INSTITUTO TECNOLOGICO DE TUXTLA GUTIERREZ

REPORTE FINAL DE RESIDENCIA PROFESIONAL

CARRERA

INGENIERIA ELECTRONICA

TITULO:

DISEÑO DE UN ELECTROCARDIÓGRAFO DIGITAL USANDO COMO PLATAFORMA EL KIT DE DESARROLLO SPARTAN-3E DE XILINX

RESIDENCIA PROFESIONAL QUE PRESENTA:

MARCOS RAMON ARCHILA BONIFAZ

ASESOR INTERNO:

DR. RUBEN HERRERA GALICIA

TUXTLA GUTIERREZ CHIAPAS, JUNIO 2009

DATOS PERSONALES

Nombre de la institución: **Instituto Tecnológico de Tuxtla Gutiérrez**

Nombre del residente: **Marcos Ramon Archila Bonifaz**

Carrera: **Ingeniería Electrónica**

Numero de control: **04270193**

Asesor interno: **Dr.Ruben Herrera Galicia**

Revisor de residencia profesional: **M. en C.Arnulfo Cabrera Gomez.**

DATOS DE LA EMPRESA

Nombre de la empresa: **Instituto Tecnológico de Tuxtla Gutiérrez.**

Nombre del proyecto: **Diseño De Un Electrocardiógrafo Digital Usando Como Plataforma El Kit De Desarrollo Spartan-3E De Xilinx.**

Giro, ramo o sector: **Publico.**

Domicilio: **Carretera Panamericana km.1050**

Ciudad: **Tuxtla Gutiérrez**

RFC: **SEP210905778**

Nombre del titular de la empresa: **M en C. Tomas Palomino Solorzano**

INDICE GENERAL

El trabajo aquí descrito esta enfocado a diseñar un sistema capaz de adquirir señales bioeléctricas provenientes del corazón de un ser humano y procesarlas por medio de un FPGA obteniendo así una señal con la menor interferencia y ruido posibles que garanticen un registro confiable de la intensidad y duración de los impulsos del corazón y que pueda ser transportado por un medio de comunicación para finalmente ser visualizado en el monitor de una computadora.

1.1 INTRODUCCION.

Un electrocardiograma (ECG) es un registro gráfico de los potenciales eléctricos producidos por el tejido cardiaco. El corazón es especial entre los músculos del cuerpo en vista de que posee la propiedad de la generación de un impulso eléctrico automático. Este impulso provoca la excitación de las fibras musculares y resulta en la contracción cardiaca. La conducción de este impulso origina pequeñas corrientes eléctricas que se propagan a todo el cuerpo. El ECG se obtiene colocando electrodos en varios sitios de la superficie del cuerpo que se conectan a un aparato de registro.

Un electrocardiograma consiste en una serie de formas de onda que ocurren en un orden repetido. Estas formas de onda provienen de una línea de fondo plana llamada línea isoeléctrica. Cualquier desviación de esta línea indica actividad eléctrica. Un ciclo de corazón es representado por un grupo de cinco desviaciones principales designadas por las letras P, Q, R, S, T. La onda P representa la despolarización del atria y es asociada con su contracción. La primera desviación negativa es la onda Q y es seguida de una desviación positiva llamada onda R. El complejo se termina con una desviación negativa conocida como onda S. El complejo QRS denota la despolarización de los ventrículos y esta asociado con su contracción. Algunos tiempos típicos para las partes del ECG son: P-R (0.2 s), QRS (0.1 s), Q-T (0.38 s).

Para interpretar un electrocardiograma se recomienda tomar una serie de

medidas básicas: la frecuencia cardiaca, ritmo, eje del complejo QRS en el plano frontal, intervalo PR, duración del QRS. Con esta secuencia de lectura se detecta prácticamente cualquier alteración electrocardiográfica.

Para implementar el electrocardiógrafo digital propuesto se usara el kit de desarrollo Spartan 3E de xilinx. El sistema propuesto se compone de tres parches de electrodo para adquisición de la señal ECG de la persona, un amplificador de instrumentación basado en el AD624 con amplificación de 1000 V/V, dos filtros analógicos, un kit de desarrollo Spartan 3E, una interfase de comunicación USB- RS232, y una PC. El diseño completo del sistema se compone de cinco unidades: sensado, amplificación, digitalizado, filtrado, y comunicación. Un diagrama a bloques del sistema se presenta en la Fig.1.

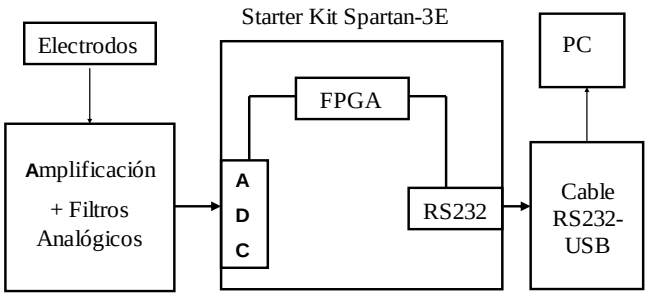


Fig. 1 Electrocardiógrafo digital; Diagrama a bloques.

2.1 MARCO TEORICO

1) Programación del FPGA. El lenguaje usado para programar al FPGA es el lenguaje de descripción de hardware VHDL. Este lenguaje es utilizado para describir circuitos en un nivel alto de abstracción y ha sido aceptado como un medio estándar de diseño. El lenguaje VHDL está creado específicamente para el diseño de hardware, se pueden implementar con él circuitos lógicos, tanto combinacionales como secuenciales. Éste lenguaje también permite describir elementos más complejos, como CPU y retrasos de tiempo.

Un programa en VHDL consta de dos partes. La entidad que sirve para relacionar al diseño con el mundo exterior. Aquí se declara lo que se va a crear como una caja negra, de la que se especifican sus entradas y sus salidas. En la segunda parte, la arquitectura, se describe como trata el circuito la información correspondiente a las entradas para obtener las salidas. Dentro de la arquitectura pueden haber subprogramas, cada uno cumpliendo una función y todos ellos gobernados por un bloque principal. Los subprogramas en su funcionamiento pueden ser secuénciales o concurrentes. La estructura de programación es similar a la de los lenguajes de mediano nivel: C, ada. Usa declaraciones de: tipos de datos, variables, y arreglos. También usa estructuras de selección (if, case), y estructuras de repetición (for, while). Los puertos de las entidades se definen siempre para los tipos std_logic y std_logic_vector. Las operaciones aritméticas estándares solo están definidas para los tipos signed y unsigned.

En VHDL la concurrencia consta de un conjunto de procesos ejecutándose asincrónicamente y comunicándose mediante una red de señales. En el interior de un proceso las acciones se suceden en el orden secuencial marcado por las construcciones, pero el tiempo avanza solo hasta que finaliza el proceso.

2) Filtros digitales. En lo básico de su funcionalidad un filtro digital se comporta de igual manera que un analógico. Un filtro es un sistema encargado de alterar el contenido de la información espectral de una señal de entrada X(t), produciendo una señal de salida Y(t). Los filtros analógicos son implementados mediante el uso de dispositivos discretos que operan sobre formas de onda continuas. Los filtros digitales son implementados mediante el uso de circuitos lógicos o en programas de computador que operan sobre secuencias de números que son obtenidos por el muestreo de ondas continuas.

Los filtros digitales tienen un extendido uso gracias a la facilidad que se presenta para montar estos diseños en dispositivos de procesamiento digital DSP. En otros casos son diseñados e implementados en circuitos lógicos programables como el FPGA.

Los filtros digitales tienen ventajas sobre los filtros análogos y estas son algunas de ellas: Un filtro digital en si mismo no introduce ruido, gracias a la forma en que se implementa (Software o Circuito digital). La precisión de un filtro digital depende del error de redondeo, el cual esta determinado por el número de bits usado para representar a las variables en el filtro. Es fácil y barato cambiar las características de operación del filtro. Esto a diferencia de los filtros analógicos donde se requiere una reconstrucción del hardware. Además su desempeño no esta en función del deterioro de sus componentes, de las variaciones de la temperatura o de las variaciones del voltaje de la fuente de alimentación.

Esta clase de sistemas se encuentra caracterizada por una ecuación lineal en diferencias con coeficientes constantes, ec. (1). Mediante la transformada Z se pueden caracterizar como una función de transferencia racional, ec. (2).

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$

ec. (1)

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}$$

ec. (2)

En las ecuaciones (1) y (2) los polos y los ceros del sistema se encuentran dados por los {bk} y {ak}, los cuales determinan las características de la respuesta en frecuencia del sistema. Existen dos tipos básicos de filtros digitales, los No-recursivos y los recursivos. Los filtros No-recursivos se caracterizan por no poseer realimentaciones de lo cual se aprecia que la salida se encuentra dada solo en función de la entrada actual y las entradas pasadas, ec. (3).

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k)$$

ec. (3)

$$H(z) = \sum_{k=0}^{M-1} b_k z^{-k}$$

ec. (4)

Para los filtros recursivos la ecuación en diferencias se encuentra expresada como una suma de dos funciones polinomiales, ec. (5), la cual lleva a encontrar una función de transferencia dada por la ec. (6).

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$

ec. (5)

$$H(z) = \frac{\sum_{k=0}^{M} a_k z^{-k}}{1 - \sum_{k=1}^{N} b_k z^{-k}} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}{1 - b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n}}$$

ec. (6)

A los primeros pertenecen los filtros tipo FIR, los cuales se caracterizan por no poseer realimentación y a los segundos los filtros tipo IIR en los cuales la salida se encuentra dada en función de la entrada actual y pasadas y de las salidas pasadas.

3.1 JUSTIFICACIÓN

En México existen deficiencias relacionadas con equipo de diagnostico de enfermedades que den una respuesta eficaz, a precio competitivo, y que sea transportable. Las patologías cardiacas son de las principales causas de

deceso en la población de edad adulta, pueden evitarse muchas de estas muertes si se detectan a tiempo estos problemas cardiacos.

Un ECG tiene significado de diagnóstico. Un paciente con enfermedad cardiaca tiene un ECG anormal y un individuo normal puede tener un ECG anormal. No obstante, un paciente puede recibir la seguridad garantizada de que no tiene padecimiento cardiaco en base exclusivamente de un ECG normal.

Un FPGA ofrece ventajas muy favorables para el diseño y desarrollo de un electrocardiógrafo: flexibilidad, capacidad de procesado en paralelo y velocidad. Entre esas ventajas resalta la flexibilidad, cualidad inherente a los circuitos de lógica programable, que permite hacer cambios en el funcionamiento y hacer otro procesamiento digital de la señal sin cambiar el hardware.

El desarrollo del presente electrocardiógrafo se hace con el propósito de ayudar a la comunidad médica a contar con un aparato que dé una respuesta eficaz, a precio competitivo, que sea transportable y capaz de auxiliar al especialista en la elaboración de un diagnóstico acertado.

Este proyecto es importante porque es una herramienta útil para diagnosticar enfermedades cardiacas. Con su desarrollo se esta colaborando con el sector salud. Y tiene la ventaja de que al utilizar un FPGA permite hacer modificaciones al diseño sin tener que cambiar el hardware.

4.1OBJETIVOS

Objetivo general

Implementar un sistema para adquirir señales bioeléctricas provenientes del corazón de un ser humano con bajo nivel de ruido e interferencia para procesarlas por medio de un FPGA obteniendo una señal que garantice un registro confiable de la intensidad y duración de los impulsos del corazón que sea transportable por una interfase de comunicación RS232-USB para finalmente ser visualizada en el monitor de una computadora.

Objetivos específicos

- Implementar un electrocardiógrafo digital usando para esto el kit de desarrollo Spartan 3E de Xilinx.

5.1 DISEÑO DEL ELECTROCARDIÓGRAFO.

1) Unidad de adquisición analógica. Debido a que la corriente que genera el cuerpo humano es del orden de milivolts se debe tener extrema precaución en esta etapa ya que es muy fácil que se presenten ruidos no deseados junto con la señal útil. Las probables fuentes de ruido en un ambiente de hospital son: la red eléctrica, las lámparas fluorescentes, los motores eléctricos u otro equipo médico.

Para la adquisición de la señal ECG de la persona se usan tres parches de electrodo. Se coloca firmemente el primer electrodo sobre la parte interna del tobillo derecho. Un segundo electrodo a un costado de la costilla izquierda, a la altura del codo izquierdo. Un tercer electrodo sobre el corazón, sobre los músculos del ventrículo derecho. Se conectan los caimanes del cable a los conectores de los parches de los electrodos. Se conecta el caimán negro, la referencia, al tobillo derecho. Esto es el punto de referencia para la línea isoeléctrica. Se conecta el caimán verde, negativo, a la costilla izquierda. El caimán rojo, positivo, al corazón.

2) Unidad de amplificación. Para amplificar la señal proveniente de los electrodos se pretende usar el amplificador de instrumentación AD624 con una amplificación total aproximada de 1000 V/V, en combinación con un filtro analógico pasabanda de 0.05 Hz a 100 Hz, de segundo orden, y un filtro rechaza banda de 60 Hz. Para acoplar el voltaje que entregara el sensor (Offset = 1V) al voltaje de 1.65v, requerido como nivel cero por el convertidor ADC del Kit Spartan 3E, se agregara un amplificador con ganancia 1.65 V/V.

3) Unidad de digitalizado. En esta parte se convierte la señal analógica del cuerpo (voltaje) a una palabra digital de 14 bits con signo en notación de complemento a dos. Esto se hace mediante el convertidor analógico - digital incluido en el LTC1407A-1 del kit de desarrollo Spartan 3E. Este ADC convierte voltajes de ±1.25v relativos a 1.65v.

4) filtrado digital. En esta etapa se pretende mejorar la señal aplicando distintos tipos de filtros; pasa altas (0.05 Hz), rechaza banda (60Hz) y pasa bajas (150Hz). El objetivo de éstos es eliminar frecuencias menores a 0.1Hz, mayores a 150Hz y el ruido de la fuente de alimentación. La frecuencia de cualquier pulso cardiaco en reposo no sobrepasa los 20 Hz, y así se evitaran señales extrañas provocadas hasta por los mismos circuitos.

Aquí se presenta un ejemplo de un programa con un filtro pasa bajas. Durante el desarrollo del proyecto se pretende implementar y probar otros filtros. Para la programación se usa lenguaje VHDL para ser descargada al FPGA Spartan 3E del Kit de desarrollo, usando para esto el entorno de programación ISE 10.1.

5) Unidad de comunicación. La comunicación del kit con la computadora será a través del puerto serie RS232 usando para esto un cable conversor de puerto serie RS232 a USB.

6.1 PROGRAMACIÓN

El software que utilizamos es el el xilinx 10.1,de xilinx,en el cual utilizamos el lenguaje VHDL el cual nos facilita nuestra programación, después un ejemplo de cómo utilizar correctamente nuestro programa xilinx 10.1,luego una explicación del programa que diseñamos el cual controla el amplificador y adc del Spartan 3e.

EJEMPLO DE LA PROGRAMACIÓN EN EL FPGA SPARTAN 3E.

Utilizando el xilinx 10.1, y tomando como ejemplo una compuerta or.

1- abrir la interfaz grafica xilinx ise 10.1.
2- limpiar los apartados

file- close project

3- crear un project

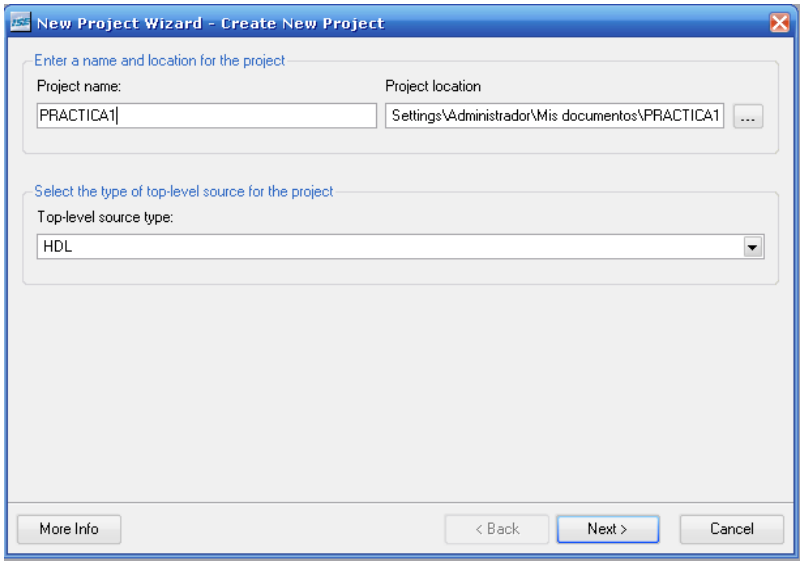
3.1 file- new project

3.2 dar nombre al projecto(no debe haber espacios)

3.3 indicar la ruta. project location.

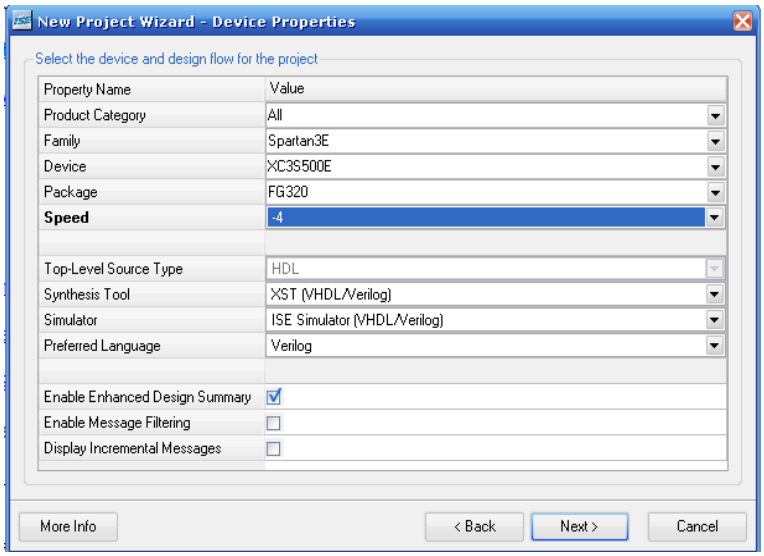
3.4 seleccionar el tipo de descriptor top-level source type.

3.5 next



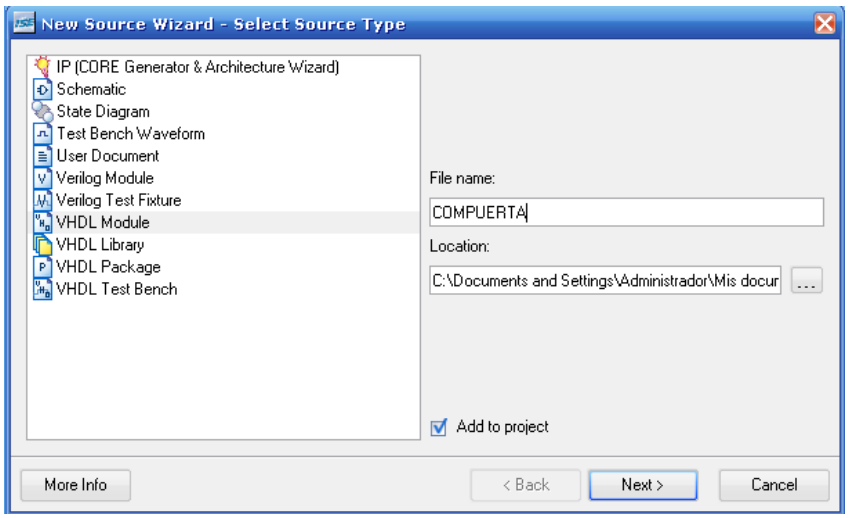
4.seleccionar los siguientes parametros que son los indicados para trabajar con el spartan 3e (xc3s500e).

4.1 next



4.2 seleccionar new source

4.3 seleccionar vhdl module y dar nombre al file name:



4.4 next

5. asignar los puertos, las e/s, los nombres de la identidad y arquitectura.

5.1 escribir nombres y definir su tipo i/o.

en el ejemplo a y b son entradas y q unica salida.

5.2 next.

5.3 finish.

5.4 yes.

5.5 next.

5.6 next.

5.7 finish.

6- seleccionar compuerta.vhd

6.1 colocar el codigo de programacion en la arquitectura.

en este caso es la de una compuerta or.

q<= a or b;

(entre begin y end behavioral)

6.2 guardar los cambios.

file-save all.

7. revisar el codigo.

processes:

synthesize.xts- (con doble click).

7.1 yes.

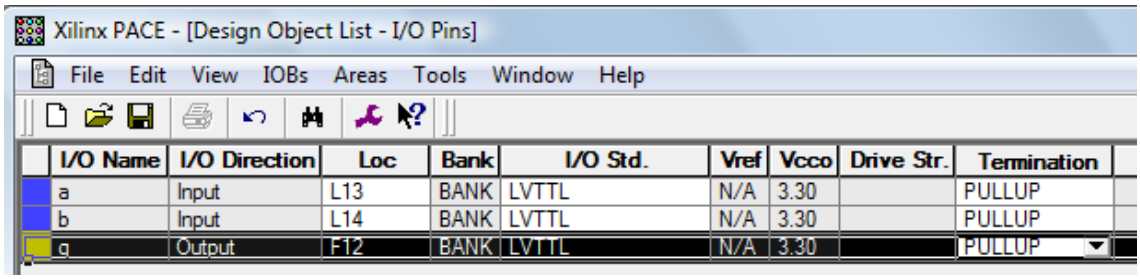
8. Asignar pines.

8.1 processes-user constraints-floorplan area/io/logic- post synthesis.

8.2 yes (agrega archivo .ucf)

8.3 una vez abierta la ventana del archivo .ucf,en desing object list i/o pins, escribir los siguientes parámetros.

en este proyecto utilizamos los puertos sw0(l13) y sw1(l14) como entrada y led0(f12) como salida.



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination
a	Input	L13	BANK	LVTTL	N/A	3.30		PULLUP
b	Input	L14	BANK	LVTTL	N/A	3.30		PULLUP
q	Output	F12	BANK	LVTTL	N/A	3.30		PULLUP

8.4cerrar ventana.

8.5 yes(guarda cambios).

9. cargar el programa en el kit.

9.1se conectan los cables al kit; alimentacion y usb.

9.2 se conecta el kit con la pc(cable usb).

9.3 se enciende la alimentacion del kit.

9.4seleccionar configure target device(con doble click).

9.5ok

9.6 finish

9.7abrir el archivo compuerta.bit

9.8cancel.

9.9cancel.

9.10ok.

9.11 seleccionar program(doble click)

¡listo!

El programa ahora esta cargado en el spartan 3e y nos daremos cuenta que se enciende un led en amarillo indicando que se acaba de cargar un programa.

al apagar el spartan 3e se borra el programa.

7.1 ADC Y AMPLIFICACION .

El Spartan 3e FPGA incluye dos canales de circuito de registro analógico, consistiendo en un pre-amplificador y un adc, consiste en una tecnología lineal ltc6912-1 .

Salida digital para entradas analógicas.

Los circuitos de captura analógica convierten el voltaje analógico de vina o vinb y lo convierte en una representación digital de 14 bits expresado por la siguiente ecuación.

$$D[13:0] = GAIN \times (\frac{Vin - 1.65V}{1.25}) \times 8192$$

Gain = ganancia cargada al amplificador.

El voltaje de referencia para el amplificador y el adc es 1.65v,generado por un divisor de voltaje, consecuentemente,1.65v es adquirido de la entrada de voltaje vina o vinb.

El rango máximo del adc es 1.25v, centrado con la referencia de 1.65v,por ello 1.25v aparece en el denominador de la escala en la entrada analógica correspondiente.

Finalmente, el adc presenta 14 bits, en complemento a 2 digitales en la salida, un bit 14, de dos complementos representa un numero entre -8192 y 8191, de ahí que la cantidad es escalada por 8192,o 2 ala 13.

7.1.2 Amplificador .

El LTC6912-1 contiene dos amplificadores independientes invertidos con ganancia programable. El propósito del amplificador es servir como escalador para la entradas de voltaje vina o vinb, así que el máximo rango de conversión del dac es 1.65 +-1.25v.

Interfaz

La siguiente tabla representa la lista de señales de interfaz entre el FPGA y el amplificador. el spi_mosi,spi_miso y spi_sck son señales compartidas con otros dispositivos en el spi bus, el amp_cs es una señal de activacion low seleccionada a la entrada del amplificador.

SEÑAL	PIN FPGA	DIRECCION	DESCRIPCION
spi_mosi	t4	fpga->amp	Dato serial: maestro a esclavo del amplificador.
amp_cs	n7	fpga->amp	Activo en bajo, chip seleccionado el amplificador es activado cuando retorna alto.
spi_sck	u16	fpga->amp	reloj
amp_shdn	p7	fpga->amp	activación, reset

amp_dout	e18	fpga<-amp	Serial data: previas repeticiones ala activación de la ganancia.

Ganancia programable.

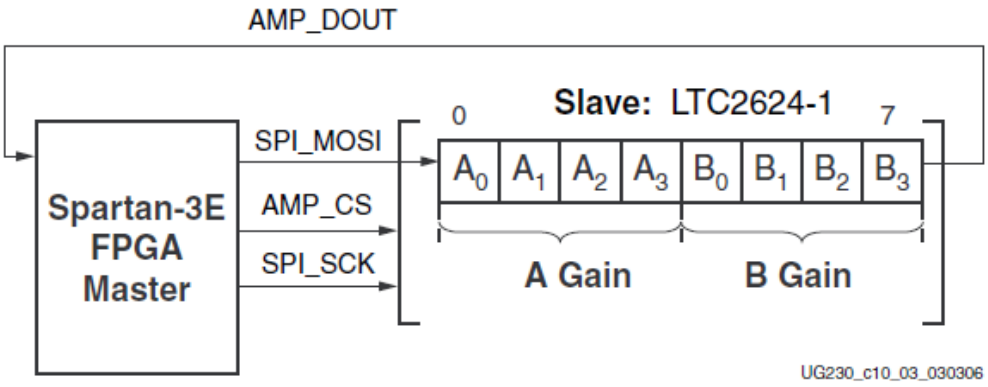
Cada canal analógico tiene asociado una ganancia de amplificación programable, señales analógicas presentadas a las entradas vina o vinb en header j7 son amplificados relativos a 1.65v, que son referencia generado por un divisor de voltaje de 3.3v.

La ganancia de cada pre-amplificador es programable de -1 a -100. Como se muestra.

ganancia	a3	a2	a1	a0	rango de voltaje en entrada	
	b3	b2	b1	b0	mínimo	máximo
0	0	0	0	0		
-1	0	0	0	1	0.4	2.9
-2	0	0	1	0	1.025	2.275
-5	0	0	1	1	1.4	1.9
-10	0	1	0	0	1.525	1.775
-20	0	1	0	1	1.5875	1.6625
-50	0	1	1	0	1.625	1.675
-100	0	1	1	1	1.6375	1.6625

Interfaz de control spi.

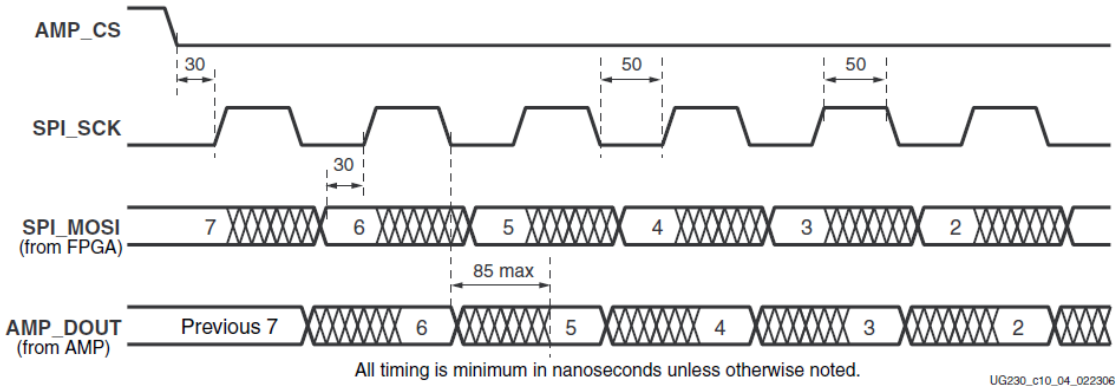
En la siguiente figura destaca el spi basado en la interfaz de comunicación con el amplificador.la ganancia para cada amplificador es enviada como un comando de 8 bits, consistiendo de archivos de 4 bits. El bit mas significativo,b3 es enviado primero.



El amp_dout salida del amplificador repite las asignaciones previas de ganancia, estos valores pueden ser ignorados para más aplicaciones.

El spi bus de transacción comienza cuando el FPGA declara amp_cs estado bajo. Los amplificadores capturan información serial sobre spi_mosi en el cambio a nivel alto del spi_sck.

Los amplificadores presentan información serial sobre la señal amp_dout en el cambio a nivel bajo de spi_sck.



Ubicación ucf

```
NET "SPI_MOSI" LOC = "T4"  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
NET "AMP_CS"   LOC = "N7"  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
NET "SPI_SCK"  LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "AMP_SHDN" LOC = "P7"  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
NET "AMP_DOUT" LOC = "E18" | IOSTANDARD = LVCMOS33 ;
```

7.1.2ADC

El LTC1407A-1 contiene dos adc, ambas entradas analógicas son simultáneamente muestras cuando el ad_conv es aplicado.

Interfaz

La siguiente tabla muestra las señales de interfaz entre el FPGA, y el adc.el spi_mosi,spi_miso,y spi_sck son señales compartidas con otros dispositivos en el spi bus. La señal dac_cs es la activacion baja enclavada en la entrada al dac.el dac_clr es activado bajo, asincrónicamente reset entrada al dac.

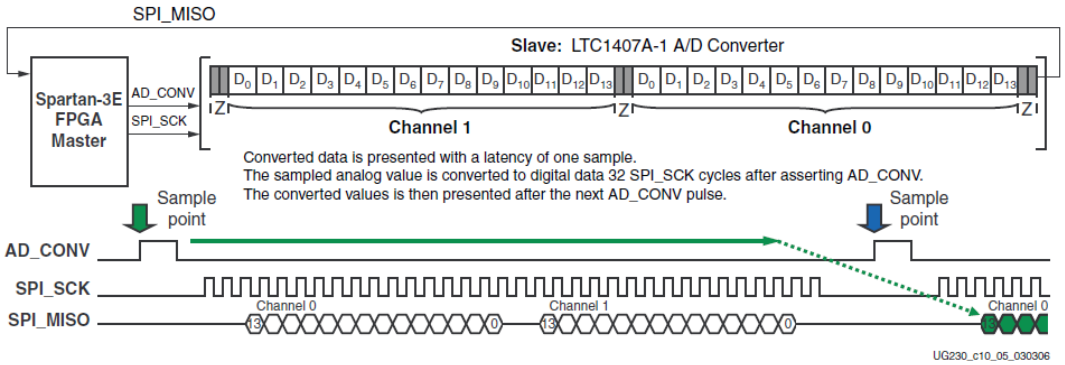
SEÑAL	PIN FPGA	DIRECCION	DESCRICCION
spi_sck	u16	fpga->adc	Reloj
ad_conv	p11	fpga->adc	activación alto,reset
spi_miso	n10	fpga<-adc	Información serial: master input, serial output, presenta la representación digital de los valores de muestras analógicas como dos complementos de 14 bits binarios.

Control de interfaz de spi

La siguiente figura contiene un ejemplo de spi bus transacción al adc.

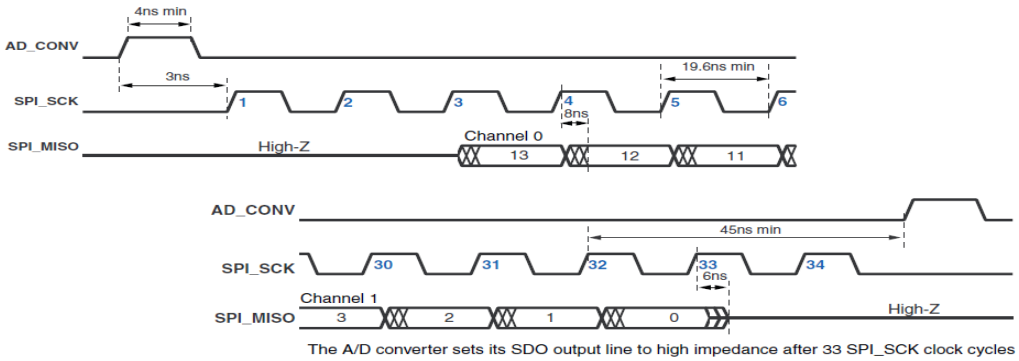
Cuando el ad_conv esta en nivel alto, el adc simultáneamente muestra ambos canales analógicos. Los resultados de la conversión son presentadas hasta que el ad_conv es asignado, un retrasa miento de una muestra, la máxima razón de muestra es 1.5 MHz.

El adc presenta la representación digital de lo valores de la muestra analógica con 14 bits, de complemento a 2 de valor binario.



La siguiente figura muestra a detalle los tiempos de transaccion.la señal

ad_conv no es un spi tradicional de enclavamiento de activacion.estar seguro de proveer suficientes spi_sck ciclos de reloj así que el adc abandona la señal spi_miso en alto –estado de impedancia. de otro modo, el bloque adc comunica al otro spi.usa 34 ciclos comunicando secuencias. el adc 3 estado es información de salida para dos ciclos de reloj antes y después de cada bit 14 de transferencia.



Ubicación del ucf

```
NET "AD_CONV"  LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
NET "SPI_SCK"  LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "SPI_MISO" LOC = "N10" | IOSTANDARD = LVCMOS33 ;
```

Deshabilitado de otros dispositivos en el spi bus para evitar conflictos de comunicación.

Las señales del spi bus son compartidas por otros dispositivos en la placa. Es importante que otros dispositivos sean deshabilitados cuando el FPGA se comunica con al amplificador o adc para evitar argumentación en el bus.la siguiente tabla contiene las señales y valores lógicos requeridos para deshabilitar cierto dispositivo. Aunque el strataflash PROM es un dispositivo paralelo, son bits menos significativos compartidos con la señal spi_miso. El PLATFORM flash PROM es solo potencialmente activado si el FPGA es cargado para modo de configuración serial maestro.

8.1 RESULTADOS.

PROGRAMA IMPLEMENTADO.

En nuestro programa manejamos una maquina de estados ya que como

estamos trabajando con datos seriales, tenemos que hacer la lectura y envió bit por bit.

Lo primero es declarar las señales que nos servirán en nuestro programa, para obtener el funcionamiento ideal del amplificador como el del adc, las señales del amplificador son:

SPI_MOSI-> salida, dato serial de ocho bits.

AMP_CS->salida,activo en bajo, chep select.

SPI_CLK->salida,reloj.

AMP_SHDN->salida,activo en alto,reset de amplificador.

SEÑALES DEL ADC:

AD_CONV-> salida,apagado y reset en alto del adc.

clk50->entrada,reloj de 50mhz del reloj del fpga.

SPI_MISO->entrada,dato serial representado en 14 bit de complemento a 2.

LUEGO DECLARAMOS LA SEÑAL PARA LA MUESTRA DE CONVERSION.

data->salida a 8 led's.

Las siguientes señales tienen que ser declaradas y después deshabilitadas para el funcionamiento ideal del spi bus.

SPI_SS_B->salida,spi serial flash.

DAC_CS->salida, dac.

SF_CEO->salida,strataflash parallel flash prom.

FPGA_INIT_B->salida,platform flash prom.

AHORA DECLARAREMOS LAS SEÑALES VARIABLES DE ENTRADA EN NUESTRO PROGRAMA.

rstpre->entrada,reset del amplificador.

rstadc->entrada, reset del adc.

onpre->entrada,activacion del amplificador.

sel1->entrada,seleccionador de canal.

sel2->seleccionador de nivel.

Después dentro de la arquitectura behavioral declaramos las siguientes señales variables que no son entradas ni salidas del spartan 3e, sino variables a utilizar en nuestro programa.

signal cntclk : integer range 0 to 50000:=0;

signal clk : std_logic:='0'

signal edo : integer range 0 to 120:=0

constant gain : std_logic_vector (7 downto 0):="00010001;

signal canal1,canal0,c1,c0: std_logic_vector (13 downto 0):="00000000000000;

Bueno después de begin empezamos a codificar nuestro programa, lo primero es deshabilitar señales en el SPI BUS que pueden interferir en el funcionamiento del amplificador y adc.

A continuación empezamos con el primer proceso el cual activa el reloj de 50mhz del FPGA.Luego hacemos un proceso (onpre) el cual activa al amplificador.

Después tenemos el proceso preampli en el cual va incluido el reset tanto del amplificador como el adc, y con una maquina de estados empezamos

haciendo la lectura de amplificación a través del spi_mosi, recordemos que el spi_mosi hace la lectura del amplificador en el flanco de subida, por eso la lectura se hace después de cada flanco de caída del reloj.

Una vez terminada la lectura del amplificador, nos encontramos en el estado 17 el cual empieza el proceso del adc, lo primero es activar el adc y esperar dos pulsos de reloj antes y después de 14 bits, para empezar la conversión en 14 bits al canal 0 y después al canal 1.

Por último tenemos un proceso el cual nos da la posibilidad de elegir nuestra muestra, en los 8 led's, el sel1 nos sirve para seleccionar el canal ya sea el canal1 o canal 0 y el sel2 su nivel alto o bajo, haciendo énfasis en que el nivel bajo representa los 8 bit menos significativos y el nivel alto los 8 bits más significativos.Luego damos fin a nuestro proceso y arquitectura.

Programa.

```
library IEEE;                          --declaración de las librerías para la ejecución del programa.
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adc is                          --Declaración de la entidad del programa.
Port ( spi_mosi : out  STD_LOGIC;      --Dato serial, 8 bits, Maestro a esclavo del amplificador
amp_cs : out  STD_LOGIC;               --Activo en bajo, Deshabilitado en 1 del amplificador
spi_clk : out  STD_LOGIC;              --reloj de síncrono de convertidor y amplificador.
amp_shdn : out  STD_LOGIC;             -- Activo en alto; Reset del amplificador
ad_conv : out  STD_LOGIC;              --Apagado y reset en alto del convertidor
```

```
clk50 : in std_logic;                    --reloj de  50 MHz del FPGA.

spi_miso : in  STD_LOGIC;                -- Dato serial representado en 14 bits
de complemento a 2.

data : out std_logic_vector (7 downto 0);        --Salida a LED's

SPI_SS_B    : out STD_LOGIC;  --mugrero FPGA ; Deshabilitado en 1

DAC_CS      :       out STD_LOGIC;    --mugrero FPGA ; Deshabilitado en 1

SF_CE0 : out STD_LOGIC;                  --mugrero FPGA ; Deshabilitado en 1

FPGA_INIT_B : out STD_LOGIC;             --mugrero FPGA ; Deshabilitado en
1

rstpre: in std_logic;                    --reset del preamplificador.

rstadc: in std_logic;                    --reset del adc.

onpre: in std_logic;                     --activacion del amplificador.

sel1,sel2: in std_logic);                --seleccionar canal y 8 bits mas o
menos significativos.

end adc;


architecture Behavioral of adc is        --declaracion de la arquitectura.

--declaración de señales

signal cntclk : integer range 0 to 50000:=0;--declaracion de variable tipo entera.

signal clk : std_logic:='0';             --declaracion de variable tipo bit.

signal edo : integer range 0 to 120:=0;      -- declaracion de variable tipo entera.


constant gain : std_logic_vector (7 downto 0):="00010001";-- declaración de
variable tipo vector de 8 bits,temporal de la ganancia del amplificador.(ganancia
de -1).

signal canal1,canal0,c1,c0: std_logic_vector (13 downto
0):="00000000000000";-- declaración de variable tipo vector de 14bits, temporal
del canal1,canal0,c1 y c0.


Begin – inicio de la arquitectura.


SPI_SS_b<='1';   --desabilitado en 1.
```

```
dac_cs<='1'; --desabilitado en 1.
sf_ce0<='1'; --desabilitado en 1.
fpga_init_b<='1'; --desabilitado en 1.

reloj2k: process(clk50)      --se incluye en la lista sensible a clk50.
Begin                        --inicio del proceso reloj2k.
        if clk50='1' and clk50'event then  --condición para el pulso de subida del reloj  de 50MHz.
                if cntclk<250 then          --condición para cuando la variable cntclk sea menor a 250.
                        clk<= not clk;--se asigna la negacion a clk.
                        cntclk<=cntclk+1; -- incrementamos 1 a cntclk .
                else
                        cntclk<=0;       -- de lo contrario se le asigna 0 a cntclk.
                end if;
        end if;
end process reloj2k;                    --fin del proceso reloj2k

process(onpre)
begin                   --inicio del process
        if onpre='1' then  -- condicion para activar el onpre e iniciar.
                amp_cs<='0';          --activo en bajo
        else
                amp_cs<='1';
        end if;

end process;  -- fin del process.

preampli: process(clk,rstpre,rstadc)  ---Pre-Amplificador reloj; reloj de subida
begin                                   --incio del proceso preampli.
```

```
if rstpre='1' then                    --condicion para resetear el amplificador.
        edo<=0;                       --se le asigna a la variable edo el valor
de 0.
        amp_shdn<='1';                --se activa en alto, reset del amplificador.
        spi_clk<='0';                 --se pone a 0 el reloj síncrono.
else
        if rstadc='1' then      --condicion para resetear el convertidor.
                edo<=17;        --se le asigna a la variable edo el valor de 17.
        else
        if clk='1' and clk'event then        --condición para cuando haya un pulso
de subida del reloj.
                case edo is           --inicia maquina de estados.
                        when 0 =>          --inicia primer estado.
                                amp_shdn<='0';        -- inicia nueva lectura del
amplificador.
                                spi_clk<='0';         - -reloj se le asigna 0.
                                spi_mosi<=gain(7);    --inicia lectura del  bit 7 al spi
mosi.
                                edo<=1;               -- edo se le asigna el valor
de 1.
                        when 1 =>
                                spi_clk<='1';         --flanco de subida de reloj.
                                edo<=2;               -- estado 2.
                        when 2 =>
                                spi_clk<='0';         --flanco de caida de reloj.
                                spi_mosi<=gain(6);  --lectura del bit 6 al spi mosi.
                                edo<=3;               -- estado 3.
                        when 3 =>
                                spi_clk<='1';         --flanco de subida de reloj.
                                edo<=4;               -- estado 4.
                        when 4 =>
```

```
            spi_clk<='0';
            spi_mosi<=gain(5);
            edo<=5;
        when 5 =>
            spi_clk<='1';
            edo<=6;
        when 6 =>
            spi_clk<='0';
            spi_mosi<=gain(4);
            edo<=7;
        when 7=>
            spi_clk<='1';
            edo<=8;
        when 8 =>
            spi_clk<='0';
            spi_mosi<=gain(3);
            edo<=9;
        when 9 =>
            spi_clk<='1';
            edo<=10;
        when 10 =>
            spi_clk<='0';
            spi_mosi<=gain(2);
            edo<=11;
        when 11 =>
            spi_clk<='1';
            edo<=12;
        when 12 =>
            spi_clk<='0';
```

```
            spi_mosi<=gain(1);
            edo<=13;
        when 13 =>
            spi_clk<='1';
            edo<=14;
        when 14 =>
            spi_clk<='0';
            spi_mosi<=gain(0);
            edo<=15;
        when 15 =>
            spi_clk<='1';
            edo<=16;
        when 16 =>
            spi_clk<='0';
            edo<=17;
        ------------------Fin programa del preamplificador.
        ------------------inicio de la lectura del adc
        when 17 =>
            ad_conv<='1'; -- reset del convertidor.
            spi_clk<='0';
            edo<=18;
        when 18 =>
            edo<=19;
        when 19 =>
            ad_conv<='0';          --inicia proceso del convertidor
adc.
            edo<=20;
        when 20 =>
            spi_clk<='1';  --primer pulso
```

```
            edo<=21;
        when 21 =>
            spi_clk<='0';
            edo<=22;
        when 22 =>
            spi_clk<='1';  --segundo pulso
            edo<=23;
        when 23 =>
            spi_clk<='0';
            edo<=24;
        ---------- inicia lectura canal 0
        when 24 =>                  -- dato 13 del canal 0 del adc.
            spi_clk<='1';
            edo<=25;
        when 25 =>
            spi_clk<='0';
            c0(13)<=spi_miso;
            edo<=26;
        when 26 =>                  -- dato 12 del canal 0 del adc
            spi_clk<='1';
            edo<=27;
        when 27 =>
            spi_clk<='0';
            c0(12)<=spi_miso;
            edo<=28;
        when 28 =>                  -- dato 11 del canal 0 del adc.
            spi_clk<='1';
            edo<=29;
        when 29 =>
```

```
            spi_clk<='0';
            c0(11)<=spi_miso;
            edo<=30;
        when 30 =>                  -- dato 10 del canal 0 del adc
            spi_clk<='1';
            edo<=31;
        when 31 =>
            spi_clk<='0';
            c0(10)<=spi_miso;
            edo<=32;
        when 32 =>                  -- dato 9 del canal 0 del adc.
            spi_clk<='1';
            edo<=33;
        when 33 =>
            spi_clk<='0';
            c0(9)<=spi_miso;
            edo<=34;
        when 34 =>                  -- dato 8 del canal 0 del adc
            spi_clk<='1';
            edo<=35;
        when 35 =>
            spi_clk<='0';
            c0(8)<=spi_miso;
            edo<=36;
        when 36 =>                  -- dato 7 del canal 0 del adc
            spi_clk<='1';
            edo<=37;
        when 37 =>
            spi_clk<='0';
```

```
            c0(7)<=spi_miso;
            edo<=38;
        when 38 =>                  -- dato 6 del canal 0 del adc
            spi_clk<='1';
            edo<=39;
        when 39 =>
            spi_clk<='0';
            c0(6)<=spi_miso;
            edo<=40;
        when 40 =>                  -- dato 5 del canal 0 del adc
            spi_clk<='1';
            edo<=41;
        when 41 =>
            spi_clk<='0';
            c0(5)<=spi_miso;
            edo<=42;
        when 42 =>                  -- dato 4 del canal 0 del adc
            spi_clk<='1';
            edo<=43;
        when 43 =>
            spi_clk<='0';
            c0(4)<=spi_miso;
            edo<=44;
        when 44 =>                  -- dato 3 del canal 0 del adc
            spi_clk<='1';
            edo<=45;
        when 45 =>
            spi_clk<='0';
            c0(3)<=spi_miso;
```

```
            edo<=46;
        when 46 =>                  -- dato 2 del canal 0 del adc
            spi_clk<='1';
            edo<=47;
        when 47 =>
            spi_clk<='0';
            c0(2)<=spi_miso;
            edo<=48;
        when 48 =>                  -- dato 1 del canal 0 del adc
            spi_clk<='1';
            edo<=49;
        when 49 =>
            spi_clk<='0';
            c0(1)<=spi_miso;
            edo<=50;
        when 50 =>                  -- dato 0 del canal 0 del adc
            spi_clk<='1';
            edo<=51;
        when 51 =>
            spi_clk<='0';
            c0(0)<=spi_miso;
            edo<=52;
        ------------------fin canal 0
        when 52 =>
            spi_clk<='1';  --primer pulso
            edo<=53;
        when 53 =>
            spi_clk<='0';
            edo<=54;
```

```
when 54 =>
        spi_clk<='1';  --segundo pulso
        edo<=55;
when 55 =>
        spi_clk<='0';
        edo<=56;
------------------inicia lectura canal 1
when 56 =>                  -- dato 13 del canal1 del adc
        spi_clk<='1';
        edo<=57;
when 57 =>
        spi_clk<='0';
        c1(13)<=spi_miso;
        edo<=58;
when 58 =>                  -- dato 12 del canal1 del adc
        spi_clk<='1';
        edo<=59;
when 59 =>
        spi_clk<='0';
        c1(12)<=spi_miso;
        edo<=60;
when 60 =>                  -- dato 11 del canal1 del adc
        spi_clk<='1';
        edo<=61;
when 61 =>
        spi_clk<='0';
        c1(11)<=spi_miso;
        edo<=62;
when 62 =>                  -- dato 10 del canal1 del adc
```

```
            spi_clk<='1';
            edo<=63;
        when 63 =>
            spi_clk<='0';
            c1(10)<=spi_miso;
            edo<=64;
        when 64 =>                  -- dato 9 del canal1 del adc
            spi_clk<='1';
            edo<=65;
        when 65 =>
            spi_clk<='0';
            c1(9)<=spi_miso;
            edo<=66;
        when 66 =>                  -- dato 8 del canal1 del adc
            spi_clk<='1';
            edo<=67;
        when 67 =>
            spi_clk<='0';
            c1(8)<=spi_miso;
            edo<=68;
        when 68 =>                  -- dato 7 del canal1 del adc
            spi_clk<='1';
            edo<=69;
        when 69 =>
            spi_clk<='0';
            c1(7)<=spi_miso;
            edo<=70;
        when 70 =>                  -- dato 6 del canal1 del adc
            spi_clk<='1';
```

```
            edo<=71;
        when 71 =>
            spi_clk<='0';
            c1(6)<=spi_miso;
            edo<=72;
        when 72 =>                  -- dato 5 del canal1 del adc
            spi_clk<='1';
            edo<=73;
        when 73 =>
            spi_clk<='0';
            c1(5)<=spi_miso;
            edo<=74;
        when 74 =>                  -- dato 4 del canal1 del adc
            spi_clk<='1';
            edo<=75;
        when 75 =>
            spi_clk<='0';
            c1(4)<=spi_miso;
            edo<=76;
        when 76 =>                  -- dato 3 del canal1 del adc
            spi_clk<='1';
            edo<=77;
        when 77 =>
            spi_clk<='0';
            c1(3)<=spi_miso;
            edo<=78;
        when 78 =>                  -- dato 2 del canal1 del adc
            spi_clk<='1';
            edo<=79;
```

```
when 79 =>
        spi_clk<='0';
        c1(2)<=spi_miso;
        edo<=80;
when 80 =>                  -- dato 1 del canal1 del adc
        spi_clk<='1';
        edo<=81;
when 81 =>
        spi_clk<='0';
        c1(1)<=spi_miso;
        edo<=82;
when 82 =>                  -- dato 0 del canal1 del adc
        spi_clk<='1';
        edo<=83;
when 83 =>
        spi_clk<='0';
        c1(0)<=spi_miso;
        edo<=84;
----------------fin canal1
when 84 =>
        spi_clk<='1';  --primer pulso
        edo<=85;
when 85 =>
        spi_clk<='0';
        edo<=86;
when 86 =>
        spi_clk<='1';  --segundo pulso
        edo<=87;
when 87 =>
```

```
                    spi_clk<='0';
                    edo<=88;
                ------------------fin de la lectura del adc
                when 88 =>
                    edo<=89;
                    canal1<=c1;   -- se le asigna a canal1 la variable c1.
                    canal0<=c0;  -- se  le asigna a canal 0 la variable c0.
                    ad_conv<='0';--apagado del converidor.
                --when 89 =>
                    --edo<=17;
                when others =>
            end case;--fin màquina de estados.
            end if;
        end if;
end if;
end process preampli;

process (sel1,sel2,canal1,canal0,clk)
begin
if clk='1' and clk'event then   -- condicion para cuando haya un pulso de reloj.
if sel1 = '1' then                  -- seleccionamos canal 1(sel1=canal).
        if sel2='0' then          --seleccionamos nivel alto(sel2=nivel).
            data<=canal1(13 downto 6); -- muestra nivel alto del canal1.
        else
            data<=canal1(7 downto 0);   --muestra nivel bajo del canal1.
        end if;
else
        if sel2 = '0' then
            data<=canal0(13 downto 6);   -- muestra nivel alto del canal0.
```

```
        else
                data<=canal0(7 downto 0);   --muestra nivel bajo del canal.
        end if;
end if;
end if;
end process;  -- fin del process.


end Behavioral;--fin de la arquitectura.
```

RESULTADOS DEL AMPLIFICADOR CON GANANCIA -1 Y EL CONVERTIDOR ANALÓGICO-DIGITAL DEL FPGA SPARTAN 3E.

VOLTAJE DE ENTRADA	VALOR DECIMAL		VALOR DE BITS
	TEORICO	PRACTICO	
0.4	8192	7805	01111001111101
0.6	6880	6109	01011111011101
0.8	5570	5093	01001111100101
1	4259	3512	01110110111000
1.2	2948	2735	00101010101111
1.4	1638	1502	00010111011110
1.5	983	672	00001010100000
1.65	0	0	00000000000000
1.7	-327	-112	10000001110000
1.9	-1638	-1638	10010110010100
2.1	-2948	-2948	10100001101001

9.1 CONCLUSION

Para la adquisición de la señal ECG se usaran tres parches de electrodo. Se colocara el electrodo de referencia sobre la parte interna del tobillo derecho. El electrodo negativo a un costado de la costilla. El tercer electrodo sobre el corazón. Para conectar a los electrodos con la entrada del amplificador de instrumentación con amplificación 1000V/V se usara un cable modelo KENDALL-LTP marca TYCO HelthCare. Las señales ECG serán filtradas analógicamente y después las señales ECG serán amplificadas con ganancia

1.65V/V para posteriormente ser convertidas de analógico a digital, para su procesamiento en el FPGA. Una vez aplicados los filtros digitales sobre la señal digitalizada del ECG, los resultados obtenidos a la salida del mismo serán transmitidos a través del puerto serie a una PC para su visualización.

Diseño y construcción de la etapa de amplificación. Para amplificar la señal proveniente de los electrodos se usara el amplificador de instrumentación AD624 con una amplificación total de 1000 V/V. Para acoplar el voltaje que entregara el sensor (Offset = 1V) al voltaje de 1.65v, requerido como nivel cero por el convertidor ADC del Kit Spartan 3E, se agregara un amplificador con ganancia 1.65 V/V.

Diseño y construcción de los filtros analógicos. Se implemento un filtro analógico pasa banda de 0.05 Hz a 100 Hz, de segundo orden, y un filtro rechaza banda de 60 Hz.

Activación del Modulo de Conversión ADC del KIT de Desarrollo Spartan 3E. En esta parte se convierte la señal analógica del cuerpo (voltaje) a una palabra digital de 14 bits con signo en notación de complemento a dos. Esto se hace mediante el convertidor analógico - digital incluido en el LTC1407A-1 del kit de desarrollo Spartan 3E. Este ADC convierte voltajes de ±1.25v relativos a 1.65v.

Desarrollo de la Programación de los Filtros Digitales en VHDL. En esta etapa se pretende mejorar la señal aplicando distintos tipos de filtros; pasa altas (0.05 Hz), rechaza banda (60Hz) y pasa bajas (150Hz). El objetivo es eliminar frecuencias menores a 0.1Hz, mayores a 150Hz y el ruido de la fuente de alimentación. Así se evitaran señales extrañas provocadas hasta por los mismos circuitos.

Desarrollo de la Programación para comunicación del FPGA con la PC. La comunicación del kit con la computadora será a través del puerto serie RS232 usando para esto un cable conversor de puerto serie RS232 a USB. La programación del FPGA para la comunicación será en VHDL.

Desarrollo de la Programación para visualización de los datos en la PC. La interfase grafica del usuario será en lenguaje C++ Builder. Las señales ECG se guardaran en el disco duro de la PC y se llamaran para su visualización.

CRONOGRAMA DE ACTIVIDADES.

ACTIVIDAD (2009)	ENERO	FEBRERO	MARZO	ABRIL	MAYO	JUNIO
Análisis bibliográfico	X	X	X	X	X	X
Cáp. I Introducción.	X	X				
Cáp. II Fundamento Teórico.			X	X		
Diseño y construcción de la etapa de amplificación.		X	X			
Diseño y construcción de los filtros analógicos.			X	X		
Desarrollo de la Programación de los Filtros Digitales en VHDL.				X	X	
Activación del Modulo de Conversión ADC del KIT de Desarrollo Spartan 3E		X	X	X		
Desarrollo de la Programación para comunicación del FPGA con la PC.			X	X	X	X
Desarrollo de la Programación para visualización de los datos en la PC.				X	X	X
Reporte final.						X

BIBLIOGRAFIA

[1] Victor Borshevich, Alexandr V. Mustyatsa, Wiacheslav L. Oleinik, "*Fuzzy Spectral Analysis of Heart´s Rhythms",* Fifth IFSA World Congress 1993; págs:561-563; 1993

[2] Ken Chapman, "PicoBlase Amplifier and A/D Converter Control for Spartan 3-E Starter Kit", Xilinx Ltd, 23 february 2006

[3] "Spartan-3E Starter Kit Board User Guide", UG230(v1.0) june 20, 2008 en http://www.xilinx.com

[4] "IEEE Standard VHDL Language Reference Manual", IEEE Std 1076-1987

[5] Miguel Alberto Melgarejo Rey, "Desarrollo de un filtro digital promedio en FPGA", en

http://www.udistrital.edu.co/comunidad/grupos/glogica/index.html