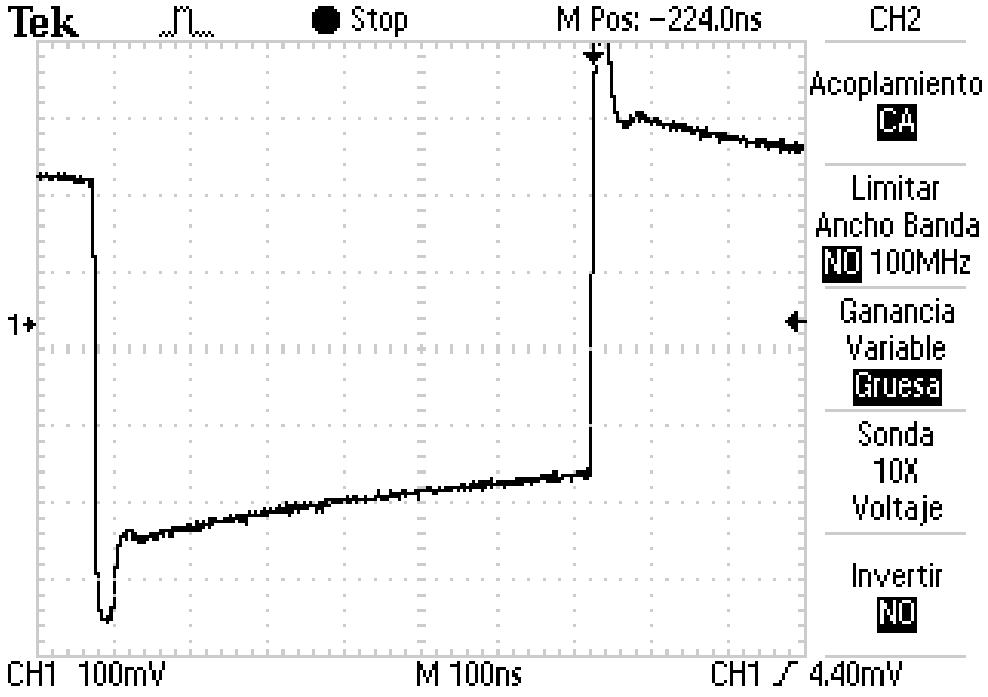


Fig. 4.19 Señal Cuadrada generada con el Sistema ultrasonido basado en FPGA.

Conclusiones y Trabajo Futuro

CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se realizó el desarrollo, (diseño, simulación, implementación), de un sistema de ultrasonido basado en un FPGA. Para llevar a cabo la investigación fue necesario el uso de la tarjeta de desarrollo AC701 de *Xilinx*, que consta de un convertidor Analógico Digital embebido, de 16 canales, además se requirió del uso de un convertidor Digital Analógico DAC3162EMV de *Texas Instruments*. Se estudió cada uno de los elementos necesarios para la trasmisión y recepción de señales; convertidores AD/DA, respuesta en frecuencia, formatos numéricos, tiempo de vuelo, FPGA.

En comparación con el uso de equipos comerciales, plataformas como DSP's y Microprocesadores, los FPGA's brindan mayor flexibilidad y bajo costo en el desarrollo de un sistema digital. El caso del sistema aquí presentado, brinda ventajas sustanciales, siendo las más importante de ellas; la posibilidad de enviar la señal que se desee, (siempre y cuando cumpla con las condiciones: 256 muestras y formato q12), para la excitación de los transductores; y la posibilidad de estudio y caracterización de materiales en ensayos no destructivos.

Las señales generadas con el FPGA, mostradas en las figuras 4.12 a 4.15, cumplen con las características de frecuencia necesarias para la excitación de los transductores de ultrasonido a usar 300KHz, así como la forma diseñada en Matlab. La frecuencia de la señal transmitida se puede incrementar aumentando la velocidad del ciclo de reloj del FPGA.

La transmisión de la señales y tiempo de espera es realizada en tiempo real, cumpliendo uno de los objetivos marcados en esta tesis; cabe mencionar que él envió de señales puede activarse una vez que el LED4 se apague, indicando el tiempo de captura de la señal recibida, este proceso, se puede repetir las veces que sea necesario así como enviar la señal que se desee, (Senoidal, Sinc, cuadrada, triangular).

Comparando los resultados del sistema de ultrasonido en FPGA aquí presentado con el sistema comercial *Ultratek USB-UT350T*, se pueden destacar dos ventajas: flexibilidad y bajo costo. Flexibilidad en cuanto a la trasmisión, cualquier señal que cumpla con 256 muestras q12, tiene la capacidad de ser transmitida mediante nuestro sistema, cabe destacar que se cuenta con la capacidad de recepción de señales con el convertidor ADC embebido en el FPGA. Hablando de bajo costo el equipo comercial *Ultratek USB-UT350T* tiene un costo de $10,000 dolares, el valor de la tarjeta de desarrollo AC701, el convertidor DAC3162EVM y el Adaptador FMC-DAC, no superan los 1,200 dolares por lo cual estamos hablando de una reducción en cuanto a costo del 88%.

Respecto al trabajo futuro: una de las limitantes importantes del sistema es el voltaje de salida, el cual no supera los 800 mVpp, por esta razón es necesario culminar la implementación de la electrónica analógica adecuada para la excitación y recepción de los transductores, además se tiene previsto hacer un sistema multicanal en la recepción del pulso, como consiguiente implementar algoritmos de reconstrucción tomográfica en el FPGA, efectuar las proyecciones de las imágenes reconstruidas mediante el puerto HDMI y posterior almacenamiento en tarjeta SD, con la finalidad de realizar un sistema de tomografía basado en ultrasonido, portable, bajo costo y en tiempo real.

Anexos

CODIGO MATLAB, DESCRIPCION DE HARDWARE EN VHDL Y UCF

ANEXOS

CODIGO MATLAB, DESCRIPCION DE HARDWARE EN VHDL y UCF.

A.1 | Codigo Matlab

Generación de las señales de excitación de los transductores ultrasónicos

```
t = (0:0.00614:2*pi);
x_sen = 1 + sin(t);
figure;plot(x_sen);
xq_sen=round(x_sen*2^12);
csvwrite('xq_sen1024.txt',xq_sen)



x_cos = 1 + cos(t);
figure;plot(x_cos);
xq_cos=round(x_cos*2^12);
csvwrite('xq_cos1024.txt',xq_cos)

t1 = (-pi:0.00614:pi)
x_sinc = 1 + sinc(t1);
figure;plot(x_sinc);
xq_sinc=round(x_sinc*2^12);
csvwrite('xq_sinc1024.txt',xq_sinc)
```

```
 x_puls(1:512)= zeros;
x_puls(513:1024)=4096;
figure;plot(x_puls);
csvwrite('xq_puls1024.txt',x_puls)


subplot(4,1,1), plot(xq_sen);
subplot(4,1,2), plot(xq_sinc);
subplot(4,1,3), plot(xq_dien);
subplot(4,1,4), plot(x_puls);
```

A.2 | Descripción de Hardware en VHDL

```
library ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

--XADC
use ieee.numeric_std.all;
Library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
--

ENTITY ram IS
    GENERIC (bits: INTEGER:=12;
                    words :INTEGER := 256);
    PORT (b1,b2,b3,b4,clk: IN std_logic;
        dacclkp: out std_logic;
                dacclkn: out std_logic;
```

```
                    --gut: inout std_logic;
                    --data_in : in integer range -4096 to 4096;
                    data_p: OUT integer range -4096 to 4096;
                    data_n: out integer range -4096 to 4096;
----------------------XADC---------------------------
                    --
        VP_IN              : in  STD_LOGIC;                    -- Dedicated Analog Input
Pair
        VN_IN              : in  STD_LOGIC
-----------------------------------------------------
                    );
end ram;

ARCHITECTURE ram of ram is
type vector_array1 is ARRAy (0 to words-1) OF integer range -4096 to 4096;
    constant                                                    memory1:
vector_array1 :=( 0,101,201,302,402,503,602,702,801,899,997,1095,1192,1288,1383,1477,1571,
1663,1755,1846,1935,2023,2110,2196,2280,2363,2445,2525,2603,2680,2756,2830,2902,2972,3040
,3107,3172,3234,3295,3354,3411,3466,3518,3569,3617,3663,3707,3749,3788,3826,3860,3893,392
3,3951,3976,3999,4020,4038,4054,4067,4078,4086,4092,4095,4096,4094,4090,4084,4075,4063,40
49,4033,4014,3993,3969,3943,3915,3884,3850,3815,3777,3737,3695,3650,3603,3554,3503,3450,3
394,3337,3278,3216,3153,3087,3020,2951,2881,2808,2734,2658,2581,2502,2421,2339,2256,2171,
2085,1997,1909,1819,1728,1636,1544,1450,1355,1260,1163,1066,969,871,772,673,573,473,373,2
73,172,71,-30,-130,-231,-331,-432,-532,-632,-731,-830,-928,-1026,-1123,-1220,-1316,-1411,-
1505,-1598,-1690,-1782,-1872,-1961,-2049,-2135,-2221,-2305,-2387,-2468,-2548,-2626,-2703,-
2778,-2851,-2922,-2992,-3060,-3126,-3190,-3252,-3313,-3371,-3427,-3481,-3533,-3583,-3631,-
3676,-3720,-3761,-3800,-3836,-3870,-3902,-3931,-3959,-3983,-4006,-4025,-4043,-4058,-4070,-
4080,-4088,-4093,-4096,-4096,-4093,-4089,-4081,-4072,-4059,-4045,-4028,-4008,-3986,-3962,-
3935,-3906,-3874,-3840,-3804,-3766,-3725,-3682,-3637,-3589,-3539,-3488,-3434,-3378,-3320,-
3260,-3198,-3134,-3068,-3000,-2931,-2860,-2787,-2712,-2636,-2558,-2478,-2397,-2315,-2231,-
2146,-2059,-1972,-1883,-1793,-1701,-1609,-1516,-1422,-1327,-1232,-1135,-1038,-940,-842,-743,-
```

644,-544,-444,-344,-243,-142,-42);

type vector_array2 is array (0 to words-1) of integer range -4096 to 4096;

constant memory2: vector_array2 := (161,140,116,88,57,25,-8,-42,-75,-107,-137,-165,-188,-208,-223,-233,-237,-236,-229,-217,-199,-176,-148,-116,-81,-43,-3,37,78,117,154,189,219,245,266,281,289,290,285,272,253,227,195,157,115,69,20,-30,-81,-131,-180,-225,-266,-302,-332,-354,-368,-374,-371,-359,-338,-308,-269,-223,-170,-112,-48,18,86,155,222,286,346,399,445,481,508,523,525,515,493,457,408,348,276,194,103,6,-97,-203,-309,-413,-513,-607,-691,-763,-821,-862,-886,-888,-869,-828,-762,-672,-558,-420,-259,-75,128,351,589,842,1105,1375,1650,1926,2199,2466,2723,2967,3194,3401,3585,3744,3876,3978,4049,4088,4095,4069,4011,3921,3802,3654,3480,3282,3063,2826,2574,2311,2040,1765,1489,1216,949,692,448,218,7,-185,-356,-504,-628,-728,-803,-855,-883,-889,-874,-840,-789,-722,-643,-553,-455,-352,-247,-140,-36,64,157,243,319,385,438,479,508,523,525,515,494,461,419,369,312,249,183,115,46,-21,-86,-147,-202,-251,-293,-326,-351,-367,-374,-372,-361,-342,-315,-282,-243,-199,-152,-102,-51,-0,49,96,140,180,214,243,265,280,289,290,285,273,255,231,202,169,133,94,54,13,-27,-65,-102,-135,-165,-190,-210,-225,-234,-237,-235,-228,-215,-197,-175,-149,-120,-89,-56,-22,11,44,75,104,130,153);

type vector_array3 is array (0 to words-1) of integer range -4096 to 4096;

constant memory3: vector_array3 := (-4096,-4064,-4032,-4000,-3968,-3936,-3904,-3871,-3839,-3807,-3775,-3743,-3711,-3679,-3647,-3615,-3583,-3551,-3519,-3487,-3455,-3422,-3390,-3358,-3326,-3294,-3262,-3230,-3198,-3166,-3134,-3102,-3070,-3038,-3006,-2973,-2941,-2909,-2877,-2845,-2813,-2781,-2749,-2717,-2685,-2653,-2621,-2589,-2556,-2524,-2492,-2460,-2428,-2396,-2364,-2332,-2300,-2268,-2236,-2204,-2172,-2140,-2107,-2075,-2043,-2011,-1979,-1947,-1915,-1883,-1851,-1819,-1787,-1755,-1723,-1690,-1658,-1626,-1594,-1562,-1530,-1498,-1466,-1434,-1402,-1370,-1338,-1306,-1274,-1241,-1209,-1177,-1145,-1113,-1081,-1049,-1017,-985,-953,-921,-889,-857,-825,-792,-760,-728,-696,-664,-632,-600,-568,-536,-504,-472,-440,-408,-375,-343,-311,-279,-247,-215,-183,-151,-119,-87,-55,-23,9,41,74,106,138,170,202,234,266,298,330,362,394,426,458,490,523,555,587,619,651,683,715,747,779,811,843,875,907,940,972,1004,1036,1068,1100,1132,1164,1196,1228,1260,1292,1324,135

```
6,1389,1421,1453,1485,1517,1549,1581,1613,1645,1677,1709,1741,1773,1806,1838,1870,1902,19
34,1966,1998,2030,2062,2094,2126,2158,2190,2222,2255,2287,2319,2351,2383,2415,2447,2479,2
511,2543,2575,2607,2639,2671,2704,2736,2768,2800,2832,2864,2896,2928,2960,2992,3024,3056,
3088,3121,3153,3185,3217,3249,3281,3313,3345,3377,3409,3441,3473,3505,3537,3570,3602,3634
,3666,3698,3730,3762,3794,3826,3858,3890,3922,3954,3987,4019,4051,4083);

type vector_array4 is array (0 to words-1) of  integer range -4096 to 4096;
    constant memory4: vector_array4 := (-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-4096,-
4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,409
6,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,40
96,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4
096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,
4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,409
6,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,40
96,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4096,4
096,4096,4096,4096,4096);

type vector_array5 is array (0 to words-1) of  integer range -4096 to 4096;
    signal datos: vector_array4;

------------XADC------------------------
component xadc_wiz_v2_1
  port (
```

```
        CONVST_IN          : in  STD_LOGIC;                      -- Convert Start Input
        DADDR_IN           : in  STD_LOGIC_VECTOR (6 downto 0);  -- Address bus
for the dynamic reconfiguration port
        DCLK_IN            : in  STD_LOGIC;                      -- Clock input for the
dynamic reconfiguration port
        DEN_IN             : in  STD_LOGIC;                      -- Enable Signal for the
dynamic reconfiguration port
        DI_IN              : in  STD_LOGIC_VECTOR (15 downto 0); -- Input data bus
for the dynamic reconfiguration port
        DWE_IN             : in  STD_LOGIC;                      -- Write Enable for the
dynamic reconfiguration port
        RESET_IN           : in  STD_LOGIC;                      -- Reset signal for the
System Monitor control logic
        BUSY_OUT           : out  STD_LOGIC;                     -- ADC Busy signal
        CHANNEL_OUT        : out  STD_LOGIC_VECTOR (4 downto 0); -- Channel
Selection Outputs
        DO_OUT             : out  STD_LOGIC_VECTOR (15 downto 0); -- Output data
bus for dynamic reconfiguration port
        DRDY_OUT           : out  STD_LOGIC;                     -- Data ready signal for
the dynamic reconfiguration port
        EOC_OUT            : out  STD_LOGIC;                     -- End of Conversion
Signal
        EOS_OUT            : out  STD_LOGIC;                     -- End of Sequence Signal
        JTAGBUSY_OUT       : out  STD_LOGIC;                     -- JTAG DRP
transaction is in progress signal
        JTAGLOCKED_OUT     : out  STD_LOGIC;                     -- DRP port lock
request has been made by JTAG
        JTAGMODIFIED_OUT   : out  STD_LOGIC;                     -- Indicates JTAG
Write to the DRP has occurred
        ALARM_OUT          : out STD_LOGIC;                      -- OR'ed output of all
the Alarms
```

```
        VP_IN              : in  STD_LOGIC;                    -- Dedicated Analog Input
Pair
        VN_IN              : in  STD_LOGIC
);
end component;

------------dac ports--------------------
signal dcp: std_logic;
signal dcn: std_logic;
-----------------------------------------
--------------adc ports--------------------

signal                  CONVST_IN:          STD_LOGIC;                  -- Convert
Start Input
signal                  DADDR_IN :          STD_LOGIC_VECTOR (6 downto 0);    -
- Address bus for the dynamic reconfiguration port
signal        DCLK_IN           :   STD_LOGIC;                  -- Clock input for the
dynamic reconfiguration port
signal        DEN_IN            :   STD_LOGIC;                  -- Enable Signal for the
dynamic reconfiguration port
signal        DI_IN             :  STD_LOGIC_VECTOR (15 downto 0);    -- Input data
bus for the dynamic reconfiguration port
signal        DWE_IN            :  STD_LOGIC;                   -- Write Enable for the
dynamic reconfiguration port
signal        RESET_IN          :  STD_LOGIC;                   -- Reset signal for the
System Monitor control logic
signal        BUSY_OUT          :   STD_LOGIC;                  -- ADC Busy signal
signal       CHANNEL_OUT        :   STD_LOGIC_VECTOR (4 downto 0);    -- Channel
Selection Outputs
signal        DO_OUT            :   STD_LOGIC_VECTOR (15 downto 0);   -- Output
data bus for dynamic reconfiguration port
```

```
signal         DRDY_OUT          :   STD_LOGIC;                    -- Data ready signal
for the dynamic reconfiguration port
signal         EOC_OUT           :  STD_LOGIC;                     -- End of Conversion
Signal
signal         EOS_OUT           :   STD_LOGIC;                    -- End of Sequence
Signal
signal         JTAGBUSY_OUT      :  STD_LOGIC;                     -- JTAG DRP
transaction is in progress signal
signal         JTAGLOCKED_OUT    :   STD_LOGIC;                    -- DRP port lock
request has been made by JTAG
signal         JTAGMODIFIED_OUT  :   STD_LOGIC;                    -- Indicates JTAG
Write to the DRP has occurred
signal         ALARM_OUT         :  STD_LOGIC;                     -- OR'ed output of all
the Alarms

signal gut: std_logic;
signal r: std_logic;
signal konto: integer := 7500000;
signal addr: integer range 0 to 256;
signal data: integer range -4096 to 4096;
type estados is (inicio,eins,zwei,drei,vier,hoffen,bekommen);
signal presente: estados:=inicio ;
BEGIN
XADc : xadc_wiz_v2_1
  port map (
        CONVST_IN          => CONVST_IN,
        DADDR_IN           => DADDR_IN,
        DCLK_IN            => DCLK_IN,
        DEN_IN             => DEN_IN,
        DI_IN              => DI_IN,
        DWE_IN             => DWE_IN,
```

```
      RESET_IN          => RESET_IN,
      BUSY_OUT          => BUSY_OUT,
      CHANNEL_OUT       => CHANNEL_OUT,
      DO_OUT            => DO_OUT,
      DRDY_OUT          => DRDY_OUT,
      EOC_OUT           => EOC_OUT,
      EOS_OUT           => EOS_OUT,
      JTAGBUSY_OUT      => JTAGBUSY_OUT,
      JTAGLOCKED_OUT    => JTAGLOCKED_OUT,
      JTAGMODIFIED_OUT  => JTAGMODIFIED_OUT,
    ALARM_OUT         => ALARM_OUT,                -- OR'ed output of all the
Alarms
      VP_IN             => VP_IN,
      VN_IN             => VN_IN
     );

    PROCESS(clk)
    begin
     IF (clk'event and clk='1') THEN
     case presente is
          when inicio => if b1='1' then
                                        addr <= 0;
                                        dcp <= '0';
                                        dcn <= '1';
                                        data<= memory1(addr);
                                        data_p <= data;
                                        data_n <= -data;
                                        dacclkp <= '1';
                                        dacclkn <= '0';
                                   presente <= eins;
                                   elsif b2='1' then
```

```
                    addr <= 0;
                        dcp <= '0';
                        dcn <= '1';
                        data<= memory2(addr);
                        data_p <= data;
                        data_n <= -data;
                        dacclkp <= '1';
                        dacclkn <= '0';
presente <= zwei;
                  elsif b3='1' then
                    addr <= 0;
                        dcp <= '0';
                        dcn <= '1';
                        data<= memory3(addr);
                        data_p <= data;
                        data_n <= -data;
                 dacclkp <= '1';
                        dacclkn <= '0';
presente <= drei;
                  elsif b4='1' then
                    addr <= 0;
                        dcp <= '0';
                        dcn <= '1';
                        data<= memory4(addr);
                        data_p <= data;
                        data_n <= -data;
                        dacclkp <= '1';
                        dacclkn <= '0';
presente <= vier;
          elsif gut='1' then
                        addr <= 0;
```

```
                presente <= hoffen;
            elsif r='1' then
               presente <= bekommen;
            end if;

    when eins => if addr < 255 then
                addr <= addr +1;
                data <= memory1(addr);
                data_p <= data;
                data_n <= -data;
                dcp <= not dcp;
                dcn <= not dcn;
                dacclkp <= dcp;
                dacclkn <= dcn;
                gut <= '0';
                presente <= eins;

            else
                gut <= '1';
                addr <= 0;
                data <= memory1(addr);
                data_p <= data;
                data_n <= -data;
                dacclkp <= '1';
                dacclkn <= '0';
                presente <= inicio;
            end if;

when zwei => if addr < 255 then
                addr <= addr +1;
                data <= memory2(addr);
```

```
                    data_p <= data;
                    data_n <= -data;
                    dcp <= not dcp;
                    dcn <= not dcn;
                    dacclkp <= dcp;
                    dacclkn <= dcn;
                    gut <= '0';
                    presente <= zwei;

                else
                    gut <= '1';
                    addr <= 0;
                    data <= memory2(addr);
                    data_p <= data;
                    data_n <= -data;
                    dacclkp <= '1';
                    dacclkn <= '0';
                    presente <= inicio;
                end if;

when drei => if addr < 255 then
                    addr <= addr +1;
                    data <= memory3(addr);
                    data_p <= data;
                    data_n <= -data;
                    dcp <= not dcp;
                    dcn <= not dcn;
                    dacclkp <= dcp;
                    dacclkn <= dcn;
                    gut <= '0';
                    presente <= drei;
```

```
                    else
                        gut <= '1';
                        addr <= 0;
                        data <= memory3(addr);
                        data_p <= data;
                        data_n <= -data;
                        dacclkp <= '0';
                        dacclkn <= '1';
                        presente <= inicio;
                    end if;

when vier => if addr < 255 then
                        addr <= addr +1;
                        data <= memory4(addr);
                        data_p <= data;
                        data_n <= -data;
                        dcp <= not dcp;
                        dcn <= not dcn;
                        dacclkp <= dcp;
                        dacclkn <= dcn;
                        gut <= '0';
                        presente <= vier;
                    else
                        gut <= '1';
                        addr <= 0;
                        data <= memory4(addr);
                        data_p <= data;
                        data_n <= -data;
                        dacclkp <= '0';
                        dacclkn <= '1';
                        presente <= inicio;
```

```
                                    end if;

            when hoffen => if konto > 0 then
                                                konto <= konto -1;
                                                presente <= inicio;
                else
                                                presente <= bekommen;
                                                gut <= '0';
                                    end if;

            when bekommen => if addr < 255 then
                                                CONVST_IN <= '1';
                                                datos(addr) <= data_in;
                                                addr <= addr+1;
                                                r <='1';
                                                presente <= inicio;
                                    else
                                                r<='0';
                                                presente <= inicio;
                                    end if;

        end case;
    end if;
    end process;
    end ram;
```

A.3 | UCF

```
#Relojes

net "fpga_clkc_p" TNM_NET =
"fpga_clkc_p";
TIMESPEC "TS_SYS_CLK_IN" =
PERIOD "fpga_clkc_p" 3.0 us;

net "fpga_clkc_n"          LOC =
p3;
net "fpga_clkc_p"          LOC =
r3;
net "fpga_clkc_n"
IOSTANDARD = LVDS_25;
net "fpga_clkc_p"
IOSTANDARD = LVDS_25;
net "fpga_clkc_n"
DIFF_TERM = FALSE;
net "fpga_clkc_P"
DIFF_TERM = FALSE;
#Buttons
#SW3 (Evaluation Board for the
Artix7)
NET "b1" LOC = P6;
NET "b1" IOSTANDARD =
LVCMOS15;

#SW5
NET "b2" LOC = T5;
NET "b2" IOSTANDARD =
SSTL15;
#SW4
NET "b3" LOC = U5;
NET "b3" IOSTANDARD =
SSTL15;
#SW6
NET "b4" LOC = U6;
NET "b4" IOSTANDARD =
SSTL15;

#leds
net "led1"                 LOC =
M26    ;
net "led2"                 LOC =
T24    ;
net "led3"                       LOC
=  T25    ;
net "led4"                 LOC =
R26    ;
net "led1"
IOSTANDARD = LVCMOS33  ;
net "led2"
IOSTANDARD = LVCMOS33  ;
net "led3"
IOSTANDARD = LVCMOS33  ;
```

```
net "led4"
IOSTANDARD = LVCMOS33  ;

#PORTS
#DAC

NET "data_p[0]" LOC = M14;
NET "data_n[0]" LOC = L14;
NET "data_p[1]" LOC = B22;
NET "data_n[1]" LOC = A22;
NET "data_p[2]" LOC = B19;
NET "data_n[2]" LOC = A19;
NET "data_p[3]" LOC = H16;
NET "data_n[3]" LOC = G16;
NET "data_p[4]" LOC = F18;
NET "data_n[4]" LOC = F19;
NET "data_p[5]" LOC = H14;
NET "data_n[5]" LOC = H15;
NET "data_p[6]" LOC = G25;
NET "data_n[6]" LOC = F25;
NET "data_p[7]" LOC = E26;
NET "data_n[7]" LOC = D26;
NET "data_p[8]" LOC = G24;
NET "data_n[8]" LOC = F24;
NET "data_p[9]" LOC = G22;
NET "data_n[9]" LOC = F22;
NET "data_p[10]" LOC = L17;
NET "data_n[10]" LOC = L18;
NET "data_p[11]" LOC = M16;
NET "data_n[11]" LOC = M17;
NET "data_p[12]" LOC = E21;
NET "data_n[12]" LOC = D21;
NET "data_p[13]" LOC = E20;
NET "data_n[13]" LOC = D20;

INST "data_p[0]" IOSTANDARD =
"LVCMOS18";
inst "data_n[0]" IOSTANDARD =
"LVCMOS18";
inst "data_p[1]" IOSTANDARD =
"LVCMOS18";
inst "data_n[1]" IOSTANDARD =
"LVCMOS18";
inst "data_p[2]" IOSTANDARD =
"LVCMOS18";
inst "data_n[2]" IOSTANDARD =
"LVCMOS18";
inst "data_p[3]" IOSTANDARD =
"LVCMOS18";
inst "data_n[3]" IOSTANDARD =
"LVCMOS18";
inst "data_p[4]" IOSTANDARD =
"LVCMOS18";
inst "data_n[4]" IOSTANDARD =
"LVCMOS18";
inst "data_p[5]" IOSTANDARD =
"LVCMOS18";
inst "data_n[5]" IOSTANDARD =
"LVCMOS18";
inst "data_p[6]" IOSTANDARD =
"LVCMOS18";
inst "data_n[6]" IOSTANDARD =
```

```
"LVCMOS18";
inst "data_p[7]" IOSTANDARD =
"LVCMOS18";
inst "data_n[7]" IOSTANDARD =
"LVCMOS18";
inst "data_p[8]" IOSTANDARD =
"LVCMOS18";
inst "data_n[8]" IOSTANDARD =
"LVCMOS18";
inst "data_p[9]" IOSTANDARD =
"LVCMOS18";
inst "data_n[9]" IOSTANDARD =
"LVCMOS18";
inst "data_p[10]" IOSTANDARD =
"LVCMOS18";
inst "data_n[10]" IOSTANDARD =
"LVCMOS18";
inst "data_p[11]" IOSTANDARD =
"LVCMOS18";
inst "data_n[11]" IOSTANDARD =
"LVCMOS18";
inst "data_p[12]" IOSTANDARD =
"LVCMOS18";
inst "data_n[12]" IOSTANDARD =
"LVCMOS18";
inst "data_p[13]" IOSTANDARD =
"LVCMOS18";
inst "data_n[13]" IOSTANDARD =
"LVCMOS18";

NET "data_p[0]" DIFF_TERM =
"TRUE";
NET "data_n[0]" DIFF_TERM =
"TRUE";
NET "data_p[1]" DIFF_TERM =
"TRUE";
NET "data_n[1]" DIFF_TERM =
"TRUE";
NET "data_p[2]" DIFF_TERM =
"TRUE";
NET "data_n[2]" DIFF_TERM =
"TRUE";
NET "data_p[3]" DIFF_TERM =
"TRUE";
NET "data_n[3]" DIFF_TERM =
"TRUE";
NET "data_p[4]" DIFF_TERM =
"TRUE";
NET "data_n[4]" DIFF_TERM =
"TRUE";
NET "data_p[5]" DIFF_TERM =
"TRUE";
NET "data_n[5]" DIFF_TERM =
"TRUE";
NET "data_p[6]" DIFF_TERM =
"TRUE";
NET "data_n[6]" DIFF_TERM =
"TRUE";
NET "data_p[7]" DIFF_TERM =
"TRUE";
NET "data_n[7]" DIFF_TERM =
"TRUE";
```

```
NET "data_p[8]" DIFF_TERM =
"TRUE";
NET "data_n[8]" DIFF_TERM =
"TRUE";
NET "data_p[9]" DIFF_TERM =
"TRUE";
NET "data_n[9]" DIFF_TERM =
"TRUE";
NET "data_p[10]" DIFF_TERM =
"TRUE";
NET "data_n[10]" DIFF_TERM =
"TRUE";
NET "data_p[11]" DIFF_TERM =
"TRUE";
NET "data_n[11]" DIFF_TERM =
"TRUE";
NET "data_p[12]" DIFF_TERM =
"TRUE";
NET "data_n[12]" DIFF_TERM =
"TRUE";
NET "data_p[13]" DIFF_TERM =
"TRUE";
NET "data_n[13]" DIFF_TERM =
"TRUE";
```

```
NET "dacclkp" LOC = D19;
NET "dacclkp" IOSTANDARD =
"LVCMOS18";
NET "dacclkp" DIFF_TERM =
"TRUE";
NET "dacclkn" LOC = C19;
NET "dacclkn" IOSTANDARD =
"LVCMOS18";
NET "dacclkn" DIFF_TERM =
"TRUE";

 # PlanAhead Generated physical
constraints
INST "XADc/XADC_INST" LOC =
XADC_X0Y0;
NET "VP_IN" LOC = N12;
NET "VN_IN" LOC = P11;

NET "VP_IN" iostandard =
LVCMOS25;
NET "VN_IN" iostandard =
LVCMOS25;
```

Bibliografía

BIBLIOGRAFIA

[1] George D. Ludwing, Francis W. Struthers

"Condiderations, Underlaying, the use of ultrasound to detect: gallstones and foreing bodies in tissue", Naval Research Laboratory Washington D.C, Report. (1949).

[2] Gerald R. Harris

"Progress in Medical Ultrasound Exposimetry", IEEE transactions on ultrasonics, ferroelectrics, and frequency control, vol. 52, no. 5 pp. 716-736 (2005).

[3] Fredric J. Harris

"*Multirate Signal Processing for Communication Systems",* Ed, Prentice Hall, (2004).

[4] IEEE Computer Society

"IEEE Standard for Floating-Point Arithmetic" IEEE (2008).

[5] Jian Shen, and Emad S. Ebbini

"A new coded-excitation Ultrasound Imaging System: Part: 1 Basic Principles" IEEE transactions on ultrasonics, ferroelectrics, and frequency control, vol. 43, no. 1, (1996)

[6] John G. Proakis, Dimitris G. Manolakis

"*Trataiento Digital de Señales",* ed. Prentice Hall (2007).

[7] Rama Pailoor and Dev Pradhan

"*Digital Signal Processor (DSP) for Portable Ultrasound",* Aplication Report Texas Instruments SPRAB18A (2008).

[8] Ali Tangel, Mehmet Yakut, Engin Afacan, Ulvi Güvenc, Hasan Sengül

"An FPGA-Based multiple-output PWM pulse generator for ultrasonic cleaning machines", Applied Electronics (AE), International Conference on 2010

[9] Zebadúa Augusto

"Diseño de un receptor Digital de señales RF mediante un FPGA" Facultad de Ingeniería, UNAM (2012).

[10] Aitken Michael

"Fpga Based Data Acquisition System for Ultrasound Tomograpgy" University of Cape Town (2006).

[11] Umar Alqasemi, Hai Li, Andrés Aguirre

"FPGA-Based Reconfigurable Processor for UltraFast Intelace Ultrasound and Photoacoustic Imaging", ÆEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control , vol. 59, no. 7, 1344-1353 (2012).

[12] Gi-Duck Kim, Changhan Yoon, Sang-Bum Kye, Youngbae Lee, Jeeun Kang, et al.

"A Single FPGA-Based Portable UltraSound Imaging System for Point-of-Care Applications ", IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 59, no. 7. pp 1386-1394(2012)

[13] **Weibao Qiu, Yanyan Yu, Fu Keung Tsang, Lei Sun**

"An FPGA-Based Open Platform for Ultrasound Biomicroscopy", IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 59, no. 7 pp 1432-1442 (2012).

[14] **Enrico Boni, Luca Bassi, Alessandro Dallai, et al.**

"A Reconfigurable and Programmable FPGA-Based System for Nonstandard Ultrasound Methods",EEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control , vol. 59, no. 7, pp 1378-1285 (2011).

[15] **Amauri Amorin Assef, Joaquim Miguel Maia, et al.**

"A Programmable FPGA-based 8-Channel Arbitrary Waveform Generator for Medical Ultrasound Research Activities", 34th Annual International Conference of the IEEE EMBS San Diego, California USA, pp 515-518 (2012).

[16] **Sheng-Wen Huang, Pai-Chi Li**

"Ultrasonic Computed Tomography Reconstruction of the Attenuation Coefficient Using a Linear Array", IEEE (2005).

[17] **Jon Alexander**

"Xilinx FPGAs in Portable Ultrasound Systems", Xilinx White Paper: 7 Series, Virtex-6, and Spartan-6 FPGAs (2012).

[18] **Facultad de Ingeniería Venezuela**

"Cursoo Básico de Ultrasonido", **Facultad de Ingeniería Venezuela http://goo.gl/gRtunb** (2012).

[19] Olympus

"Ultrasonic Transducers, Technical Notes" Olympus pp 40-49 (2011).

[20] Xilinx

"Constraints Guide " V 13.4 Xilinx (2012)

[21] Hong-Swee Lim

"High Performance DSP Solutions for Ultrasound" Xilinx (2008)

[22] Elsa C. Raslwski

"Computed Tomography. A great source of radiation exposure" Arch Argent Pediatr pp 273,274 (2008).

[23] David J. Brenner, Eric J. Hall, D.Phil

"Computed Tomograph: An Increasing Source of Radiation Exposure" New England Journal of Medicine, Massachusetts Medical Society. pp 2278-2284 (2007).

[24] https://www.nde-ed.org/index_flash.htm (2014)