PRACTICA EMPRESARIAL EN LA UEN BIOINGENIERIA DE LA FUNDACIÓN CARDIOVASCULAR DE COLOMBIA

HOLGUER ANDRES BECERRA DAZA

UNIVERSIDAD PONTIFICIA BOLIVARIANA

ESCUELA DE INGENIERIAS Y ADMINISTRACIÓN

FACULTAD DE INGENIERIA ELECTRONICA

BUCARAMANGA

2011

PRACTICA EMPRESARIAL EN LA UEN BIOINGENIERIA DE LA FUNDACIÓN CARDIOVASCULAR DE COLOMBIA

HOLGUER ANDRES BECERRA DAZA

Práctica empresarial para obtener el título de

INGENIERO ELECTRONICO

Asesor Supervisor:

ING. FABIO ALONSO GUZMAN SERNA

UNIVERSIDAD PONTIFICIA BOLIVARIANA

ESCUELA DE INGENIERIAS Y ADMINISTRACIÓN

FACULTAD DE INGENIERIA ELECTRONICA

BUCARAMANGA

2011

NOTA DE ACEPTACIÓN

_____

PRESIDENTE DEL JURADO

_____

JURADO

_____

JURADO

_____

BUCARAMANGA, SEPTIEMBRE DE 2011

DEDICATORIA

A Dios por todas las oportunidades, bendiciones y personas que puso a mi lado.

A mi familia por ser un apoyo incondicional en todo momento.

A DMC por brindarme su apoyo y amor incondicional en los momentos mas dificiles.

A Dr. Yair Linn por enseñarme las bases de lo que hizo posible esta practica empresarial.

EL AUTOR

AGRADECIMIENTOS

El autor expresa sus agradecimientos a:

*La UNIVERSIDAD PONTIFICIA BOLIVARIANA por formar profesionales íntegros en la calidad humana e intelectual.

*Al Ingeniero Fabio Guzmán Serna, por guiarme y ayudarme en el transcurso de la práctica empresarial.

*A la Fundación Cardiovascular de Colombia especialmente a la UEN Bioingeniería por darme la oportunidad de hacer la práctica empresarial en sus instalaciones.

LISTA DE FIGURAS

TABLA DE CONTENIDO

RESUMEN

TITULO: PRACTICA EMPRESARIAL EN LA UEN BIOINGENIERIA DE LA FUNDACIÓN CARDIOVASCULAR DE COLOMBIA

AUTOR: HOLGUER ANDRES BECERRA DAZA

FACULTAD: FACULTAD DE INGENIERIA ELECTRONICA

DIRECTOR: FABIO ALONSO GUZMAN SERNA

La práctica de ingeniería electrónica se hizo en la UEN (unidad estratégica de negocios) Bioingeniería de la Fundación Cardiovascular de Colombia, dedicada a diseñar, desarrollar y producir dispositivos médicos que brindan servicios de calidad al sector de la salud a nivel nacional. El principal objetivo de esta práctica era realizar la descripción hardware, la implementación y programación del sistema embebido NIOS II utilizando una FPGA, con el propósito de hacer la adquisición de datos de tarjetas electrónicas que conforman un monitor de signos vitales para presentar los datos en una interfaz gráfica de usuario utilizando una pantalla LCD TouchScreen. Para el diseño de esto se utiliza una tarjeta de desarrollo llamada Nios II Embedded Evaluation Kit (NEEK) de la compañía Altera producida por Terasic corp.

Se hace el estudio de las tarjetas de adquisición de parámetros fisiológicos para hacer la decodificación y codificación de datos, seguido de esto se procede a realizar la descripción de hardware de un sistema que soporte los periféricos necesarios para hacer la adquisición y visualización de signos vitales sobre una pantalla LCD y finalmente se diseña un software compatible con el hardware diseñado para generar una interfaz gráfica de usuario donde se muestren los datos arrojados por cada una de las tarjeta electrónicas que conforman un monitor de signos vitales. Las actividades se realizaron con éxito y bajo la supervisión de

jefe de Diseño y Desarrollo. Cada actividad realizada tuvo en cuenta las normas internacionales AAMI que rigen los estándares de calidad cuando se diseñan equipos médicos. Todo el trabajo fue evaluado como positivo para la empresa y contribuye un gran aporte al conocimiento de la misma para el diseño de equipos médicos basados en sistemas embebidos reconfigurables.

PALABRAS CLAVES:

FPGA, SISTEMA EMBEBIDO NIOS II, INTERFAZ GRAFICA DE USUARIO, DESCRIPCION DE HARDWARE, SIGNOS VITALES, TARJETA DE ADQUISICION DE DATOS, EQUIPO MEDICO, MONITOR DE SIGNOS VITALES.

Vº Bº DIRECTOR DE TRABAJO DE GRADO

ABSTRACT

TITLE: BUSINESS PRACTICE IN UEN BIOENGINEERING AT CARDIOVASCULAR FOUNDATION OF COLOMBIA.

AUTHOR: HOLGUER ANDRES BECERRA DAZA

FACULTY: FACULTY OF ELECTRONIC ENGINEERING

DIRECTOR: FABIO ALONSO GUZMAN SERNA

The practice of electronic engineering was done in UEN (strategic business unit) Bioengineering at Cardiovascular Foundation of Colombia, dedicated to desing, develop and produce medical devices that provide quality services to the healthcare industry nationwide. The main objective of this practice was to make the hardware description, implementation and programming of the NIOS II embedded system using a FPGA, in order to realize the data acquisition electronic boards that form a vital signs monitor to present the data in a graphical user interface using a LCD touchscreen. For the design were used a Nios II development called Embedded Evaluation Kit (NEEK) of the company produced by Terasic Altera corp. In first place, were studied of acquisition cards physiological parameters for decoding and encoding of data, then, were described the hadware of a system that supports the necessary peripherals to make the acquisition and display of vital signs on a LCD screen. Finally, were created the design of software and hardware with the purpose to generate a graphical user interface that shows the data obtained from each of the electronic cards that form a vital signs monitor.

All activities were completed successfully under supervision of Chief Desing and Development. Each activity was made using AAMI rules; those are quality standards in the design medical equipment. This work was evaluated as positive

for the company and it’s a great contribution to the knowledge of it for the design of medical equipment based on reconfigurable embedded systems.

KEY WORDS:

FPGA, EMBEDDED SYSTEM NIOS II, GRAPHIC USER INTERFACE, HARDWARE DESCRIPTION, VITAL SINGS, DATA ACQUISITION ELECTRONIC BOARD, MEDICAL DEVICES, VITAL SINGS MONITOR.

Vº Bº DEGREE DIRECTOR OF WORK

INTRODUCCIÓN

En la Fundación Cardiovascular de Colombia (FCV) y específicamente en la UEN (**U**nidad **E**stratégica de **N**egocios) Bioingeniería se diseñan y producen dispositivos médicos necesarios para la supervisión y medición de parámetros fisiológicos. Uno de estos productos es el monitor de signos vitales que está situado hoy en día en muchos hospitales y clínicas a nivel nacional, entre estas clínicas se encuentra el Instituto del corazón(IC); en el cual constantemente se atienden pacientes con problemas cardiovasculares que están conectados a monitores que miden Electrocardiografía(ECG), Oximetría(SpO2), Temperatura(TEMP), Respiración(RESP), entre otros parámetros que deben ser censados con la mayor precisión y veracidad posible ya que muchos pacientes dependen de un diagnostico hecho por un médico que se basa en las mediciones que los monitores arrojan en sus pantallas.

Hoy en día, los monitores que funcionan en el IC están hechos en sistemas basados en PCs convencionales y no sobre sistemas dedicados, por esta y otras razones los monitores hechos en la UEN Bioingeniería se bloquean y sobrecalientan, corriendo el riesgo de que la RAM interna y otros componentes se quemen. De igual forma en ocasiones los datos visualizados en pantalla presentan ruidos indeseables, consumen mucha potencia y se reinician, generando eventos adversos que pueden llegar a ser de alto riesgo para un paciente en cirugía o un paciente con alguna patología cardiovascular delicada, esto imposibilita que el monitor de signos vitales pueda ser implementado en salas de cirugías. La UEN Bioingeniería ha decidido cambiar la arquitectura y el diseño actual de sus monitores implementando tecnologías de punta como lo son las FPGAs (Field Program Gates Arrays) para mejorar el desempeño, consumo de potencia y confiabilidad de este tipo de dispositivos.

Las FPGAs son chips que poseen la capacidad de cambiar su lógica interna gracias a que en su interior están compuestas un por campo de arreglo de compuertas lógico programables ó CLBs, en esta tecnología se puede diseñar lógicas digitales complejas que permiten hacer desde sencillos chips como los de la serie 74xx, hasta procesadores tan complejos como los ARM utilizados actualmente hoy en casi cualquier dispositivo móvil. Las FPGAs también se conocen porque en su lógica interna pueden crearse múltiples sistemas que funcionen de manera independiente dándole la ventaja de ser un dispositivo con capacidades de soportar sistemas hardware de manera paralela. Se programan por medio de lenguajes HDLs como Verilog, VHDL, System Verilog y otros, con los cuales se puede hacer descripción de hardware de cualquier sistema digital por

medio de palabras, generando que de esta manera sean fáciles de reprogramar en su lógica interna.

Gracias a las ventajas que posee el uso de este tipo de tecnologías, la UEN Bioingeniería ha decidido incursionar e innovar en su uso contratando a un estudiante de la Universidad Pontificia Bolivariana como practicante, que conozca de esta tecnología y que pueda aportar su conocimiento en el diseño de un nuevo monitor de signos vitales basado en FPGAs.

El estudiante contratado, logro diseñar un sistema base capaz de cumplir con todas la necesidades que requiere un monitor de signos vitales para hacer monitoreo de los parámetros fisiológicos ECG, TEMP, RESP y NIBP por medio de una interfaz gráfica de usuario. Mostrándole a la UEN Bioingeniería que trabajar con este tipo de tecnologías (FPGA), puede traer un mayor grado innovación y seguridad a sus productos.

DESCRIPCION DE LA EMPRESA

1.1 RESEÑA HISTORICA DE LA FCV

La historia de la Fundación Cardiovascular de Colombia se remonta al año 1986 cuando un grupo de especialistas y personalidades de Bucaramanga se propuso crear una entidad privada sin ánimo de lucro dedicada a tratar las enfermedades del corazón, logrando en octubre de 1990 que un grupo de médicos iniciara las actividades de consulta y prueba de esfuerzo en la Fundación Tercera Edad de la Congregación Mariana, y las primeras cirugías cardiovasculares en la Clínica Bucaramanga.

En el año 1992 entró a formar parte de la Clínica Carlos Ardila Lulle, adquiriendo el cuarto piso, ampliando así todos los servicios diagnósticos e intervencionistas de cardiología y cirugía vascular periférica, utilizando salas de cirugía, unidad de cuidados intensivos y hospitalización de esta moderna clínica.

Posteriormente en octubre de 1997 se inauguró la nueva sede del Instituto del Corazón, un moderno edificio de 14 pisos con una capacidad de 123 camas de hospitalización distribuidas entre la unidad de Cuidados Intensivos Postquirúrgica, unidad de Cuidado Intensivos unidad de Cuidados Intensivos Pediátrica, unidad de Cuidados Intermedios Adultos, tres pisos de hospitalización, 4 salas de cirugía, 2 salas de Hemodinámica y 1 de más del servicio de urgencias durante las 24 horas del día cumpliendo así con todos los requisitos y normas exigidas por el Ministerio de salud relacionadas con enfermedades cardiovasculares.

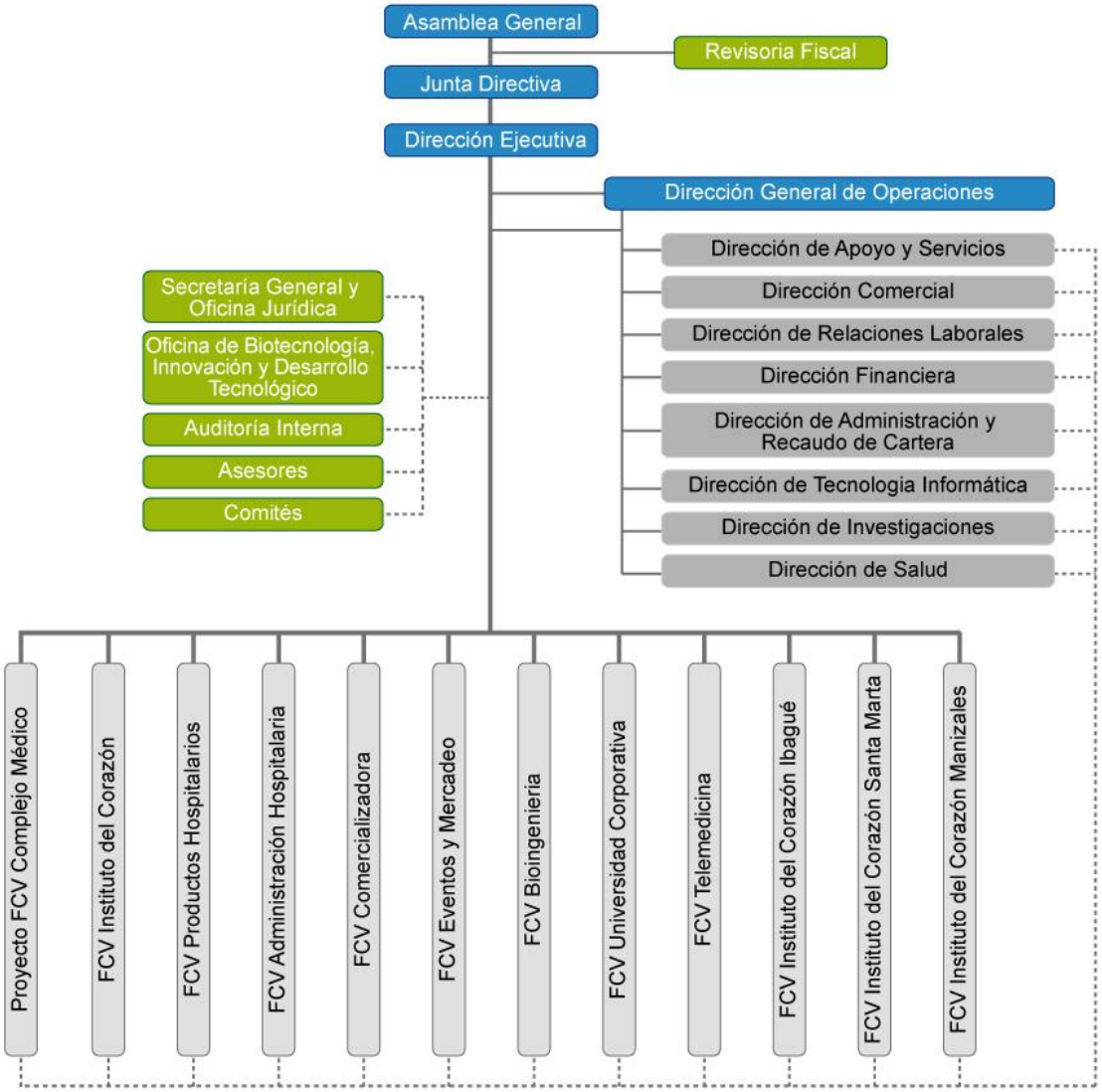
Misión empresarial:
La FCV (Fundación Cardiovascular de Colombia) es una organización empresarial Sin ánimo de lucro que provee servicios y productos de salud de alta calidad para el desarrollo del sector buscando permanentemente el bienestar de la comunidad.

Visión empresarial:
En el año 2020 la Fundación Cardiovascular de Colombia será una organización Reconocida a nivel nacional e internacional por la excelencia e innovación de sus Productos y servicios orientados principalmente al sector salud

Datos de la Empresa:
El instituto del corazón está actualmente ubicado en la Calle 155a # 23-58 Urbanización el bosque Floridablanca, Santander, Colombia con el número

Telefónico 6399292.

FIGURA 1. ORGANIGRAMA DE LA FCV



1.2 DESCRIPCIÓN DEL AREA DE PRÁCTICA

Misión:
La UEN FCV Bioingeniería, es una unidad empresarial de negocios de la Fundación Cardiovascular de Colombia que contribuye con el desarrollo científico y tecnológico en la salud mediante la producción de conocimiento, innovación, transferencia y apropiación de tecnologías dirigidas al mejoramiento de las condiciones de vida de la población colombiana con posicionamiento del desarrollo Tecnológico y la producción nacional en el contexto internacional.

Visión:
En el 2020 la unidad empresarial FCV Bioingeniería será reconocida en el país como una entidad desarrolladora y productora de equipos médicos confiables y competitivos, con un alto componente de innovación tecnológica.

Datos de la UEN FCV Bioingeniería:
La UEN FCV Bioingeniería está ubicada actualmente en Cra 5 # 6 -33 Floridablanca, Santander, Colombia con el número telefónico 6497304 ext. 4151.

1.3 DIAGNOSTICO DE LA EMPRESA

La FCV brinda productos de salud de alta calidad para el desarrollo del sector buscando permanentemente el bienestar de la comunidad y por esto en su UEN de Bioingeniería la FCV ha diagnosticado la necesidad de implementar equipos electro médicos sobre sistemas embebidos basados en FPGAs, ya que son dispositivos que por su flexibilidad pueden configurar rápidamente su hardware interno haciéndolo un dispositivo capaz de cumplir casi cualquier tarea sin necesidad de cambiar las tarjetas físicas o chips en el hardware externo, también son dispositivos de baja potencia que pueden trabajar rápidamente y con alto rendimiento gracias a que en ella se puede diseñar hardware que trabaje en paralelo en comparación a otros dispositivos.

En Bioingeniería actualmente se está diseñando y produciendo monitores de signos vitales con equipos basados en PC convencionales y ha resultado problemático ya que no son dispositivos dedicados, consumen mucha potencia, se bloquean fácilmente, no son portales y requieren de licencia en el software que se maneja para diseñar interfaces de usuario(Desarrolladas en Labview), por estas razones la UEN Bioingeniería ha decidido utilizar sistemas embebidos utilizando FPGAs gracias a sus múltiples ventajas.

La FCV en su UEN Bioingeniería contrata al practicante por falta de personal para investigar en el tema, disminuir tiempo de desarrollo, hacer transferencia de conocimiento y brindar la posibilidad al estudiante en proceso de grado de que aplique sus conocimientos.

OBJETIVOS

2.1 OBJETIVO GENERAL

Realizar la descripción de hardware, la implementacion y programación del procesador NIOS II en una FPGA que permita manejar las tarjetas electrónicas que conforman un monitor de signos vitales para presentar los datos en una interfaz gráfica de usuario amigable utilizando una pantalla touchscreen.

2.2 OBJETIVOS ESPECIFICOS

- ✓ Definir y establecer la referencia y proveedor de tarjetas electrónicas OEM que se deben comprar para conformar un monitor de signos vitales ECG, RESP, TEMP, NIBP y SpO2.

- ✓ Estudiar e implementar el protocolo de transmisión y recepción de cada una de las tarjetas OEM ECG, RESP, TEMP, NIBP y SpO2.

- ✓ Estudiar e implementar las características del procesador NIOS II de Altera corp. para hacer la una descripción de hardware compatible con cada una de las tarjetas OEM de signos vitales, pantalla touch y video.

- ✓ Programar el procesador NIOS II para generar una interfaz gráfica de usuario, donde se puedan observar los parámetros fisiológicos medidos por cada una de las tarjetas OEM de signos vitales.

2.3 ACTIVIDADES A DESARROLLAR PARA CUMPLIR CON LOS OBJETIVOS

- ✓ Definir y establecer la referencia y proveedor de las tarjetas OEM de signos vitales.
- ✓ Estudiar e implementar cada una de las tarjetas OEM de signos vitales
- ✓ Estudiar e implementar cada una de las tarjetas OEM de signos vitales
- ✓ Programar el procesador NIOS II para generar una interfaz de gráfica de usuario

2.3.1 CRONOGRAMA DE ACTIVIDADES

Actividad	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10	Semana 11	Semana 12	Semana 13	Semana 14	Semana 15	Semana 16
Definir y establecer la referencia y proveedor De las tarjetas OEM de signos vitales																
Estudiar e implementar cada una de las tarjetas OEM de signos vitales																
Estudiar e implementar las características del procesador NIOS II																
Programar el procesador NIOS II para generar una interfaz de gráfica de usuario																

3. MARCO TEORICO

3.1 TARJETAS OEM DE SIGNOS VITALES

Las tarjetas OEM de adquisición de parámetros fisiológicos, son dispositivos médicos capaces de medir o censar los cambios físicos y eléctricos que genera el cuerpo humano para convertirlos en parámetros tales como lo son la electrocardiografía(ECG), oximetría(SpO2), Temperatura(TEMP), presión no invasiva(NIBP) y Respiración(RESP). Estas tarjetas hacen adquisición, filtrado y procesamiento de las señales eléctricas del cuerpo medidas a través de electrodos, sensores infrarrojos, sensores de temperatura y sensores de presión para enviarlas mediante un protocolo de comunicación(SPI, RS232, I2C u otro) a un host que decodificara esta información y la convertirá en datos leíbles por el usuario.

La ventaja que tiene utilizar tarjetas OEM en sistemas de visualización de parámetros fisiológicos, es que ahorran el proceso de procesamiento, adaptación y filtrado de señales que requieren un alto grado de procesamiento matemático y cuidado en el diseño eléctrico y electrónico para evitar ruidos indeseables que pueden causar mediciones erróneas causales de que los médicos hagan falsos diagnósticos que pongan en riesgo la vida del paciente.

3.2 NORMAS QUE RIGEN EL DISEÑO DE LOS DISPOSITIVOS MEDICOS

De acuerdo a las normas que rigen el diseño y producción de dispositivos médicos la UEN Bioingeniería está certificada en las normas ISO 9001 y ISO 13485, por lo tanto todos los procesos y diseños de dispositivos médicos electrónicos deben basarse en todas las normas de calidad vigentes por la norma AAMI(American National Standard), centrándose en la sección ANSI/AAMI/IEC 60601-1-2:2007 que habla de la seguridad que debe brindar cualquier dispositivo medico electrónico en cuanto a la compatibilidad eléctrica y electromagnética, esta norma es importante debido a que las tarjetas OEMs seleccionadas(compradas) para el monitor de signos vitales deben estar construidas y diseñadas teniendo estas normas, de no serlo está prohibido hacer uso de estas en algún diseño que este en proceso la UEN Bioingeniería ya que pueden causar que los productos que se

saquen al mercado causen factores indeseados al usuario provocando un servicio poco fiable, la perdida de los certificados de calidad y multas económicas grandes.

La norma AAMI no solo está centrada en la parte eléctrica y electrónica, también reúne todo lo que el diseñador debe tener en cuenta a la hora de hacer diseño de dispositivos médicos como: diseño de carcasa, duración de baterías, software, movilidad del dispositivo e incluso en el diseño de interfaces graficas de usuario(GUI) como se enfoca en la sección ANSI/AAMI HE75:2009, que debe ser estudiada por un diseñador gráfico o industrial para construir una GUI que se acomode a las necesidades médicas y del usuario final.

3.3 LAS FPGAS HOY EN DIA

Hoy en día existen diferentes marcas de FPGAs en el mercado, las más conocidas son las de Xilinx, Altera y las Actel. Cada marca de FPGAs ofrece servicios de aporte al conocimiento con el que brindan tutoriales, cursos y libros para aprender y aplicar las FPGAs en campos complejos que van desde el procesamiento digital de imágenes hasta las comunicaciones digitales. Dependiendo de la compañía se pueden obtener servicios enfocados hacia diferentes áreas, en el caso de Actel es una compañía que ofrece su marca de FPGAs enfocando sus aplicaciones al campo industrial, donde aplican la tecnología en sistemas de potencia y control avanzado, por otro lado la marca Xilinx ofrece sus FPGAs enfocadas hacia el campo de comunicaciones digitales y procesamiento de video en campos industriales, mientras que la marca Altera se enfoca hacia el campo del entretenimiento en el desarrollo de aplicaciones de procesamiento de video, audio y comunicaciones digitales. El que cada compañía tenga un enfoque tecnológico diferente, no quiere decir que las FPGAs de Altera no puedan utilizarse en campos en los que aplica Actel, ni quiere decir que Actel no pueda aplicar en campos como los de Xilinx.

Una de las grandes ventajas de las FPGAs es su flexibilidad a la hora de cambiar su hardware interno por medio de descripción de hardware utilizando lenguajes de programación como VHDL o Verilog, actualmente las FPGAs han adquirido fuerza al verse aplicadas en campos industriales y científicos como lo hace el CERN(European Organization for Nuclear Research) para medir la trayectoria el partículas en movimiento (CERN Trajectory Measurement System, 2007) donde aplican esta tecnología debido a que no existe dispositivo ASIC capaz de controlar tantas ramas de control al mismo tiempo con alta velocidad y con un bajo consumo de potencia. De igual manera en el campo medico se ha empezado a innovar en el

uso de esta tecnología como se evidencia articulo del Xcell Journal de Xilinx Corp del año 2010 (Xilinx, 2010) con el sistema Aloka que utiliza ultrasonido para visualizar imágenes dentro del cuerpo humano para hacer diagnósticos de forma no invasiva, en este artículo también se pueden encontrar avances de la NASA, Samsung y el CERN gracias al uso de FPGAs.

Las FPGAs son usadas para diseñar y verificar procesadores por su facilidad en la modificación y corrección de estos (Ho, 2009) (Ningfeng Huang, 2009) (Qiu Chuanfei, 2010) (WANG Ziting, 2009), dando a estas ventajas sobre los procesadores convencionales basados en ASICs debido a que estos no pueden cambiar su hardware interno y no pueden cambiar su funcionalidad física.

Esta tecnología cada vez está siendo más usada en investigaciones y diseño de productos, por su versatilidad de manejo, velocidad, confiablidad y cada vez más económico precio.

Altera corp, ofrece sus FPGAs haciendo tarjetas de desarrollo (ver FIGURA 2) de tipo académico e investigativo fabricadas y vendidas por Terasic corp, para que estudiantes y compañías agilicen el proceso de diseño y desarrollo de productos o proyectos, implementando esta tecnología.

Viendo las ventajas que ofrece esta tecnología la FCV ha decido aplicar las FPGAs de Altera en el diseño y producción de equipos médicos.

FIGURA 2. TARJETA DE DESARROLLO NEEK (NIOS II EMBEDDED EVALUATION KIT)

3.4 TARJETA DE DESARROLLO NEEK

La NEEK(Nios II Embedded Evaluation Kit) (ver FIGURA 2) hecha por Terasic corp, es una tarjeta de desarrollo de bajo costo, con la que los desarrolladores de hardware y software pueden hacer diseños utilizando FPGAs de Altera y la implementacion del procesador NIOS II Esta tarjeta está compuesta por 2 bordas, una de ellas contiene la una FPGA Cyclone III y la otra contiene periféricos de video, audio, comunicación y una CPLD MAX II que funciona como interfaz entre los periféricos y la FPGA. Estas dos tarjetas se unen por un puerto llamado HSMC(High Speed Mezzanine Card) como se muestra en la FIGURA 2.

La NEEK cuenta con una pantalla LCD a color con touchscreen, ranura para tarjeta SD, puerto Ethernet, PS2, Rs232, VGA-DB9, Audio-in, Audio-out, Mic-in y entrada de TV-compuesto, estos periféricos pueden ser utilizados para procesar video, audio o para hacer de adquisición de datos que puedan ser mostrados en una interfaz gráfica de usuario(GUI) implementando la pantalla LCD.

En la UEN Bioingeniería se utiliza esta tarjeta de desarrollo para la adquisición de datos y el diseño de la interfaz gráfica de usuario donde se visualizaran los parámetros fisiológicos censados y enviados por las tarjetas OEM.

3.4.1 FPGA TARJETA DE DESARROLLO NEEK

La tarjeta NEEK utiliza una FPGA Cyclone III de Altera con la referencia EP3C25F324, esta (Altera, Cyclone III Device Handbook, Volume 1, 2009) tiene una capacidad digital interna de 25kLEs(Logic Elements), que en comparación al número de compuertas que tiene internamente la Spartan 3A 700A de Xilinx corresponde al doble, además contiene internamente 4 PLLs(**P**hase-**l**ocked **l**oop**s**) internos, ram, multiplexores y multiplicadores hardware que facilitan el diseño y la implementacion de circuitos digitales complejos.

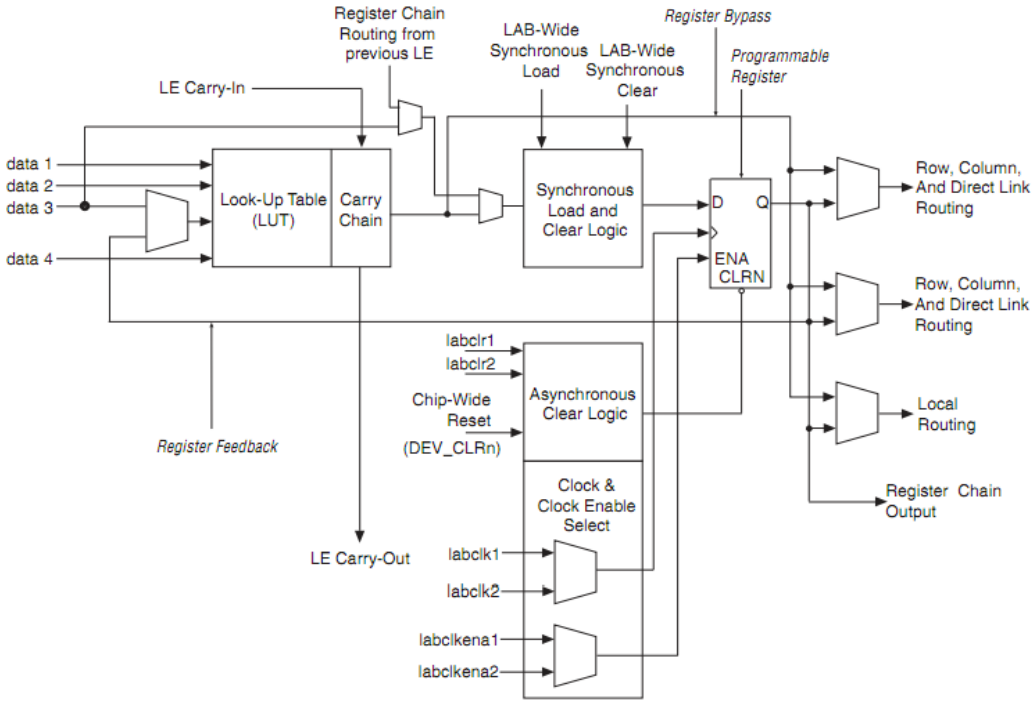
3.4.1.1 ¿QUE SON LEs?

En las FPGAs de Altera los LEs (**L**ogic **E**lement**s**) (Altera, Logic Elements and Logic Array Blocks CIII51002-2.2 Volume 1, 2009) corresponden el número de elementos lógicos internos que tiene una FPGA o una CPLD dentro de su estructura interna, la principal función de los LEs es ayudar a usar y distribuir eficientemente la lógica que conforma un circuito digital, cada LE (ver FIGURA 3) posee características como:

- Entrada de 4 look-uptable(LUT) lo que permite poder implementar cualquier función de 4 variables.
- UN registro programable.
- Una cadena carry de conexión.
- Una cadena de Registro de conexión.
- Registro de empaquetamiento.
- Registro para realimentacion.
- La capacidad para hacer las siguientes inter-conexiones:

    - Local.
    - Fila.
    - Columna.
    - Cadena de registros.
    - Enlace directo.

FIGURA 3. Arquitectura Interna de un LE(Logic Element)- imagen tomada de Cyclone III device hanbook, Volume 1-December-2009.



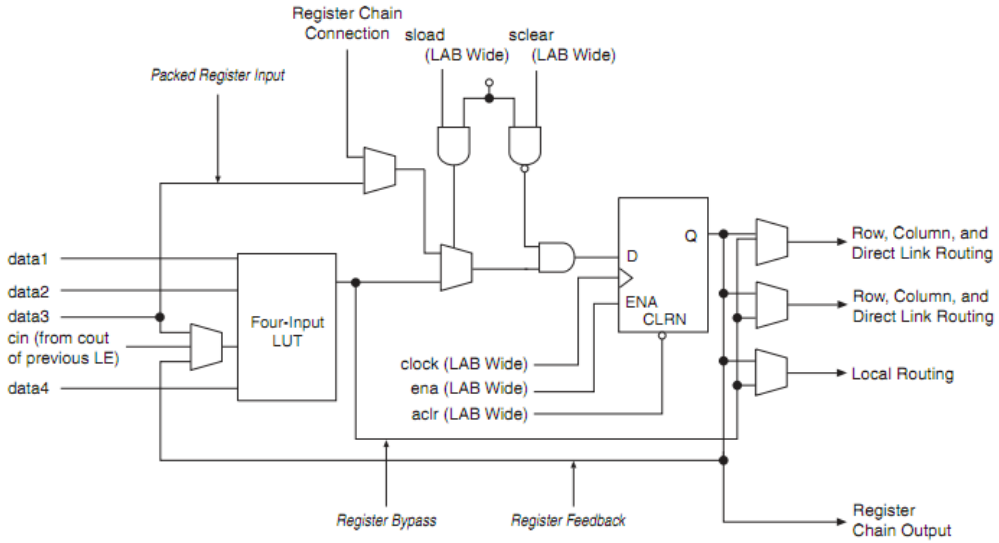
Estos LE pueden configurarse como flipflops tipo D,T, JK o SR, cada registro tiene un Data, Clock, Clock-Enable y Clear de entradas que se pueden usar en lógicas

secuenciales para generar diferentes clases de circuitos digitales, la configuración de cada LE depende de la configuración del LUT(Look-Up-Table) interno.

Cada LE tiene 3 salidas que conducen a las filas y columnas de este para generar interconexiones locales entre otros LEs. El LE tiene 2 modos de uso, el normal y aritmético.

Él LE en modo normal (ver **FIGURA 4):** Se utiliza cuando se implementa lógica combinacional corriente y lógica secuencial sencilla en la descripción de hardware, como multiplexores, flipflops tipo D y circuitos lógicos con solo compuertas, el Quartus II localiza partes de código del HDL y configura LEs en modo normal con cuanta lógica combinacional encuentre.

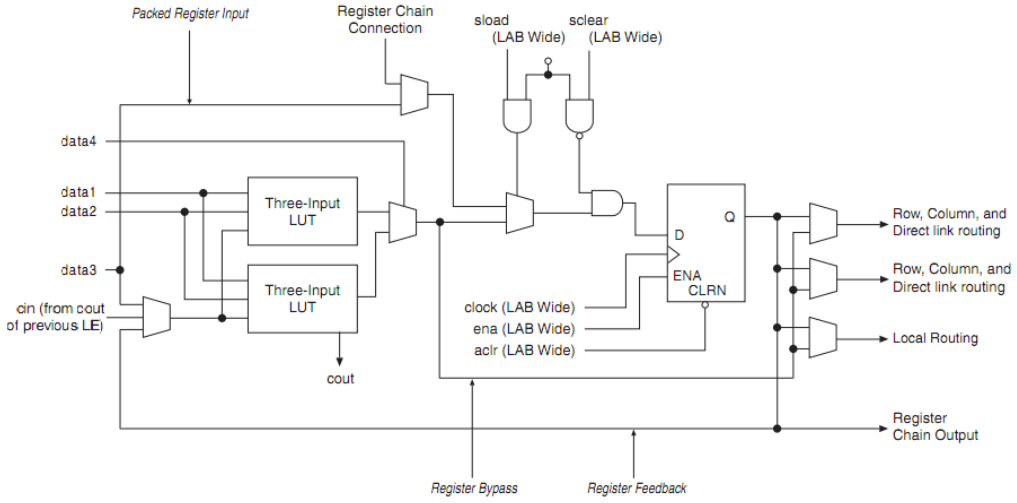
FIGURA 4. LE en modo normal- imagen tomada de Cyclone III device hanbook, Volume 1-December-2009.



Él LE en modo aritmético (ver

FIGURA 5): se utiliza cuando se implementa lógica que requiera de contadores, sumadores y comparadores. Cuando el Quartus II localiza código hardware que requiera de este tipo de lógica, Activa los LEs en modo aritmético.

FIGURA 5. LE en modo Aritmético- imagen tomada de Cyclone III device hanbook, Volume 1-December-2009.



Cuando se hace descripción de hardware por medio de lenguajes como VHDL o Verilog HDL, el software de compilación Quartus II que es el compilador de las FPGAs de marca Altera, Sintetiza, Optimiza y Genera el código para programar cada LUT de los LEs configurándolos de forma recursiva para generar interconexiones entre estos haciendo posibles las lógicas internas descritas en los lenguajes de HDL.

El conjunto de LEs genera bloques llamados LABs(**L**ogic **A**rray **B**lock**s**) , cada

LAB (ver

FIGURA 6) tiene:

- 16 LEs.
- Señales de control
- Un LE de carry.
- Cadenas de registros
- Interconexiones locales.

FIGURA 6. LAB, arquitectura interna- imagen tomada de Cyclone III device hanbook, Volume 1-December-2009.

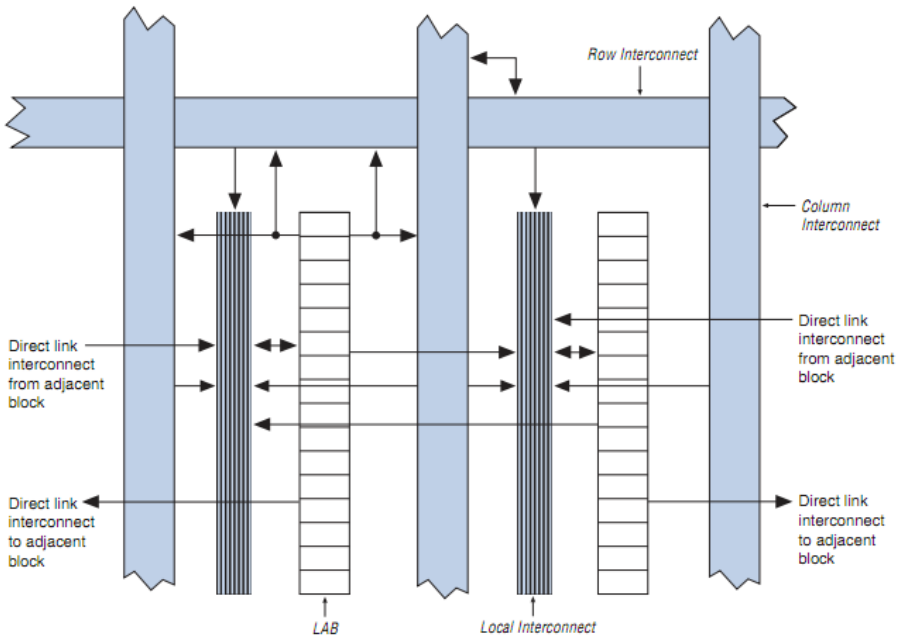
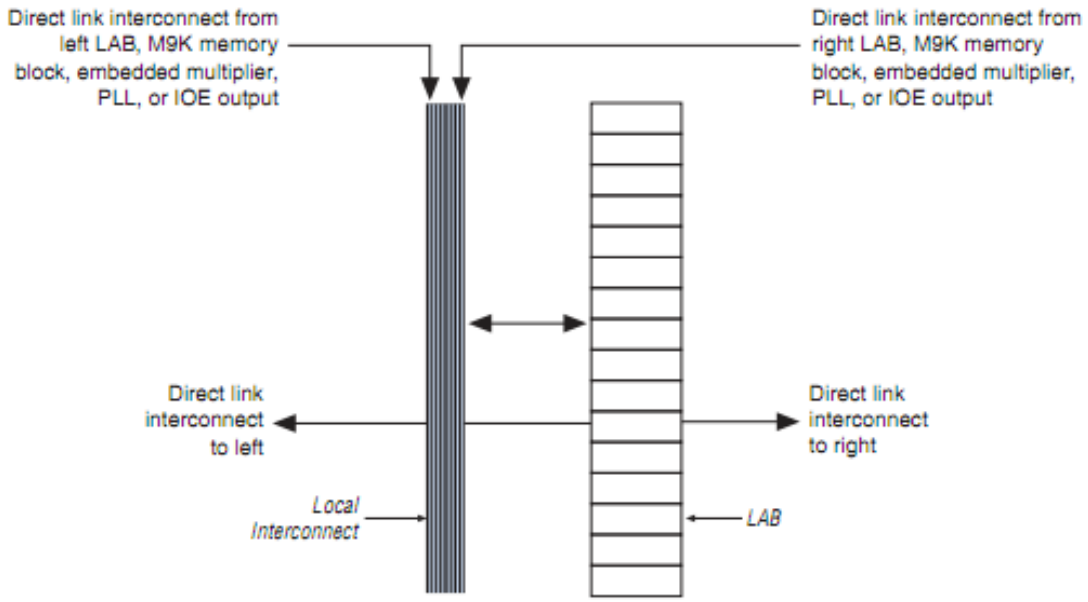


FIGURA 7. Interconexión del LAB entre módulos Hardware locales como PLLs, RAMs, MUXs, etc.- imagen tomada de Cyclone III device hanbook, Volume 1- December-2009.



Cada interconexión local conecta señales dentro del mismo LAB utilizando sus LEs internos y a su vez genera conexiones a otros LAB adyacentes para genera la lógica digital que este descrita. Los LABs también sirven para optimizar recursos y facilitar el ordenamiento de la descripción de Hardware al software Quartus II, este proceso sucede de forma automática cada vez que se sintetiza una descripción de hardware.

Los LAB tienen conexiones locales por medio de sus columnas, filas y las salidas de los LEs en el mismo LAB, estas conexiones(ver **FIGURA 7**) van unidas directamente a bloques hardware embebidos en la arquitectura de la FPGA, como lo son PLLs(Phase-locked loops), M9K RAM, multiplicadores internos y multiplexores, que ayudan a optimizar el diseño y economizar recursos a la hora de describir sistemas hardware complejos como procesadores u otro hardware que requiera o utilice una cantidad significativa de LEs, estos módulos hardware se utilizan al hacer implementacion de las librerías IP core de Altera en la descripción de hardware.

3.5 SOFTWARE QUARTUS II

Con este software se diseñan, sintetizan, implementan y programan descripciones de hardware hechas en esquemáticos digitales o mediante lenguajes HDL (VHDL, Verilog) las FPGAs y CPLDs de la marca Altera.

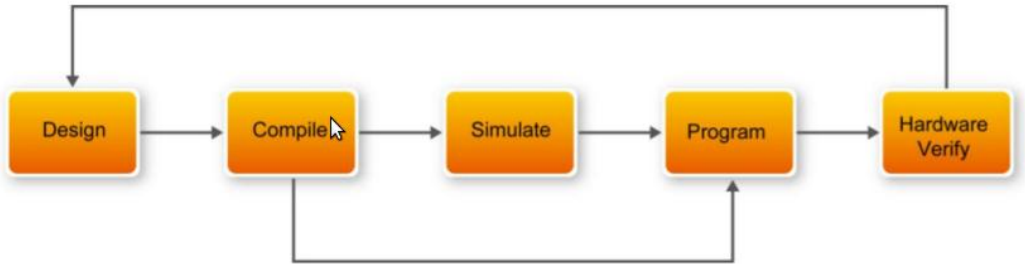
El lenguaje de descripción de hardware usado para generar los diseños en el sistema para el monitor de signos vitales implementado en la NEEK es verilog HDL, debido a su facilidad de uso para generar módulos hardware de manera rápida, eficiente y confiable en comparación a VHDL. Verilog es un lenguaje de descripción de hardware de alto nivel con el cual se hace diseño y verificación de hardware. Muchas compañías reconocidas en el diseño de hardware alrededor del mundo prefieren usar este lenguaje por las ventajas que presenta, como se explica en el artículo publicado por Stuart Sutherland en el año 2009 titulado “Is System Verilog Useful for FPGA Desing & Verification” (Sutherland, 2009).

3.5.1 PASOS PARA DISEÑAR HARDWARE SOBRE QUARTUS II

Cuando se diseña hardware por medio de HDL o esquemáticos utilizando cualquier software de compilación (Quartus II, ISE, Actel), el diseñador debe seguir una serie de pasos (ver **FIGURA 8**) básicos que lo ayuden a crear hardware verificable, confiable y seguro.

- Diseñar.
- Compilar (Sintetizar Implementar).
- Simular.
- Programar.
- Verificar Hardware.

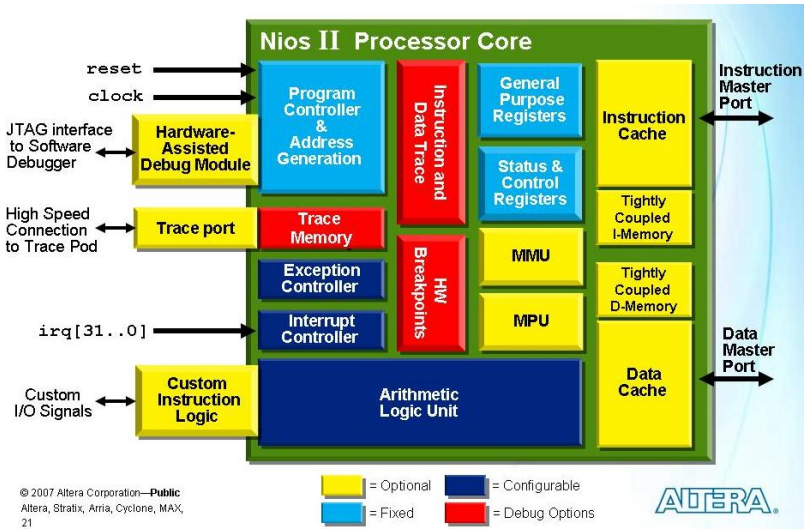
FIGURA 8. Pasos para diseñar hardware - imagen tomada del manual de usuario de la DE0-NANO de Terasic Corp.



3.6 PROCESADOR EMBEBIDO NIOS II

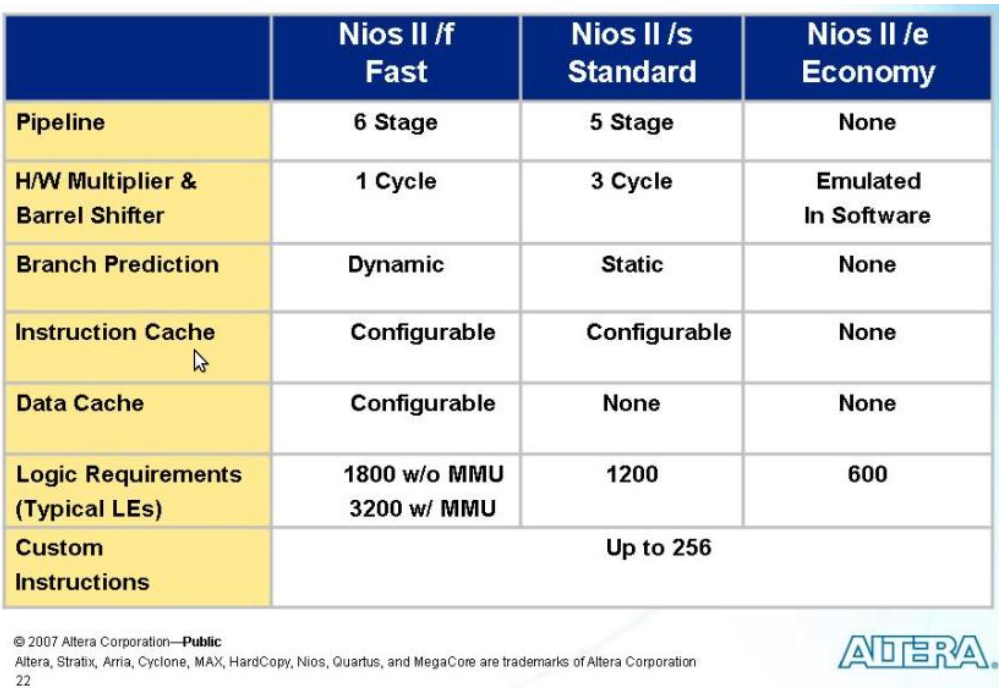
El procesador embebido de 32 bits Nios II(ver **FIGURA 9**) basado en arquitectura RISC, es un IP core de Altera corp utilizado únicamente en la FPGAs de la misma marca, el Nios II es comparable al microblaze de Xilinx y es utilizando en múltiples aplicaciones en el campo industrial y académico (Xilinx, 2010) (Ho, 2009) (Ningfeng Huang, 2009), este procesador puede adaptarse a sistemas hardware reconfigurables que requieran un bajo o alto grado de rendimiento.

FIGURA 9: Configuración interna del procesador Nios II de Altera corp.



Una de las ventajas de este procesador es que al ser reconfigurable el diseñador puede agregar o quitar periféricos sin necesidad de hacer cambios en el hardware a comparación de un sistema ASIC, optimizando de esta manera el uso de recursos y dinero ya que el sistema se adapta a las necesidades específicas del usuario y el usuario no tiene porque conformase con el sistema o desperdiciar recursos de manera innecesaria. Como lo muestra la **FIGURA 10** existen tres variaciones del procesador Nios II que pueden ser utilizadas según sean las condiciones del sistema que se desee diseñar y la FPGA que se utilice. Este procesador puede trabajar con frecuencias de reloj de hasta 300 MHz a 345MIPS dependiendo del tipo de FPGA según la arquitectura interna que está presente, debido a esto el procesador puede escalar su rendimiento para diseñar sistemas que requieran cumplir tareas de altos niveles de procesamiento(GHz) como lo muestra la **FIGURA 11**.

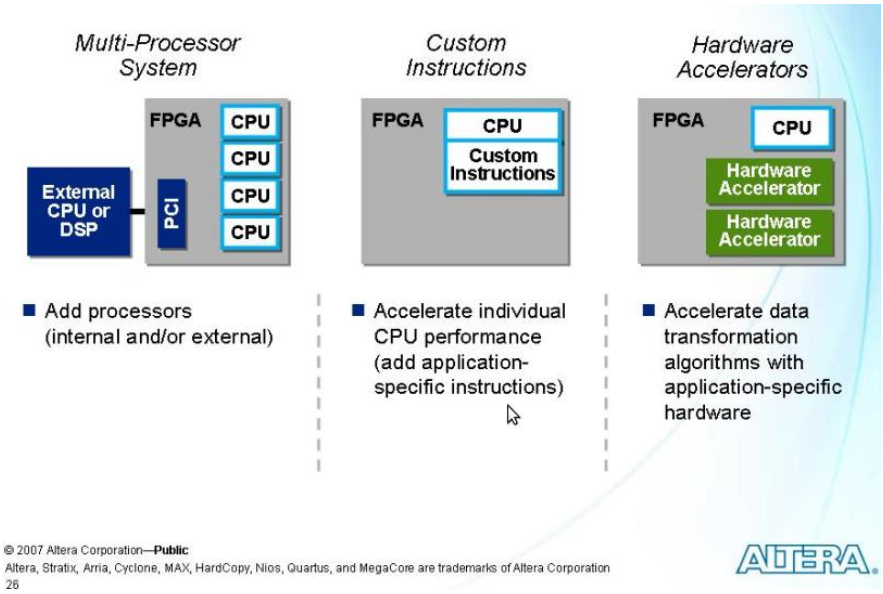
FIGURA 10. Variaciones del procesador Nios II.



	Nios II /f Fast	Nios II /s Standard	Nios II /e Economy
Pipeline	6 Stage	5 Stage	None
H/W Multiplier & Barrel Shifter	1 Cycle	3 Cycle	Emulated In Software
Branch Prediction	Dynamic	Static	None
Instruction Cache	Configurable	Configurable	None
Data Cache	Configurable	None	None
Logic Requirements (Typical LEs)	1800 w/o MMU 3200 w/ MMU	1200	600
Custom Instructions	Up to 256		

22

FIGURA 11. Las 3 maneras de escalar el rendimiento del procesador Nios II en sistemas que requieran un alto grado de procesamiento.



3.6.1 ESTRUCTURA BASICA DE UN SISTEMA BASADO EN NIOS II

Cuando se diseñan sistemas basados en el procesador Nios II se debe tener en cuenta la estructura básica para que este funcione correctamente, como se ve en la **FIGURA 9**, el procesador tiene buses de entrada y de salida que deben conectarse para que el sistema funcione de la manera adecuada, si no se conectaran correctamente estos buses el procesador puede funcionar de la manera inadecuada.

Buses del procesador Nios II:

- **JTAG:** Sirve para programar y supervisar las funciones del procesador Nios II, si no se conecta este bus el procesador no podrá funcionar.
- **Instruction Master Port:** Esta encargado del cache de instrucciones, si este bus no se conecta junto con el Data master a RAM, no se podrán leer ni ejecutar las funciones programadas por el usuario.
- **Data Master Port:** Esta encargado de supervisar, enviar y recibir los datos de los puertos, RAM y demás periféricos que estén conectados al Nios II.
- **Reset:** Esta encargado de reiniciar el Nios II, para iniciar de nuevo los procesos que se lean desde el master de instrucciones y reiniciar todo el sistema de periféricos.

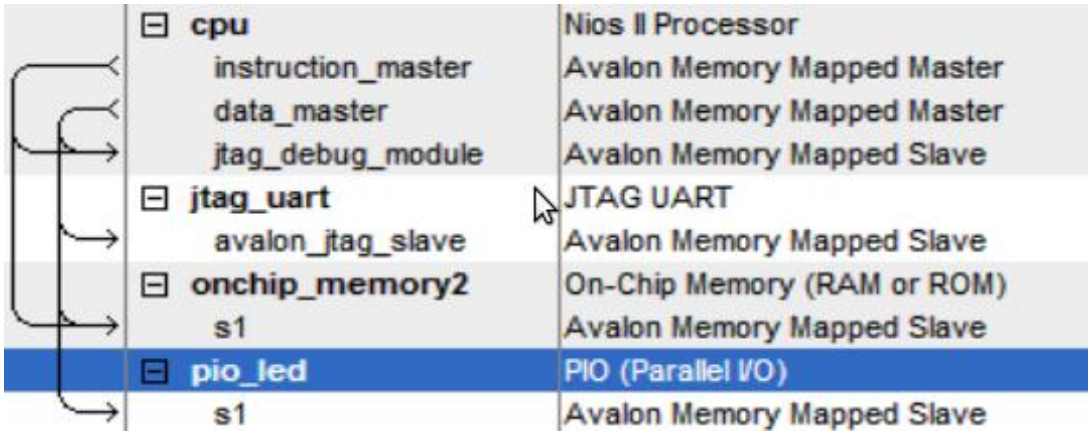
- **Clock:** Esta encargado de recibir la frecuencia de reloj del oscilador externo para que puedan completarse los ciclos de maquina necesarios para operar el procesador.
- **Irq[31:0]:** Esta encargado de recibir las interrupciones hardware de los periféricos conectados de acuerdo a la prioridad que tengan.
- **High Speed Connection to Trace Pod:** Esta encargado junto con el JTAG de supervisar el estado de compilación de procesador.

En la **FIGURA 12** se puede observar en un diagrama de conexión de un sistema básico con el procesador Nios II, como se puede observar los periféricos necesarios para un funcionamiento adecuado deben:

- **(*)CPU:** Procesador nios II, puede ser Nios II/e, Nios II/s o Nios II/f.
- **(*)JTAG:** Puerto de programación encargado de gestionar y supervisar el estado del procesador Nios II, con este periférico se programan las instrucciones en la memoria RAM(Puede ser otra memoria Flash) del procesador y se puede hacer debug del sistema.
- **(*)RAM:** Encargado de almacenar el set de instrucciones del procesador y las variables se utilicen.
- **PIO:** Puerto de salida o entrada de datos.

(*)Periférico obligatorio.

FIGURA 12. Conexión básica de un sistema basado en el procesador Nios II.



3.6.2 HERRAMIENTA SOPC BUILDER

SOPC Builder**(S**ystem **o**n **P**rogrammable **C**hip **B**uilder) es una herramienta para la construcción de sistemas basados en el procesador Nios II(ver **FIGURA 9**), donde se puede diseñar sistemas configurables y adaptables a las necesidades del usuario por medio de HDLs para generar soluciones rápidas y de bajo costo, que a comparación de un sistema basado en ASICs (ver **FIGURA 13**) se tendría que

rediseñar todo el hardware, cambiar la PCB y los elementos electrónicos lo que aumenta el costo y el tiempo del el diseño.

FIGURA 13. Sistema ASIC Multichip tradicional.

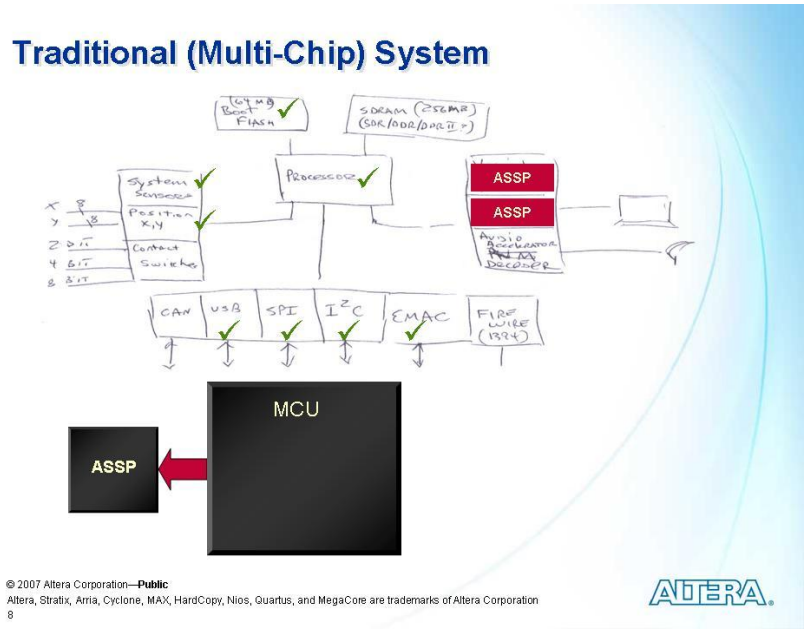
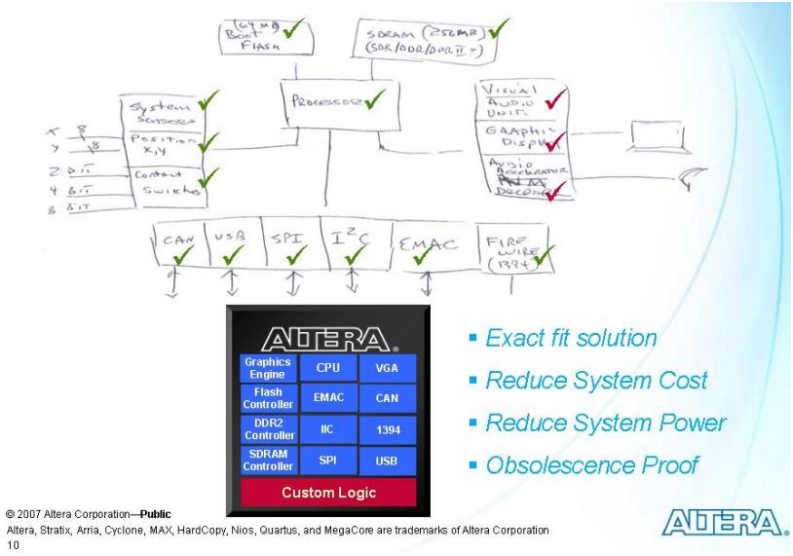


FIGURA 14. Sistema programable sobre un chip basado en Nios II, SOPC.



El SOPC es esencial para el diseño de sistemas basados en Nios II ya que es una herramienta de prototipo rápido con la cual se pueden construir sistemas en minutos a partir de librerías de IP cores de Altera u otros HDLs para hacer unión modular de los mismos al procesador Nios II.

Para programar la interfaz gráfica de usuario necesaria para la visualización de parámetros fisiológicos sobre la tarjeta NEEK (ver FIGURA 2), se debe diseñar un sistema con unas características específicas, capaz de soportar los periféricos de la misma tarjeta de desarrollo como lo son la pantalla LCD para visualización de datos, el Touchscreen para interacción de usuario y RS232 adquisición de datos.

3.6.3 HERRAMIENTA NIOS II IDE

La mayoría de sistemas embebidos tienen interfaces de desarrollo IDE para programar sus procesadores, El procesador embebido Nios II cuenta con una interfaz gráfica de usuario basada en Eclipse llamada Nios II IDE con la cual se compila código en ANSI-C estándar y lenguaje C++. Esta interfaz brinda al programador diferentes herramientas que le ayudan a compilar código de manera rápida, amigable y segura.

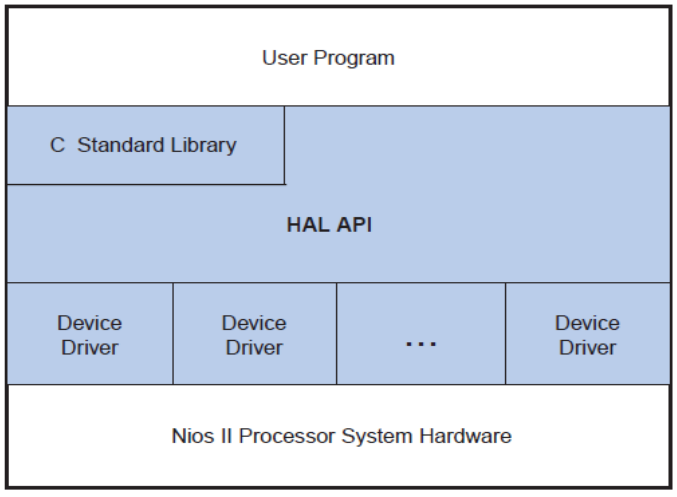
Esta herramienta de diseño permite estructurar código portable, migrable y estructurado, compatible con otros sistemas hardware, permitiendo al desarrollador utilizar librerías iguales en diferentes proyectos sin necesidad de reescribir código lo que agiliza el proceso de diseño y desarrollo de productos.

La interfaz de desarrollo Nios II IDE permite al programador de software crear proyectos en los cuales se configura el tipo de sistema a programar en la FPGA, inmediatamente después de describir un sistema hardware utilizando la herramienta SOPC builder de Altera, los archivos generados por la misma herramienta HDL(*.ptf y *.sopc) contienen la información necesaria del sistema para que de esta manera el Nios II IDE genere una descripción software sobre el hardware donde se necesita correr la aplicación a desarrollar. En esta librería se encuentran los drivers de cada periférico y se encuentra el código de inicialización de este, lo que genera una capa “framework“entre la aplicación a programar y el sistema hardware como se muestra en la **FIGURA 15**. Una vez se escribe y se compila la aplicación utilizando lenguaje C/C++, el IDE procede hacer el debug de la aplicación utilizando el puerto JTAG conectado con el cual puede observar, pausar, saltar, reiniciar y parar los procesos del procesador para observar detalladamente su funcionamiento, el compilador no solo es capaz hacer debug de la aplicación, este permite correr la aplicación en tiempo real indefinidamente mostrado en consola los valores que envié el puerto JTAG por medio de la función “printf“.

El Nios II IDE sigue los siguientes pasos para ejecutar aplicaciones software utilizando debug sobre el sistema hardware descrito:

- El IDE revisa el sistema hardware por medio de los archivos .pft y .sopc, verificando que no haya habido modificaciones a este por medio del SOPC Builder, si es así el IDE re-compila y caracteriza de nuevo el sistema para generar nuevamente la capa framework.
- El IDE compila la aplicación escrita en lenguaje C/C++ y genera un archivo *.elf.
- Verificar por medio del puerto JTAG el ID del sistema, esta identificación reside en el hardware “System ID“ que está unido al procesador. Si el sistema no es compatible el Nios II pide al usuario programar por medio de la herramienta Quartus II el sistema hardware.
- Una vez el IDE comprueba la existencia del sistema compatible por medio del ID, este procede a programar el procesador con el archivo *.elf generado en el campo de memoria donde se debe alojar el programa, después el IDE resetea el procesador para que este se inicialice.
- Una vez está corriendo la aplicación sobre el sistema, el usuario podrá ver en la consola de IDE los comandos enviados por el puerto JTAG y podrá parar, pausar y saltar los procesos de procesador.

FIGURA 15. Capas del sistema programado Nios II- Imagen tomada del Nios II Software Developer’s Handbook NII5V2-10.0.



4. DESARROLLO DE LA PRÁCTICA

4.1 CUMPLIMIENTO DE OBJETIVOS

Para el inicio de actividades era necesario partir de conocer la empresa, sus objetivos, el perfil, la misión, visión y el diagnostico actual de esta para entrar a la línea de enfoque empresarial de la FCV y su UEN Bioingeniería.

Una vez se conoce el funcionamiento y el direccionamiento estratégico de la compañía, junto a sus integrantes y colaboradores, se empieza el proceso de diseño y desarrollo para construir una interfaz gráfica de usuario para un monitor de signos vitales utilizando tarjetas OEM de parámetros fisiológicos y la tarjeta de desarrollo NEEK por medio de sistemas embebidos basados en FPGAs.

Para el cumplimiento de los objetivos se empieza con el estudio de las FPGAs de marca Altera y su compilador Quartus II, la descripción de hardware hecha por medio de la herramienta de compilación se hace en Verilog HDL debido a que es un lenguaje por excelencia utilizado para hacer diseño y verificación de sistemas hardware complejos.

Como actividad en paralelo se trabaja en el estudio para definir y establecer la referencia y proveedor de tarjetas electrónicas OEM que se compraron para conformar un monitor de signos vitales con ECG, RESP, TEMP, NIBP y SpO2. Una vez definidas se procede a estudiar e implementar el protocolo de comunicación de cada una de ellas permitiendo utilizar este en el sistema embebido diseñado e implementado en la FPGA de Altera.

Completada la codificación y decodificación de tarjetas OEM, se procede a estudiar el procesador NIOS II (IP Core) de Altera utilizando la herramienta SOPC Builder y Nios II IDE. Este estudio está separado en tres partes, el estudio de la tarjeta de desarrollo NEEK, el estudio hardware de los sistemas embebidos basados en procesadores Nios II y el estudio de la herramienta de compilación de software compatible con el procesador IP Core de Altera.

Finalizado el proceso investigativo de las herramientas Altera y las tarjetas OEM, se empieza el diseño en conjunto para llegar al desarrollo de una interfaz gráfica de usuario donde se puedan visualizar los diferentes parámetros fisiológicos dados por las tarjetas OEM definidas. El proceso de diseño y desarrollo de la GUI estuvo acompañada de procesos de verificación que ayudaron al mejoramiento continuo de los diseños generados con la tarjeta de desarrollo NEEK.

Todo el proceso de aprendizaje e investigación sobre el tema de sistemas embebidos en FPGAs, se hizo utilizando los manuales de referenciación de Altera, los ejemplos Web de Terasic corp y los libros de diseño avanzado de hardware referenciados al final del documento. Ninguna persona en la compañía era conocedor del tema, lo cual trajo al a FCV en su UEN Bioingeniería un aporte al

conocimiento absolutamente nuevo y pionero en Colombia en el campo de diseño de productos basados en plataformas FPGAs.

4.1.1 TAREAS DESARROLLADAS

4.1.1.1 DEFINIR TARJETAS OEM SIGNOS VITALES

Se contactaron las empresas extranjeras para la búsqueda de tarjetas OEM de medición de parámetros fisiológicos, Mindray, Nellcore, MedLab, ChoiceMed y SuntechMed. La búsqueda de estas empresas se hizo teniendo en cuenta el trayecto, calidad y reconocimiento por parte de otras compañías que dan excelentes referencias respecto a la calidad y responsabilidad de estas. Además que se tuvieron en cuenta los certificados de calidad de la compañía en base a las normas AAMI, para seleccionar un producto que cumplan con los estándares de calidad eléctricos y electromagnéticos. La UEN Bioingeniería tiene una base de datos con folletos de empresas que ya han contactado y han demostrado ser marcas reconocidas, esta base de datos también se utilizó para tener una referencia.

Una vez contactadas las empresas se pidió a cada una de ellas una cotización con el precio de cada tarjeta (SpO2,TEMP, RESP, ECG y NIBP), también se les pidió los certificados de calidad para verificar que la información dada como referencias de otras empresas y este atada al término legal de cada país, además de que la UEN Bioingeniería necesita seguir procesos de calidad que demuestren y garanticen productos seguros y confiables basados en las normas AAMI, por esta razón se necesita que cada componente utilizado en los equipos médicos debe regirse de igual manera bajo procesos de calidad de altos estándares y deben estar certificadas por entidades legales mundialmente reconocidas, un caso de esto es que las empresas americanas contactadas para la compra de componentes médicos estén certificadas bajo la FDA entidad encargada de la administración de productos alimenticios y médicos en USA.

Después de que las empresas responden con las debidas cotizaciones, se procede a definir cuales tarjetas OEM serán compradas o no a cada compañía contactada, el proceso de definir las tarjetas depende de los precios, referencias y certificados de calidad que ofrezca cada compañía, esta decisión la toma el Jefe de D&D(Diseño y Desarrollo) encargado de analizar cuál de las compañías es la mejor para el producto que se esté diseñando. Las compañías definidas por el Jefe de D&D para la compra de las tarjetas OEM son Mindray, Nellcore y SuntechMed.

Finalmente, definidas las empresas se llenan los formatos de requisición de importación que deberán ser aprobados por el Gerente Actual de Bioingeniería para empezar la compra por parte del área de mercadeo de la FCV en Bogotá. Como resultado de todo el proceso se hizo la compra a las compañías de las tarjetas de medición de parámetros fisiológicos de las empresas Mindray y SuntechMed (La de la empresa Nellcore no llego debido a procesos de mercadeo).

4.1.1.2 IMPLEMENTACION DE CADA UNA DE LAS TARJETAS OEM DE SIGNOS VITALES

Como resultado del proceso descrito en el cap 4.1.1.1 DEFINIR TARJETAS OEM SIGNOS VITALES, se obtienen las tarjetas Mindray y SuntechMed para la medición de parámetros fisiológicos, cada una trae consigo un datasheet donde se hace explicación del protocolo de comunicación e interfaz que utiliza para enviar y recibir datos desde un host. Estos manuales pertenecen únicamente a la FCV y son de tipo confidencial, debido a que cuando se hace la compra de cada una de las tarjetas la FCV está comprando a la compañía un producto que se ata a los términos legales de protección de la propiedad intelectual, por lo cual los manuales que adquiere la FCV en la compra son de tipo confidencial, si se revelaran al público alguno de estos manuales, traería como consecuencia problemas de tipo legal que pueden afectar a la FCV especialmente a la UEN Bioingeniería, provocando que se pierdan proveedores importantes lo que afecta posibles alianzas y diseño que se hagan con otras compañías.

Se inicia el estudio de cada tarjeta OEM, para conocer la interfaz (Puerto de comunicación) y su protocolo de comunicación. El estudio de estos es de vital importancia para garantizar una buena recepción y transmisión de datos, por parte de un host (FPGA) y la tarjeta OEM, cada protocolo se analizó de manera cuidadosa, teniendo en cuenta los métodos de detección de errores y la interfaz o tipo de comunicación que utiliza cada módulo de adquisición.

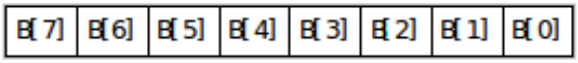
TARJETA MINDRAY: La tarjeta mindray utiliza una interfaz RS-232 comunicándose a un host (FPGA) por medio de comunicación serial asíncrona, con la cual hace transmisión y recepción de datos a una velocidad de 115200 bps, cada byte enviado cuenta con un bit de paridad de tipo impar para garantizar la integridad de cada byte enviado por la tarjeta, los paquetes de bytes enviados por esta cuentan con el método de detección de errores checksum para garantizar la integridad de cada paquete de información enviado y recibido de la tarjeta.

Esta tarjeta es capaz de censar los parámetros fisiológicos de ECG, RESP y TEMP.

PROTOCOLO DE COMUNICACIÓN MINDRAY:

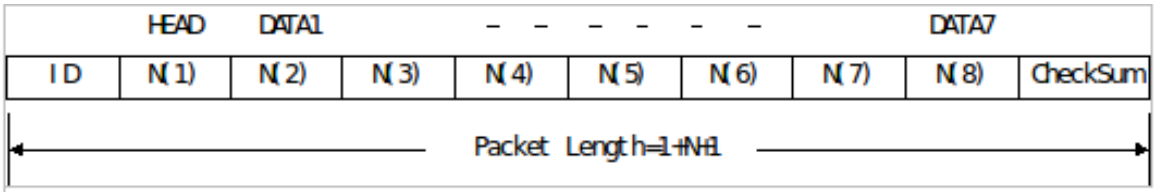
- **Baud Rate:** 115200 bps
- **Frame Format:**
  - 1 bit de inicio.
  - 8 bits de datos.
  - 1 bit de paridad impar.
  - un bit de parada.

FIGURA 16. Descripción del formato de bytes de la tarjeta Mindray.



Como lo muestra la ***FIGURA 16***, el formato de los datos entregados por la tarjeta van desde el bit más significativo (MSB) B[7] hasta el bit menos significativo(LSB) B[0].

FIGURA 17. Formato de paquetes enviados por la tarjeta Mindray.



Como se muestra en la **FIGURA 17** los paquetes de datos están conformados por varios bytes que cumplen una función específica como se explica a continuación:

ID: Identifica el paquete que se está enviando por la tarjeta mindray. Esta identificación es menor a 128(1 byte) con el fin de que sea único entre todos los bytes enviados, mientras que los otros bytes enviados por la tarjeta deben ser mayores o iguales a 128 para evitar confusión o concatenamiento no deseado de paquetes.

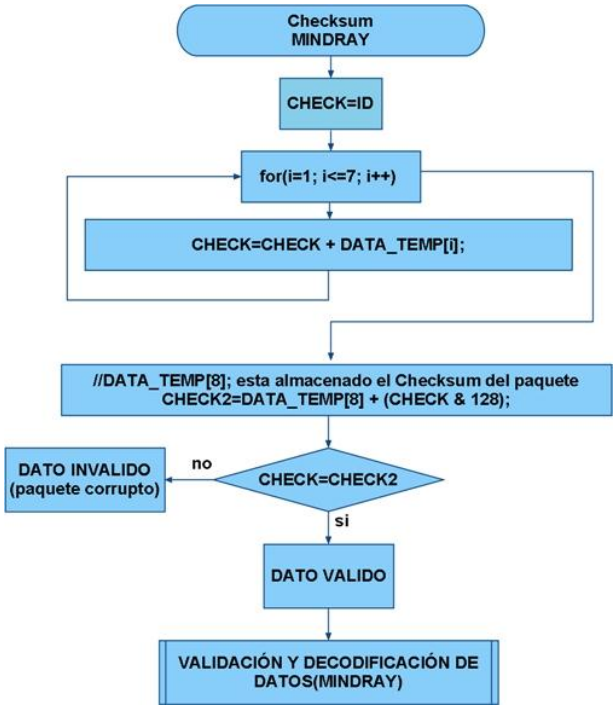
HEAD: Este byte viene después del byte de identificación ID, debe ser mayor o igual a 128, y en él se contienen los bits más significativos de

DATA1, DATA2.. Hasta DATA7, HEAD[0] seria el bit más significativo de DATA1 y HEAD[6] seria el bit más significativo de DATA7.

DATA(N): Corresponde al byte que contiene los datos o la información de cada parámetro de signos vitales enviados por la tarjeta OEM, este byte debe ser mayor o igual a 128 y su bit más significativo lo contiene el byte HEAD.

CHECKSUM: Este byte corresponde al método de detección de errores, que facilita identificar la integridad del paquete enviado por la tarjeta. Este checksum corresponde a la suma de los bytes anteriores sin tener en cuenta el bit más significativo en cada byte sumado y el desbordamiento aritmético que la suma genere. El bit más significativo de la suma debe remplazarse en el bit más significativo del CHECKSUM y si estos dos valores son iguales, el paquete habrá llegado correctamente.

FIGURA 18. DIAGRAMA DE FLUJO VERIFICACION DE DATOS POR CHECKSUM.



Código 1. Código base para la decodificación tarjeta mindray.

```
while(!EmptyUart1()){ //// si el la cola de recepción no está vacía desocúpela
```

```
        if(!EmptyUart1()){// se revisa que el buffer de entrada no este vacío

        ID = GetUart1();}// ID

        if(ID<128){ // el ID debe ser menor a 128, dependiendo de este se puede
reconocer

        // qué tipo de paquete es

        if(!EmptyUart1()){// cada dato se filtra por medio de la función lógica and
127 debido a que el bit más significativo de cada paquete, solo se utiliza para
diferenciar los bytes con el byte de ID.

        DATA_TEMP[0] = GetUart1() & 127;}// HEAD

        if(!EmptyUart1()){

        DATA_TEMP[1] = GetUart1()& 127;}//DATA1

        if(!EmptyUart1()){

        DATA_TEMP[2] = GetUart1()& 127;}//DATA2

        if(!EmptyUart1()){

        DATA_TEMP[3] = GetUart1()& 127;}//DATA3

        if(!EmptyUart1()){

        DATA_TEMP[4] = GetUart1()& 127;}//DATA4

        if(!EmptyUart1()){

        DATA_TEMP[5] = GetUart1()& 127;}//DATA5

        if(!EmptyUart1()){

        DATA_TEMP[6] = GetUart1()& 127;}//DATA6
```

```
        if(!EmptyUart1()){
        DATA_TEMP[7] = GetUart1()& 127;}//DATA7
        if(!EmptyUart1()){
        DATA_TEMP[8] = GetUart1()& 127;}//CHECKSUM
        // Se hace el CheckSum una vez se tienen los datos
        CHECK=ID;
        for(i=0; i<=7; i++){
        CHECK=CHECK+DATA_TEMP[i];
        }
        CHECK2=DATA_TEMP[8] + (CHECK & 128);
        if(CHECK2==CHECK){
// en esta parte se decodifica el paquete según el ID
// por cuestiones de confidencialidad de la FCV y Mindray Corp
// No es posible ver esta parte del código
        }


        }
}
```

Una vez se escribe el código C teniendo como base los algoritmos de decodificación (ver Código 1), se procede hacer implementacion y verificacion de

este utilizando el procesador NIOS II implementado en la FPGA Cyclone III de la tarjeta NEEK (ver FIGURA 2) por medio de descripcion de hardware.

Probado el codigo y hechas las correcciones se genera una libreria Mindray compatible y facil de implementar en una estructura software sobre el NIOS II.

Como resultado se obtienen los archivos:

- Mindray.h
- Mindray.c

TARJETA SUNTECHMED

La tarjeta mindray utiliza una interfaz RS-232 comunicándose a un host por medio de comunicación serial asíncrona, con la cual hace transmisión y recepción de datos a una velocidad de 9600 bps, los paquetes de byte enviados por la tarjeta cuentan con el método de detección de errores Checksum para garantizar la integridad de cada paquete de información enviado y recibido de la tarjeta.

Esta tarjeta es capaz de censar el parámetro fisiológico de NIBP (Presión no Invasiva).

PROTOCOLO DE COMUNICACIÓN SUNTECHMED:

Baud Rate: 9600 bps.
Frame Format:

- 1 bit de inicio.
- 8 bits de datos.
- No paridad.
- Un bit de parada.

Como lo muestra la ***FIGURA 16***, el formato de los datos entregados por la tarjeta Suntech desde el bit más significativo (MSB) B[7] hasta el bit menos significativo[LSB] B[0], lo que hace que el formato de bytes de la tarjeta suntech sea el mismo.

FIGURA 19. Formato de paquete de datos de la tarjeta SuntechMed.

MODULE START BYTE	PACKET BYTE	DATA BYTE(S)	CHECKSUM BYTE

En la

FIGURA 19 se muestran los paquetes de datos están conformados por varios bytes que cumplen una función específica como se explica a continuación:

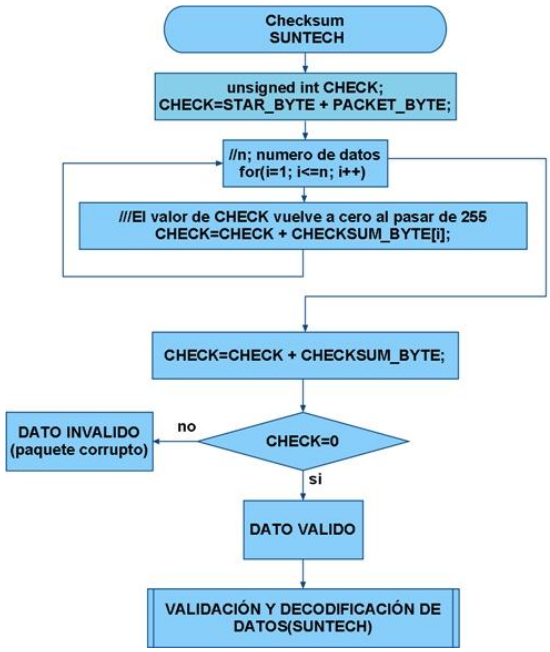
MODULE START BYTE: Este byte será siempre 0x3E si el modulo transmite al Host información y será siempre 0x3A si el host transmite datos hacia el modulo, lo que se debe tener en cuenta al enviar o recibir datos del host y desde la tarjeta SuntechMed. Este byte actúa como bandera de inicio de una trama de datos.

PACKET BYTE: Este byte es el indicador del paquete, o le da la identificación a la trama de datos, de acuerdo a este byte se conoce el tamaño en bytes de DATA.

DATA BYTE(s): Esta es una trama de bytes, puede contener comandos desde el host hacia el modulo, o datos de información del módulo al host, el DATA puede variar su tamaño de acuerdo al Indicador PACKET BYTE.

CHECKSUM BYTE: Este byte es el complemento faltante para que la suma de datos de START BYTE+PACKET BYTE + DATA BYTE(s) sea igual a cero sin tener en cuenta el overflow.

FIGURA 20. DIAGRAMA DE FLUJO VERIFICACION DE DATOS POR CHECKSUM.



Código 2: Código base para la decodificación tarjeta suntechmed.

```
while(!EmptyUart1()){ //// si el la cola no está vacía empiece a desocupar



        if(!EmptyUart1()){// se revisa que el buffer de entrada no este vacío

        NIBP_FLAG= GetUart1(); }//Bandera de inicio

        if(NIBP_FLAG==0x3E)// DATA INDICA START POR PARTE DE LA
SUNTECH; SI FUERA       DEL HOST A LA TARJETA ESTA SERIA 0X3A

        {

        if(!EmptyUart1()){

        ID_NIBP = GetUart1();}

        if(ID_NIBP==VALOR_ID)

        {
```

```
        for(i=0; i<N; i++)
        {
        if(!EmptyUart1()){
        DATA_IN_NIBP[i] = GetUart1();}
        }
        // CHECK SUM
        CHECKSUM=ID_NIBP+NIBP_FLAG;
        for(i=0; i<N; i++)
        {
        CHECKSUM=CHECKSUM+DATA_IN_NIBP[i];
        }
        if(CHECKSUM==0)
        {
        // en esta parte se decodifica el paquete según el ID
        // por cuestiones de confidencialidad de la FCV y SuntechMed
        // No es posible ver esta parte del código
        }
        }
        }
}
```

Una vez se escribe el código C teniendo como base los algoritmos de decodificación (ver Código 2), se procede hacer implementacion y verificación de este utilizando el procesador NIOS II implementado en la FPGA Cyclone III de la tarjeta NEEK (ver FIGURA 2) por medio de descripción de hardware.

Probado el código y hechas las correcciones se genera una librería Suntech compatible y fácil de implementar en una estructura software sobre el NIOS II.

Como resultado se obtienen los archivos

- Suntech.h
- Suntech.c

4.1.1.3 DESCRIPCION HDL DEL SISTEMA PARA LA GUI, UTILIZANDO EL SOPC BUILDER

La software que correrá en el sistema Nios II, debe decodificar, codificar y mostrar en la pantalla LCD de la tarjeta NEEK(ver FIGURA 2) los parámetros fisiológicos dados por las tarjetas OEM, además de esto debe ser capaz de controlar los eventos dados por la membrana touchscreen para hacer interacción con el usuario por medio de botones puestos en pantalla y leer imágenes de una memoria SD externa para la GUI renderizando logos o diseños gráficos complejos para objetos en pantalla. Debido a que el sistema debe soportar diferentes periféricos y debe comportarse a una velocidad adecuada para evitar colapsos del sistema o que este se torne lento, se plantearon los siguientes requerimientos hardware:

- Periférico para manejo de video en la pantalla LCD de 800*480.
- Periférico de adquisición de datos para tarjeta OEM.
- Periférico para manejo de membrana Touchscreen.
- Periférico para RAM DDR o SDRAM mayor o igual a 32Mb.
- Puerto JTAG de verification.
- Periférico para el control de una memoria SD.
- Periférico para lectura de memoria FLASH de 16MB.

El sistema hardware que contendrá la GUI está diseñado de acuerdo a los requerimientos planteados. Este se describe utilizando las recomendaciones básicas que se tienen al hacer diseño de sistemas confiables utilizando el procesador Nios II, como:

- Cambios de dominio de reloj (Sincronizadores de frecuencia) entre el Nios II y los periféricos.
- Arquitectura interna de la FPGA Cyclone III, Conocer el número de LEs y módulos hardware internos.
- Velocidades de periféricos (no exceder en frecuencia dominio de reloj de los periféricos) chip externos para no saturarlos.

FIGURA 21. HDL Generado, teniendo en cuenta las condiciones del diseño y los periféricos de la tarjeta NEEK, en la Herramienta SOPC Builder.

Module	Description	Clock
⊞ **cpu**	Nios II Processor	**pll_c0**
⊞ **cpu_ddr_clock_bridge**	Avalon-MM Clock Crossing Bridge	***multiple***
⊞ **slow_peripheral_bridge**	Avalon-MM Clock Crossing Bridge	***multiple***
⊞ **ddr_sdram**	DDR SDRAM Controller with ALTMEMPHY	**clk**
⊞ **lcd_sgdma**	Scatter-Gather DMA Controller	**ddr_sdram_sysclk**
⊞ **lcd_ta_sgdma_to_fifo**	Avalon-ST Timing Adapter	**ddr_sdram_sysclk**
⊞ **lcd_pixel_fifo**	On-Chip FIFO Memory	***multiple***
⊞ **lcd_ta_fifo_to_dfa**	Avalon-ST Timing Adapter	**pll_c0**
⊞ **lcd_64_to_32_bits_dfa**	Avalon-ST Data Format Adapter	**pll_c0**
⊞ **lcd_pixel_converter**	Pixel Converter (BGR0 --> BGR)	**pll_c0**
⊞ **lcd_32_to_8_bits_dfa**	Avalon-ST Data Format Adapter	**pll_c0**
⊞ **lcd_sync_generator**	Video Sync Generator	**pll_c0**
⊞ **pll**	PLL	**clk**
⊞ **sys_clk_timer**	Interval Timer	**pll_c2**
⊞ **sysid**	System ID Peripheral	**pll_c2**
⊞ **jtag_uart**	JTAG UART	**pll_c2**
⊞ **uart1**	UART (RS-232 Serial Port)	**pll_c2**
⊞ **lcd_i2c_scl**	PIO (Parallel I/O)	**pll_c2**
⊞ **lcd_i2c_en**	PIO (Parallel I/O)	**pll_c2**
⊞ **lcd_i2c_sdat**	PIO (Parallel I/O)	**pll_c2**
⊞ **touch_panel_pen_irq_n**	PIO (Parallel I/O)	**pll_c2**
⊞ **touch_panel_spi**	SPI (3 Wire Serial)	**pll_c2**
⊞ **sd_card_controller**	SD Card Controller (SPI)	**pll_c2**
⊞ **ext_flash**	Flash Memory Interface (CFI)	**pll_c0**

En el diseño de la

FIGURA 21 se puede observar la descripción de hardware que cumple con todas las condiciones de para un diseño fiable y de acuerdo a los requerimientos del sistema planteado anteriormente. En el esquema se puede observar conexiones a

distintos periféricos que de acuerdo a su función pueden ser de entrada, salida o bidireccionales.

La descripción de hardware hecha del sistema se basa en los ejemplos dados por Terasic Corp de la tarjeta NEEK y de los manuales, libros y tutoriales de la Altera Corp (Altera, Nios II Software Developer's Handbook NII5V2-10.0, 2010) (Altera, SOPC Builder User Guide UG-01096-1.0, 2010) (Altera, Embedded Peripherals IP User Guide, 2011).

DESCRIPCION DE PERIFERICOS HARDWARE DEL SISTEMA DESCRITO:

- **Nios II Processor:** En la **FIGURA 22** se pueden observar las conexiones que tiene la CPU al sistema, esta va unida a 2 Módulos sincronizadores de dominio de reloj que se conectan a la ram ddr y demás periféricos. El tipo de procesador escogido para el sistema de los tres que ofrece la gama de Nios II fue el Nios II/f (ver **FIGURA 23**), el cual cumple con todas las características necesarias para que el sistema sea rápido y cumpla con todos los requerimientos, el procesador también cuenta con multiplicadores hardware que ayudan al sistema a ser mucho más rápido en operaciones complejas.

FIGURA 22. Conexión de la CPU al sistema.

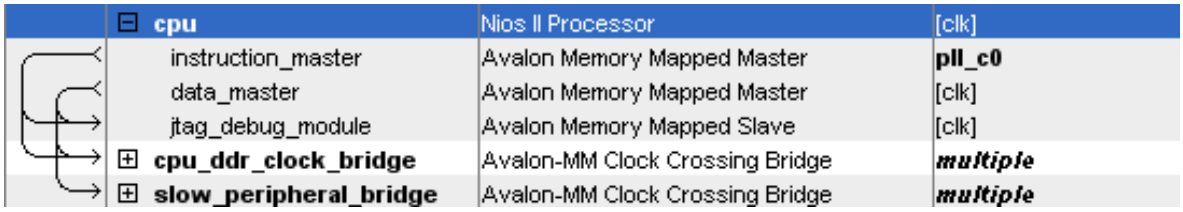
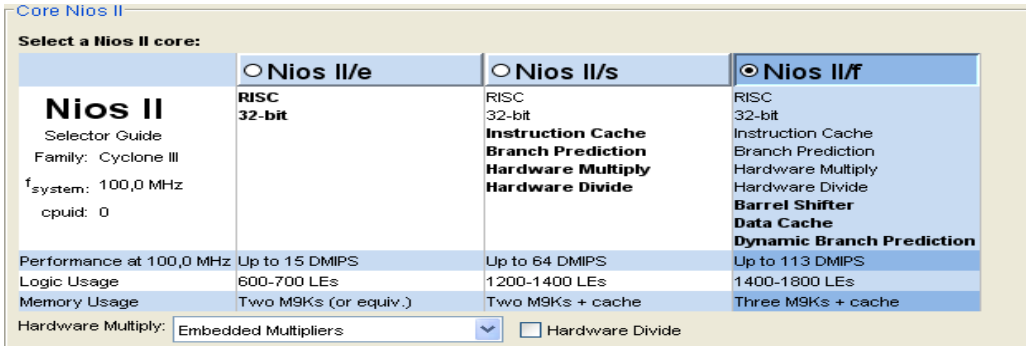


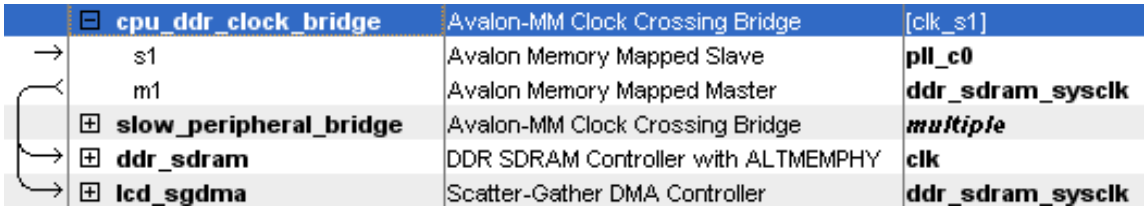
FIGURA 23. Tipo de CPU Nios II escogida para el diseño del sistema.



- **Avalon-MM Clock Crossing Bridge:** En el diseño de la
-
- **FIGURA 21** se observan 2 módulos Avalon-MM Clock Crossing Brigde, estos están encargados de la sincronización de datos entre la CPU y los periféricos. Este módulo se encarga de asegurar que los datos que se reciban y envíen a la CPU de y hacia los periféricos sean correctos, no tengan ruidos y no se pierdan. Esto lo logra gracias a que utiliza 2 módulos FIFO en su interior, uno con función de sincronización de salida y otro con función de sincronización de entrada, dependiendo de la diferencia de velocidades entre la CPU y los periféricos los módulos FIFO pueden variar su tamaño de cola lo que asegura la llegada y envió de datos desde los dos módulos hardware con diferente dominio de reloj. Los módulos Clock Crossing Bridge son muy importantes en el diseño confiable de sistemas hardware, debido a que en el sistema se debe asegura los datos de salida y de entrada de datos para no tener ruidos indeseados y como consecuencia una mala interpretación de los mismos

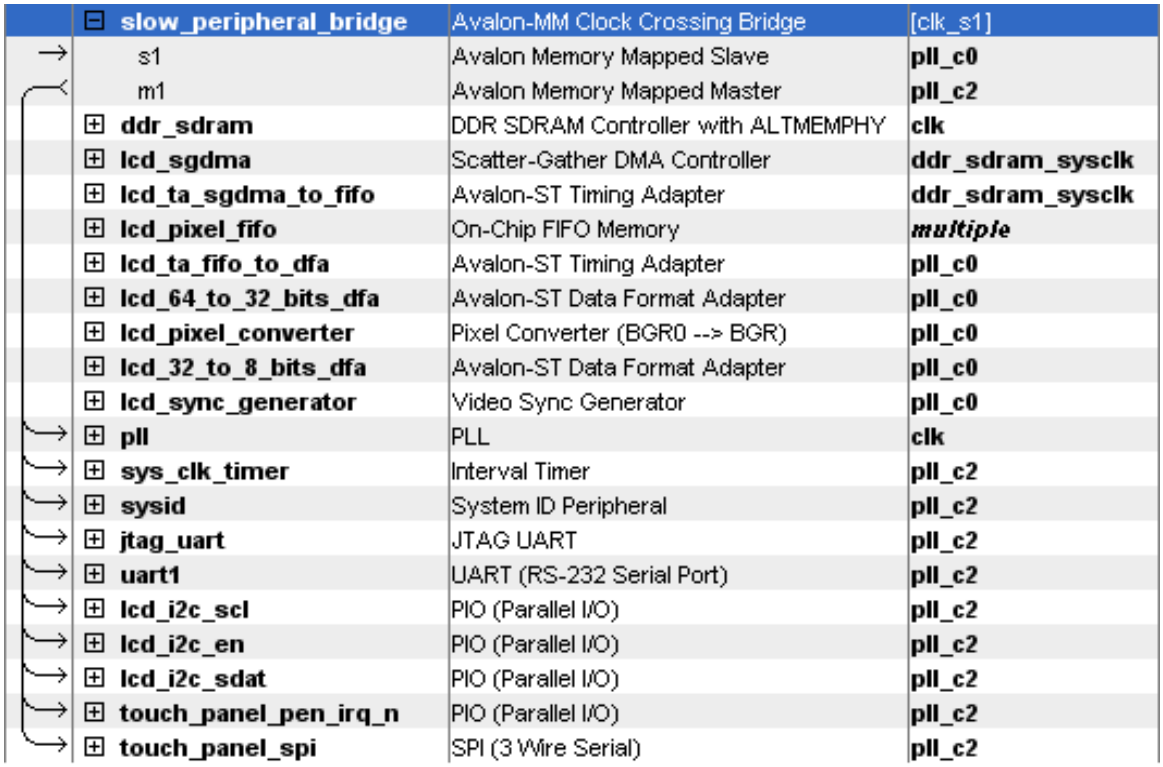
  El “cpu_ddr_clock_bridge“sincroniza los datos de entrada y salida de la CPU, la DDR SDRAM de la tarjeta NEEK y el modulo “lcd_sgma“. Si se observa en la **FIGURA 24** se puede ver la conexión entre los periféricos y el Avalon y se puede observar el cambio de relojes entre estos.

FIGURA 24. Sincronización de datos entre la CPU, la DDR y el lcd sgdma, utilizando el cpu_ddr_clock_bridge.



El “slow_periphelal_birgde“ sincroniza los datos de entrada y salida de la CPU, el pll, el Interval Timer, el System ID, JTAG, UART1 y los demás periféricos hardware que contiene el sistema (ver **FIGURA 25**).

FIGURA 25. Sincronización de datos entre la CPU, los demas periféricos utilizando el puente de slow_peripheral_bridge.



	slow_peripheral_bridge	Avalon-MM Clock Crossing Bridge	[clk_s1]
→	s1	Avalon Memory Mapped Slave	**pll_c0**
←	m1	Avalon Memory Mapped Master	**pll_c2**
	ddr_sdram	DDR SDRAM Controller with ALTMEMPHY	**clk**
	lcd_sgdma	Scatter-Gather DMA Controller	**ddr_sdram_sysclk**
	lcd_ta_sgdma_to_fifo	Avalon-ST Timing Adapter	**ddr_sdram_sysclk**
	lcd_pixel_fifo	On-Chip FIFO Memory	***multiple***
	lcd_ta_fifo_to_dfa	Avalon-ST Timing Adapter	**pll_c0**
	lcd_64_to_32_bits_dfa	Avalon-ST Data Format Adapter	**pll_c0**
	lcd_pixel_converter	Pixel Converter (BGR0 --> BGR)	**pll_c0**
	lcd_32_to_8_bits_dfa	Avalon-ST Data Format Adapter	**pll_c0**
	lcd_sync_generator	Video Sync Generator	**pll_c0**
→	pll	PLL	**clk**
→	sys_clk_timer	Interval Timer	**pll_c2**
→	sysid	System ID Peripheral	**pll_c2**
→	jtag_uart	JTAG UART	**pll_c2**
→	uart1	UART (RS-232 Serial Port)	**pll_c2**
→	lcd_i2c_scl	PIO (Parallel I/O)	**pll_c2**
→	lcd_i2c_en	PIO (Parallel I/O)	**pll_c2**
→	lcd_i2c_sdat	PIO (Parallel I/O)	**pll_c2**
→	touch_panel_pen_irq_n	PIO (Parallel I/O)	**pll_c2**
→	touch_panel_spi	SPI (3 Wire Serial)	**pll_c2**

- **DDR SDRAM:** Este módulo está encargado de gestionar la comunicación entre la RAM de 32Mb de la tarjeta NEEK y la CPU por medio de los sincronizadores Avalon.
- **Scatter-Gather DMA Controller:** Esta encargado de gestionar los datos entre la RAM y la CPU hacia el “lcd_ta_sgma_to_fifo“(ver **FIGURA 26**). Este módulo puede leer y escribir datos en la DDR SDRAM en un bloque especifico de memoria (Altera, Embedded Peripherals IP User Guide, 2011), donde se guardan y leen las imágenes que se visualizaran en la pantalla LCD por medio del módulo de sincronización de video. En la **FIGURA 27** se puede observar la configuración típica para la adaptación y sincronización de video, el diseño de la descripción de hardware hecha por

medio del SOPC-Builder se basa en los ejemplos dados por Terasic corp en su documentación para el uso de la tarjeta NEEK (Altera, My First FPGA Design Tutorial, 2008) (Altera, My First Nios II Software Tutorial, 2008).

FIGURA 26. Conexión entre la CPU, la RAM y el Scatter-Gather DMA Controller.

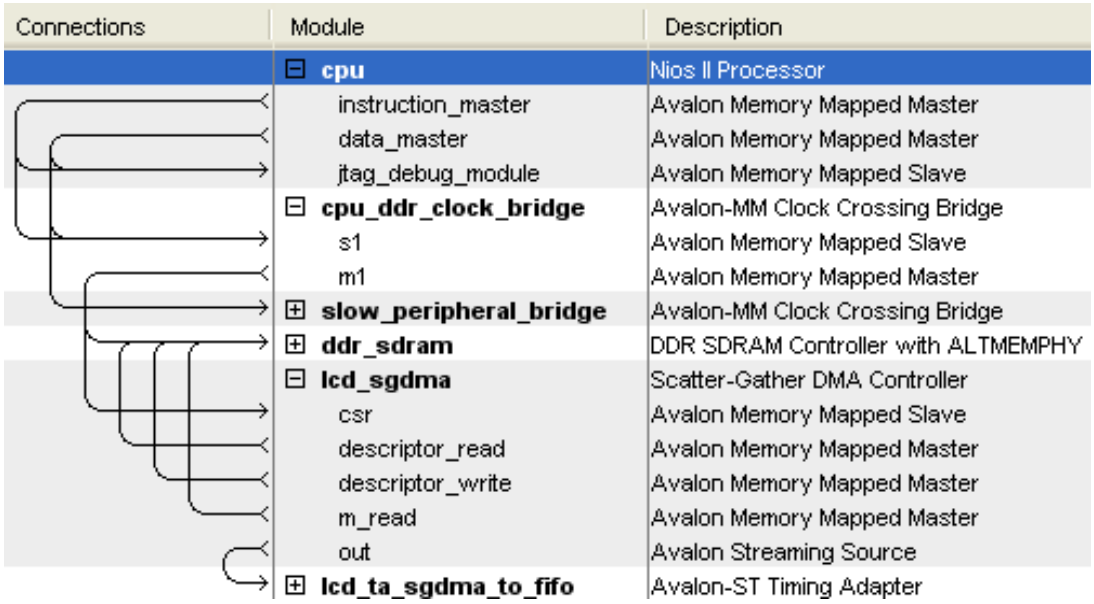
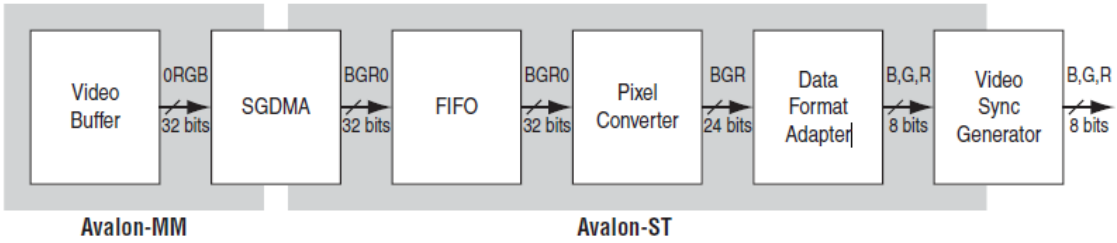


FIGURA 27. Configuración recomendada para la adaptación y sincronización de video utilizando el procesador Nios II- imagen tomada de Embedded Peripherals IP User Guide-UG-01085-11.0 de junio de 2011- Altera Corp.

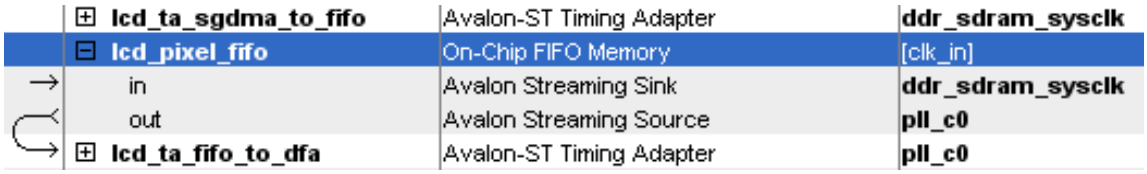


- **Avalon-ST Timing Adapter:** Este módulo está encargado de adaptar en tiempo los datos enviados de un módulo a otro formando paquetes de información, en este diseño existen dos módulos de adaptación:

- **lcd_ta_sgdma_to_fifo:** Se encarga de adaptar los datos seriales que provienen del módulo lcd_sgdma hacia el módulo FIFO “lcd_pixel_fifo“.
- **lcd_ta_fifo_to_dfa:** Se encarga de adaptar los paquetes de datos seriales que provienen del módulo lcd_pixel_fifo hacia el lcd_64_to_32_bits_dfa.

- **On-Chip FIFO Memory:** Este módulo sincroniza los datos enviados desde el módulo lcd_ta_sgdma_to_fifo, hacia el módulo lcd_ta_fifo_to_dfa. Este módulo es muy similar a los Avalon-MM Clock Crossing debido que también sirve para sincronizar datos, a diferencia del Avalon este solo sirve para sincronizar datos de salida. Si se observa en la **FIGURA 28** se puede observar que el FIFO utiliza 2 dominios de reloj el de la DDR SDRAM y el que utiliza el sincronizador de video. Con este FIFO se hace una sincronización de Alta a baja frecuencia(High to Low).

FIGURA 28. Sincronizador de Datos FIFO, High to Low.



- **Avalon-ST Data format Adapter:** Reduce el ancho de los paquetes, en el diseño existen dos de estos módulos:

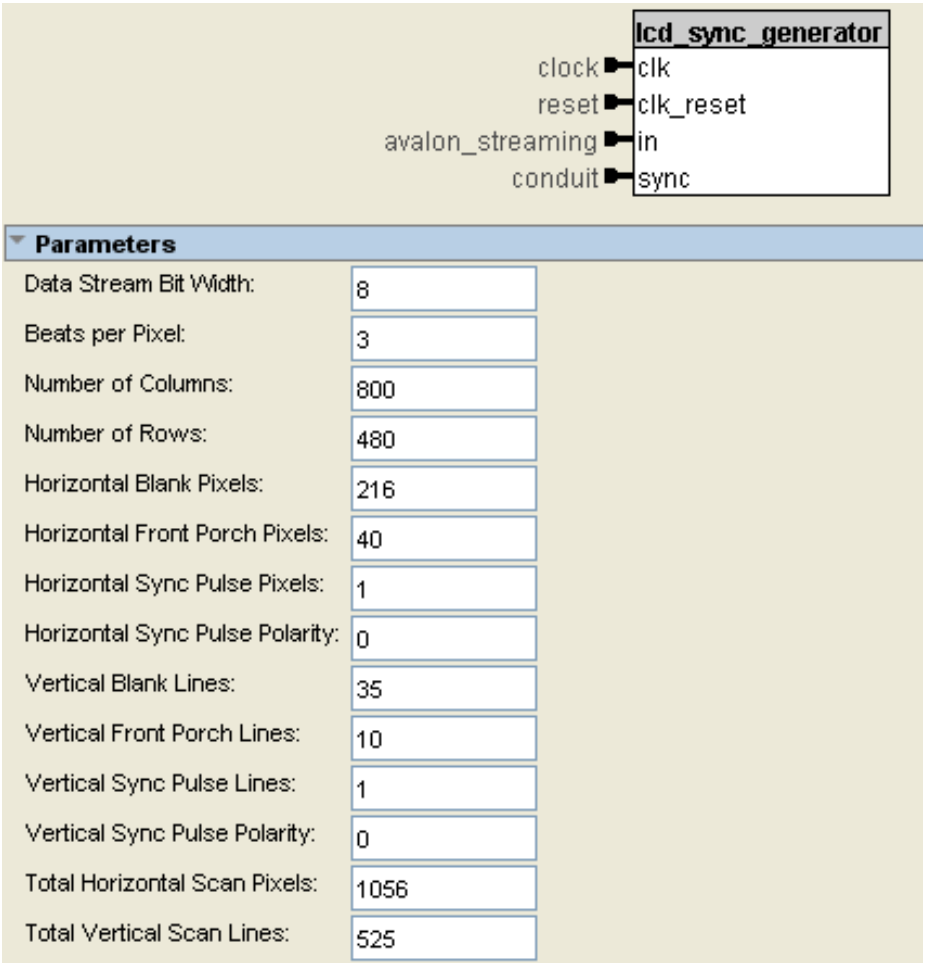
  - **lcd_64_to_32_bits_dfa:** Convierte el ancho de paquetes de 64 bits provenientes del FIFO a paquetes de 32 bits de datos que recibe módulo lcd_pixel_converter.

  - **lcd_32_to_8_bits_dfa:** Convierte el ancho de paquetes de 32 bits provenientes del lcd_pixel_converter a paquetes de 8 bits de datos que recibe el módulo lcd_sync_generator.

- **Pixel Converter:** Convierte los paquetes provenientes del módulo lcd_32_to_8_bits_dfa a un formato compatible con el módulo de sincronización de video“lcd_sync_generator“.
- **Video Sync Generator:** Modulo de sincronización de video, se encarga de enviar los datos a la pantalla LCD de la tarjeta NEEK para visualizar la GUI,

la configuración de este módulo depende de las especificaciones de tiempo de la pantalla, esta configuración se puede observar en la **FIGURA 29**.

FIGURA 29. Especificaciones de tiempo para el módulo de sincronización de video de la pantalla LCD de la tarjeta NEEK.



- **PLL:** Este módulo se encarga de configurar el PLL interno que existe dentro de la FPGA para generar relojes con unas frecuencias específicas a partir de un oscilador externo de 50 MHZ. Los relojes arrojados por el PLL pueden ser mayores o menores al reloj fuente, en este caso se generan 3 relojes a partir del oscilador de 50MHZ.

- Pll_C0: reloj con frecuencia de 100MHZ generado por el PLL a partir del reloj de 50Mhz.
- Pll_C1: reloj con frecuencia de 100MHZ desfasado -65 deg con respecto al Pll_C0.
- Pll_C2: reloj con frecuencia de 80MHZ generado por el PLL a partir del reloj de 50Mhz.

- **Interval Timer:** Generador de tiempos cada 10 ms a partir de un contador de 32 bits. Este módulo se utiliza para activar procedimientos software cuando se active la interrupción por timer por medio de software en el Nios.

- **Sytem ID Peripheral:** Esta encargado de dar una identificación al sistema, para que el compilador lo reconozca en el momento que se necesite descargar una aplicación software compatible por medio del cable JTAG. Este ID es similar al valor Cheksum para que cuando se descargue un ejecutable desde el compilador Nios II IDE, este concuerde con las características del sistema.

- **JTAG:** Modulo encargado de gestionar la comunicación entre el compilador Nios II IDE y el sistema, este módulo es uno de los más importantes ya que sin él no se podría programar la memoria del sistema.

- **UART(RS-232 Serial port):** Se encarga de gestionar la comunicación entre el chip de interfaz RS232 de la tarjeta NEEK con el procesador, para recibir y enviar datos desde y hacia un hardware externo utilizando comunicación serial.

  Este módulo tiene que configurarse de acuerdo a la velocidad de transferencia de datos que utilice el hardware externo para establecer una comunicación entre las tarjetas OEM. Esta velocidad de transferencia puede dejarse fija o cambiarse desde software.

- **PIO Parallel IO:** Este módulo se encarga de asignar al procesador puertos de salida de múltiples propósitos, estos pueden ser entrada, salida o bidireccionales según sea la aplicación o requerimientos del diseño.

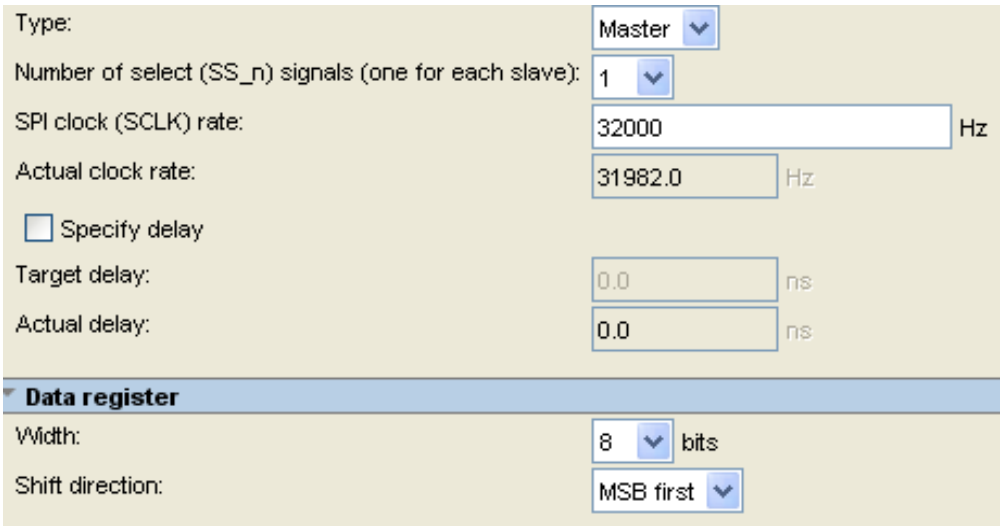
  En el sistema se tienen varios PIO, los cuales son:

  - **lcd_i2c_scl:** Es un puerto de salida de 1 bit, que ayuda a controlar la comunicación I2C entre la pantalla y el procesador, está encargado de inicializar la pantalla.

- **lcd_i2c_en:** Es un puerto de salida de 1 bit, encargado de habilitar la comunicación entre la pantalla y el procesador para su inicialización.
- **lcd_i2c_sdat:** Es un puerto bidireccional de 1 bit, encargado de recibir y enviar los datos de inicialización de pantalla.
- **touch_panel_pen_irq_n:** Es un puerto de entrada de 1 bit encargado de registrar el flanco descendente de la interrupción del driver externo de la membrana touchscreen de 4 hilos, con el motivo de conocer el momento en que se presenta un evento Touch en la pantalla y poder capturarlo.

- **SPI:** Modulo hardware encargado de gestionar la comunicación serial entre el driver touch externo que hace la conversión de datos analógicos a digitales y los envía en el protocolo SPI, cada paquete recibido por este puerto de comunicaciones tiene un tamaño de 8 bits entregados desde el más significativos hasta el menos significativo a una velocidad de 32Khz por muestra (ver **FIGURA 30**).

FIGURA 30. Configuración del módulo SPI de comunicación con el driver de la membrana TouchScreen.

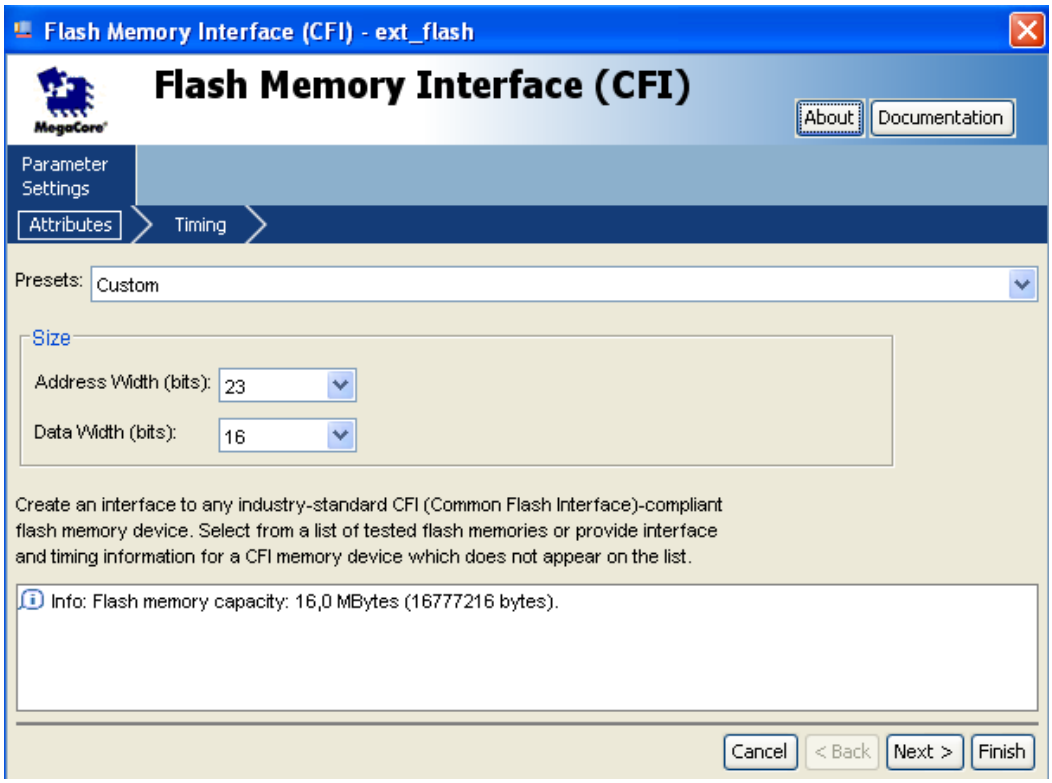


- **SD Card Controller (SPI):** Modulo hardware encargado de establecer la comunicación entre el procesador Nios II y la memoria SD, para la lectura y escritura de archivos sobre una memoria no volátil externa, el driver para controlar este periférico lo facilita Terasic Corp en sus ejemplos los cuales se pueden descargar de la página de la compañía. El uso de este hardware en el sistema permite al programador leer y escribir archivos desde una memoria

externa sin necesidad de utilizar el recurso de la memoria FLASH, lo que reduce el ciclo de vida útil de la misma.

- **Flash Memory Interface (CFI):** Modulo hardware encargado de gestionar los datos que contiene la memoria FLASH (ver **FIGURA 31**) de la tarjeta NEEK que almacena descripción hardware, el software y archivos del sistema, este módulo puede ser configurado para diferentes tamaños de memorias no volátiles, en este caso se utiliza una memoria de 16 MB. Básicamente en la memoria FLASH solo es recomendable almacenar la configuración de sistema en su hardware y su software, no es recomendado guardar archivos de propósitos generales.

FIGURA 31. Configuración modulo Flash Memory Interface(CFI), para la lectura de memoria FLASH de 16MB.



Una vez se termina el hardware descrito por medio de la herramienta SOPC Builder se añade por medio de la instanciación verilog (ver **Código 3**) al proyecto Quartus II donde se configura la FPGA de acuerdo a los componentes LCD, Touchscreen,SDRAM, FLASH, SD card y UART de la tarjeta NEEK.

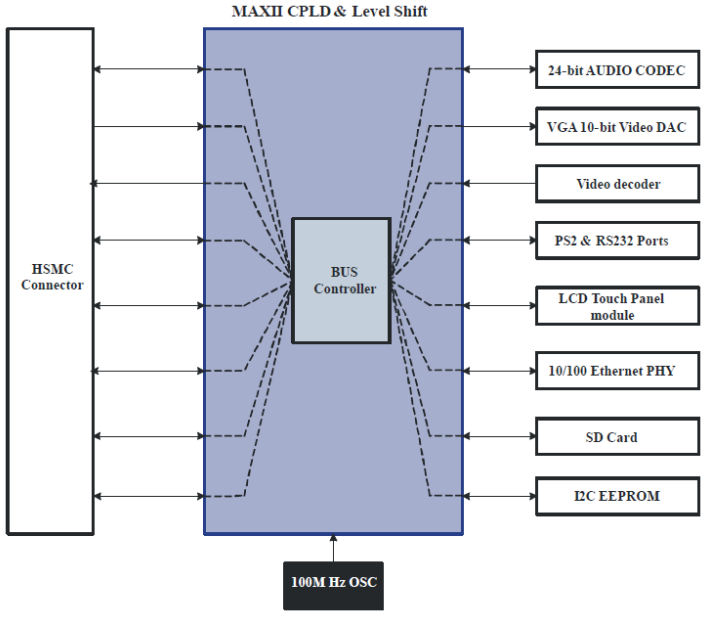
Código 3. Instanciación verilog del sistema.

```
niosII_sopc procesador_del_sistema

(

.clk (top_clkin_50),// CLK del sistema

.pll_c0_out (top_clk_to_offchip_video),// CLK VIDEO

.pll_c2_out (top_peripheral_clk),// CLK perifericos

.reset_n (top_reset_n),//Reset del sistema

// VIDEO

.DEN_from_the_lcd_sync_generator (top_HC_DEN),

.HD_from_the_lcd_sync_generator (top_HC_HD),

.RGB_OUT_from_the_lcd_sync_generator (top_HC_LCD_DATA),

.VD_from_the_lcd_sync_generator (top_HC_VD),

//LCD

.bidir_port_to_and_from_the_lcd_i2c_sdat (top_HC_SDA),

.out_port_from_the_lcd_i2c_en (top_out_port_from_the_lcd_i2c_en),

.out_port_from_the_lcd_i2c_scl (top_out_port_from_the_lcd_i2c_scl),

// TOUCH SCREEN

.in_port_to_the_touch_panel_pen_irq_n (top_HC_ADC_PENIRQ_N),

.MISO_to_the_touch_panel_spi (top_HC_ADC_DOUT),

.MOSI_from_the_touch_panel_spi (top_HC_ADC_DIN),
```

```
.SCLK_from_the_touch_panel_spi (top_SCLK_from_the_touch_panel_spi),

.SS_n_from_the_touch_panel_spi (top_SS_n_from_the_touch_panel_spi),



//UART

.rxd_to_the_uart1 (top_HC_UART_RXD),

.txd_from_the_uart1 (top_HC_UART_TXD),

//Memorira SD Card

.spi_clk_from_the_sd_card_controller (top_HC_SD_CLK),

.spi_cs_n_from_the_sd_card_controller (top_HC_SD_DAT3),

.spi_data_in_to_the_sd_card_controller (top_HC_SD_DAT),

.spi_data_out_from_the_sd_card_controller (top_HC_SD_CMD),

// DDR SDRAM

.global_reset_n_to_the_ddr_sdram (top_reset_n),

.local_init_done_from_the_ddr_sdram (top_local_init_done_from_the_ddr_sdram),

.local_refresh_ack_from_the_ddr_sdram (top_local_refresh_ack_from_the_ddr_sdram),

.local_wdata_req_from_the_ddr_sdram (top_local_wdata_req_from_the_ddr_sdram),

.mem_addr_from_the_ddr_sdram (top_mem_addr),

.mem_ba_from_the_ddr_sdram (top_mem_ba),

.mem_cas_n_from_the_ddr_sdram (top_mem_cas_n),

.mem_cke_from_the_ddr_sdram (top_mem_cke),

.mem_clk_n_to_and_from_the_ddr_sdram (top_mem_clk_n),

.mem_clk_to_and_from_the_ddr_sdram (top_mem_clk),

.mem_cs_n_from_the_ddr_sdram (top_mem_cs_n),

.mem_dm_from_the_ddr_sdram (top_mem_dm),
```

```
.mem_dq_to_and_from_the_ddr_sdram (top_mem_dq),
.mem_dqs_to_and_from_the_ddr_sdram (top_mem_dqs),
.mem_ras_n_from_the_ddr_sdram (top_mem_ras_n),
.mem_we_n_from_the_ddr_sdram (top_mem_we_n),
.reset_phy_clk_n_from_the_ddr_sdram (top_reset_phy_clk_n_from_the_ddr_sdram),
.ddr_sdram_aux_full_rate_clk_out (top_ddr_sdram_aux_full_rate_clk_out),
.ddr_sdram_aux_half_rate_clk_out (top_ddr_sdram_aux_half_rate_clk_out),
.ddr_sdram_phy_clk_out (top_ddr_sdram_phy_clk_out)
);
```

FIGURA 32. Conexión de la CPLD MAX II a la FPGA Cylone III por medio del conector HSMC.



La FPGA Cyclone III de la tarjeta de desarrollo de Terasic no está conectada directamente a los periféricos de video, audio y comunicaciones, esta está conectada a una CPLD MAX II que funciona como interfaz para aumentar el número de pines de la Cyclone III y de esta manera manejar una mayor cantidad

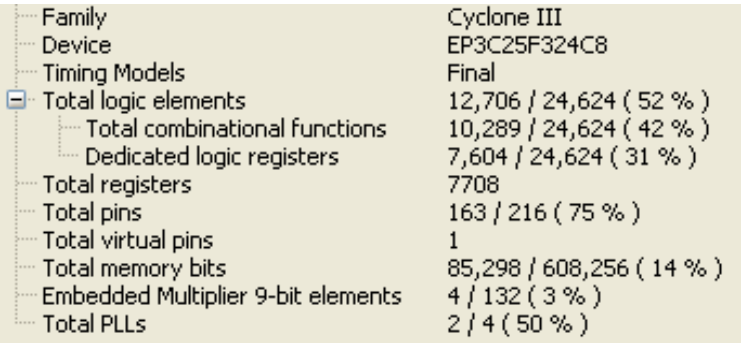
de dispositivos externos. El diagrama de la conexión en la tarjeta NEEK se puede ver su manual de usuario (Altera, Cyclone III FPGA Starter Board Reference Manual, 2008) y en la **FIGURA 32**.

4.1.1.4 RESUMEN DEL SISTEMA HDL IMPLEMENTADO EN LA FPGA CYCLONE III

Cuando se diseñan sistemas HDL utilizando la herramienta Quartus II y el SOPC, es opcional tener en cuenta el reporte final que el compilador arroja al finalizar la implementacion del sistema, ya que es importante conocer cuántos recursos puede llegar a ocupar la descripción de hardware que se va a implementar.

Como se muestra en la **FIGURA 33** de la familia Cyclone III con el dispositivo FPGA de la tarjeta NEEK (EP3C25F324C8) y el diseño HDL descrito para la GUI, solo se hizo uso del 52% de los LEs disponibles, del 75 % de los GPIOs y del 14% de memoria MK9 embebida.

FIGURA 33. Reporte del uso de LEs, según el HDL implementado.



Family	Cyclone III
Device	EP3C25F324C8
Timing Models	Final
Total logic elements	12,706 / 24,624 (52 %)
Total combinational functions	10,289 / 24,624 (42 %)
Dedicated logic registers	7,604 / 24,624 (31 %)
Total registers	7708
Total pins	163 / 216 (75 %)
Total virtual pins	1
Total memory bits	85,298 / 608,256 (14 %)
Embedded Multiplier 9-bit elements	4 / 132 (3 %)
Total PLLs	2 / 4 (50 %)

No menos importante es conocer un estimado de consumo de potencia del sistema central de procesamiento, porque según la arquitectura (nm) de la FPGA, las condiciones de frecuencia y el número de LEs activos, la disipación de potencia en el dispositivo lógico programable puede variar y afectar las condiciones de calor del sistema hardware que suministre energía a la FPGA. En este caso según la herramienta PowerPlay Analyzer (ver **FIGURA 34**) de Altera el consumo de potencia será de un estimado de 626mW que comparado con otros sistemas embebidos de bajo consumo basados en arquitectura ARM que consumen alrededor de 1W de potencia, demuestra que los sistemas basados sobre arquitecturas FPGAs consumen bajos niveles de potencia que pueden ser

incluso reducidos si se llegara hacer un diseño enfocada a aceleradores de hardware que ayuden al procesador software para reducir la necesidad de frecuencias altas de reloj aprovechando la ventaja de paralelismo que ofrece esta tecnología (Frazer, 2008).

Todo el análisis de resultados arrojados por el compilador Quartus II permite que a futuro se puedan hacer modificación para mejorar el sistema, si se desean añadir nuevos periféricos HDL, procesador Nios II para generar arquitecturas multicore, aceleradores hardware, GPU (Graphics Processor Unit) u otro hardware que pueda aumentar el rendimiento del mismo, teniendo en cuenta el nivel de uso de la FPGA en LEs y la potencia disipada.

FIGURA 34. Reporte PowerPlay Power Analyzer, estimado de consumo de potencia según el HDL generado.

Power Models	Final
Total Thermal Power Dissipation	626.43 mW
Core Dynamic Thermal Power Dissipation	262.82 mW
Core Static Thermal Power Dissipation	95.37 mW
I/O Thermal Power Dissipation	268.24 mW

Una vez el diseño HDL se finalizó, se programa el dispositivo lógico programable por medio de la interfaz JTAG utilizando el programador del Quartus II, en ese momento el hardware del sistema está listo para ejecutar cualquier software compatible desarrollado en lenguaje C/C++ en la interfaz de desarrollo Nios II IDE.

4.1.1.5 PROGRAMACION DEL PROCESADOR NIOS II PARA CONSTRUIR UNA INTERFAZ GRAFICA DE USUARIO

En la GUI del monitor de signos vitales se visualizan botones, señales y números con los cuales el usuario puede interactuar y observar los parámetros fisiológicos arrojados por las tarjetas OEM. El sistema hardware descrito en el capítulo (4.1.1.3 DESCRIPCION HDL DEL SISTEMA PARA LA GUI, UTILIZANDO EL SOPC BUILDER) está en capacidades para generar dibujos en pantalla e interactuar con el usuario por medio de la membrana táctil de la LCD. El programador de software debe conocer el funcionamiento de los periféricos que controlan la pantalla (lcd_sgma) y de esta manera escribir algoritmos capaces de comunicarse con el

framework del sistema para el dibujar en memoria las figuras geométricas necesarias para generar una interfaz gráfica de usuario.

La compañía Terasic Corp quien fue la encargada de diseñar la tarjeta NEEK facilita al programador librerías graficas las cuales puede implementar en cualquier proyecto para dibujar cualquier figura geométrica básica como líneas, triángulos, cuadrados, texto, etc. En el proyecto para generar la GUI del monitor de signos vitales se hace implementacion de las librerías graficas que ofrece Terasic en sus ejemplos.

4.1.1.6 IMPLEMENTACIÓN LIBRERIAS PANTALLA LCD Y TOUCHSCREEN

IMPLEMENTACIÓN LIBRERIAS PANTALLA LCD:

Para implementar las librerías graficas del sistema se debe conocer ¿cómo? inicializar y controlar el hardware lcd_sdgma del sistema para definir el espacio de memoria donde se alojaran las imágenes en frames generadas por el algoritmo. Para inicializar la pantalla LCD se deben utilizar los drivers y las variables de video definidas en las librerías escritas por Terasic que se nombran a continuación:

Librerías Graficas:

Contienen los algoritmos de dibujo, con los cuales se puede dibujar formas geométricas en el espacio de memoria de video.

- simple_graphics.h
- simple_graphics.c
- simple_text.h

Librerías para control de video:

Contienen las estructuras de las variables de video y los algoritmos de inicialización de la pantalla LCD.

- alt_tpo_lcd.h
- alt_tpo_lcd.c
- alt_tpo_lcd_console.c
- alt_video_display.h
- alt_video_display.c

ALGORITMO PARA EL DIBUJO EN LA PANTALLA LCD:

- Se define el puntero “display“ que contiene la estructura con las configuraciones de pantalla lista para ser inicializada y apunta al espacio de memoria definido que contendrá el video.
- Se utiliza la función de inicialización de video, que inicializa el puntero de “display“ con las configuraciones de resolución, espacio de color, dispositivo hardware y numero de frames de video.
- Se dibuja en pantalla utilizando las librerías gráficas.
- Se registra el dibujo en un frame de la pantalla.

Código 4. Código de inicialización y dibujo de figuras geométricas pantalla.

```
int main()
{
        // Puntero para la inicialización de la pantalla
        alt_video_display* display;

        // Se inicializa y se asigna el hardware de lcd_sgma
        // al puntero display, se inicializa el marco de trabajo
        // para dibujar en la pantalla LCD, se define el número de frames
        // la resolución el espacio de color a la que va funcionar la
        // pantalla.
        display = alt_video_display_init( ALT_VIDEO_DISPLAY_SGDMA_NAME,
                                          // Nombre del periférico
                                          ALT_VIDEO_DISPLAY_COLS,
                                          // Ancho en pixeles 800
                                          ALT_VIDEO_DISPLAY_ROWS,
                                          // alto en pixeles 480
                                          ALT_VIDEO_DISPLAY_COLOR_DEPTH,
                                          // profundidad de color
                                          ALT_VIDEO_DISPLAY_USE_HEAP,
                                          // locación del buffer
                                          ALT_VIDEO_DISPLAY_USE_HEAP,
                                          // descripción del buffer
                                          2 );
                                          // número de frames de video
        while(1)// Se inicia un bucle infinito
        {
        //se dibuja un rectángulo de color verde solido en la posición
        //(0,0) con un ancho de 100 pixeles y un alto de 150px
```

```
        // Sobre el display
        vid_draw_box (0, 0, 100, 150, GREEN_24, DO_FILL,display);

        //se dibuja en pantalla el texto “hola“ en color GRIS
        //en la posición (100,100) con el tipo de letra courier tamaño
        //10, sobre el display
        vid_print_string(100, 100,GRAY_24, cour10_font, display,"hola");

        //se dibuja un ciculo de color magneta sin relleno en
        //la posicion (400,200) con un radio de 100px
        vid_draw_circle(400, 200, 100, MAGENTA_24, DO_NOT_FILL,display);
        //se registra el dibujo en el espacio de memoria visible en
        // la pantalla LCD
        alt_video_display_register_written_buffer( display );
        while ( alt_video_display_buffer_is_available( display ) != 0 )
        { }
        }
// Se cierra el espacio de memoria asignado a video.
alt_video_display_close( display,
ALT_VIDEO_DISPLAY_USE_HEAP,
ALT_VIDEO_DISPLAY_USE_HEAP );
return 0;
}
```

En el **Código 4** se observa, la serie de pasos necesarios en lenguaje ANSI-C para inicializar la pantalla LCD y dibujar en pantalla. Las librerías graficas ofrecen más funciones de dibujo con las cuales se pueden generar formas geométricas más complejas como triángulos, textos alpha, imágenes y polígonos irregulares. Esta librería facilita el diseño de la GUI donde son visibles todos los parámetros fisiológicos que se adquieran por medio de cada tarjeta OEM.

IMPLEMENTACIÓN LIBRERIAS TOUCHSCREEN:

Los eventos touch generados debido a la interacción del usuario con la membrana táctil de la pantalla LCD son necesarios en la GUI ya que en ella habrán dibujados botones u otros objetos de interacción que necesitan conocer ¿en qué parte? ha sido oprimida la pantalla para que de esta manera puedan activarse o no y cumplir cierta funcionalidad si así se requiere. Terasic corp utiliza en sus ejemplos librerías que contienen funciones que ayudan al programador a identificar cada vez que

exista un evento touch, estas librerías se utilizaran en el monitor de signos vitales para agilizar el proceso de diseño y desarrollo de la GUI.

Se implementaron estas librerías utilizando un hardware compatible generado con la herramienta SOPC builder, este hardware tienen el nombre de **touch_panel_spi** y **touch_panel_irq_n** los cuales se encuentran en la capa física del sistema (ver

FIGURA 21) y hacen posible la recepción de datos por parte del hardware externo de la membrana touch. Las librerías hacen posible la comunicación con el framework para controlar, decodificar los datos y de esta forma identificar cada evento touch generado.

Libraries Touch:

- alt_touch_panel.h
- alt_touch_panel.c
- alt_touchscreen.h
- alt_touchscreen.c

ALGORITMO PARA LA CAPTURAR EVENTOS TOUCH:

- Se define la variable “touch_screen“ que contiene la estructura touch necesaria para la captura de eventos.
- Se inicializa el hardware de comunicación **touch_panel_spi** y **touch_panel_irq_n**, por medio de la función de inicialización de las librerías touch.
- Se calibra el ADC por medio de funciones, para definir valores de lectura del ADC correspondientes a dos puntos coordenados diagonales en la pantalla.
- Se capturan e interpretan los eventos touch para que el procesador realice una función específica.

Codigo 5. Algoritmo para la captura e interpretación de eventos touch y dibujo de botón simple.

```
int main()
{
        int result;
        int pen_down;
        int x;
        int y;
        // Puntero para la inicialización de la pantalla
        alt_video_display* display;
```

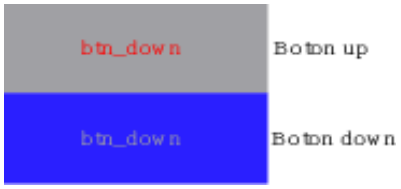
```
//Variable touch_screen la cual contiene la estructura de tipo
//alt_touchscreen encargada de almacenar los datos de los eventos
//touch
alt_touchscreen touch_screen;

//Función de inicialización de la variable touch_screen y
// el hardware touch_panel_spi y touch_panel_irq_n
result = alt_touchscreen_init(&touch_screen,
                    //el puntero interno apunta a la
                    //variable touch_screen para
                    //modificarla e inicializarla
                    TOUCH_PANEL_SPI_BASE,
                    //Se ubica la dirección del hardware
                    //touch_panel_spi para su
                    //inicialización
                    TOUCH_PANEL_SPI_IRQ,
                    // se define el grado de interrupción
                    // del hardware touch_panel_irq_n
                    TOUCH_PANEL_PENIRQ_N_BASE,
                    //Se ubica la dirección del hardware
                    // touch_panel_irq_n
                    100,
                    //se definen la muestras por segundo
                    // que se deben tomar del ADC
                    ALT_TOUCHSCREEN_SWAP_XY);
//se define el número de intercambios
//en el muestreo entre X y Y

// Se calibra de manera proporcional la lectura del ADC de acuerdo con la
// la posicion de dos puntos diagonales de la pantalla (799,0) y (0,479)
// cada una con su lectura ADC correspondiente.
alt_touchscreen_calibrate_upper_right (&touch_screen,
                        3946, 3849, // ADC readings
                        799, 0 ); // pixel coords

alt_touchscreen_calibrate_lower_left (&touch_screen,
                        132, 148, // ADC readings
                        0, 479 ); // pixel coords

// Se inicializa y se asigna el hardware de lcd_sgma
// al puntero display, se inicializa el marco de trabajo
// para dibujar en la pantalla LCD, se define el número de frames
```

```
        // la resolución el espacio de color a la que va funcionar la
        // pantalla.
        display = alt_video_display_init( ALT_VIDEO_DISPLAY_SGDMA_NAME,
                                    // Nombre del periférico
                                    ALT_VIDEO_DISPLAY_COLS,
                                    // Ancho en pixeles 800
                                    ALT_VIDEO_DISPLAY_ROWS,
                                    // alto en pixeles 480
                                    ALT_VIDEO_DISPLAY_COLOR_DEPTH,
                                    // profundidad de color
                                    ALT_VIDEO_DISPLAY_USE_HEAP,
                                    // locación del buffer
                                    ALT_VIDEO_DISPLAY_USE_HEAP,
                                    // descripción del buffer
                                    2 );
        // número de frames de video
        while(1)// Se inicia un bucle infinito
        {

                // Se captura la posicion (X,Y) donde la pantalla
                // está siendo oprimida, y captura si la pantalla está siendo
                // oprimida en la variable pen_down
                alt_touchscreen_get_pen (&touch_screen,
                                         &pen_down,
                                         &x,
                                         &y);

                if(pen_down)//si se oprime la pantalla realice
                {
                //Si la pantalla se oprime en una región
                //especifica, dibuje el botón down
                if(x>=0 && x<=122 && y>=0 && y<=43)
                {
                vid_draw_box (0, 0, 122, 43 BLUE_24, DO_FILL,display);
                vid_print_string(33, 14,GRAY_24, cour10_font, display,"btn_down");
                }
                else// de lo contrario dibuje el botón up
                {
                vid_draw_box (0, 0, 122, 43 BLUE_24, DO_FILL,display);
                vid_print_string(33, 14,GRAY_24, cour10_font, display,"btn_down");
                }
                }
```

```
                else// si no se oprime la pantalla, dibuje el
                // boton up
                {
                vid_draw_box (0, 0, 122, 43 GRAY_24, DO_FILL,display);
                vid_print_string(33, 14, RED_24, cour10_font, display,"btn_up");
                }

                // registre el botón dibujado en el espacio de memoria visible en
                // la pantalla LCD
                alt_video_display_register_written_buffer( display );
                while ( alt_video_display_buffer_is_available( display ) != 0 )
                { }
        }
// Se cierra el espacio de memoria asignado a video.
alt_video_display_close( display,
ALT_VIDEO_DISPLAY_USE_HEAP,
ALT_VIDEO_DISPLAY_USE_HEAP );
return 0;
}
```

En el **Codigo 5** se implementaron las librerías gráficas y touch de Terasic para dibujar en la pantalla LCD un botón simple y crear una interacción con el usuario.

FIGURA 35. Botones Up y Down.



El botón está compuesto por dos estados, el estado up es el botón sin oprimir y el estado down es el botón oprimido por el usuario como lo muestra la **FIGURA 35**.

LIBRERIA DE IDENTIFICACIÓN DE EVENTOS UP, DOWN E IDLE DE LA MEMBRANA TOUCH:

Una vez se hace implementacion del driver de la membrana touch para identificar los eventos de pantalla básicos, se ve necesario identificar otros eventos como lo son el evento down, up e idle.

Se decide crear una librería que complemente la dada por Terasic corp debido a que los eventos deben ser identificados apropiadamente para una mejor interacción con el usuario, teniendo en cuenta lo investigado según la norma ANSI/AAMI HE75:2009 que indica que la acción atada a la interacción con la pantalla touch en cada botón dibujado en pantalla solo se debe ejecutar solo si existe un evento up sobre el objeto.

Estos eventos también son utiles para darle a autonomía al software de redibujar los objetos de pantalla solo si es necesario y no constantemente como muestra el ejemplo del código de ejemplo anteriormente explicado. Esto hace que el rendimiento del sistema aumente de manera que el procesador pueda concentrarse en otros procesos mientras no exista interacción con el usuario.

De este algoritmo genero la librería eventos touch que está compuesta por los archivos:

- TOUCH_EVENT.h
- TOUCH_EVENT.c (ver **Codigo 6**)

Codigo 6. Codigo para la captura de eventos touch(up, down e idle).

```
#include "TOUCH_EVENT.h"


static int STATE=0;
int x,y;
static x2,y2;
int pen_is_down;

/*Función event_is_up
* esta función devuelve un valor entere de acuerdo al evento que
```

```
* se presente en la membrana touch, teniendo como base el evento down
* del driver
*
* Implementación:
* int event,pos_x,pos_y;
* event = event_is_up(&touchscreen,&pos_x,&pos_y);
*
* si event = 0, el evento es DOWN.
* si event = 1, el evento es UP.
* si event = 3, el evento es IDLE.
* si event = -1, estado no deseado.
* */
int event_is_up(alt_touchscreen* touchscreen, int* posx, int* posy)
{

        alt_touchscreen_get_pen (touchscreen, &pen_is_down, &x, &y );

        if(pen_is_down){
                STATE=1;
                x2=x;
                y2=y;
                *posx=x2;
                *posy=y2;
                printf("x=%d, y= %d \n",x,y);
                return 0;// STATE DOWN
        }
        else{
                if(STATE==1){
                STATE=0;
                *posx=x2;
                *posy=y2;
                return 1;// Now event is up// STATE UP
                }
        *posx=-1;
        *posy=-1;
        return 3; // STATE IDLE
        }

        return -1;// No deseable
}
```

4.1.1.7 OBJETOS C++ INTERFAZ GRAFICA DE USUARIO

Los objetos necesarios para la GUI del monitor de signos vitales deben tener un aspecto llamativo y amigable para que estos interactúen de acuerdo a los eventos generados por el usuario.

En lenguaje C++ se pueden describir objetos utilizando lo que se llaman clases que son un conjunto de variables(atributos) y funciones(métodos) que se enlazan para formar un elemento que puede reaccionar de múltiples maneras de acuerdo a variables internas o externas a su estructura. Los botones son un ejemplo de ello, si se oprime un botón este modifica en sus registros internos una variable le dice que esta oprimido llamando a s vez un método que redibuje el estado del botón, pocos momentos después se puede observar que el botón se ha dibujado de otra forma, color o tamaño. Un objeto grafico es aquel que reacciona a algún estimulo generado por eventos externos o internos cambiando su apariencia.

Para la GUI del monitor de parámetros fisiológicos, es necesario que en cada objeto que el procesador dibuje en pantalla reaccione y cambie de acuerdo a los eventos generados por la interacción con la membrana touch de la pantalla.

Cuatro objetos necesarios para la GUI son:

- **El Botón:** Este objeto debe interactuar con el usuario y cambiar su forma dependiendo de los eventos generados, en el caso la interfaz tendrá 2 eventos básicos el evento UP y el evento DOWN. De la misma manera el objeto podrá ejecutar otras funciones de acuerdo cada evento, como cambiar alguna variable externa a el como enviar datos por medio del puerto de comunicaciones RS232.
- **El Label:** Este objeto debe contener un texto que cambie de manera automática si así se le solicita.
- **El Picturebox:** Este objeto debe contener una imagen estática en una posicion especifica de la pantalla.
- **El GraphXY:** Este objeto está encargado de graficar en una posicion especifica de la pantalla una gráfica redimensionable en sus dos ejes coordenados de acuerdo a los datos adquiridos en la comunicación con las diferentes tarjetas OEM, para tener visualización de las gráficas de ECG y RESP.

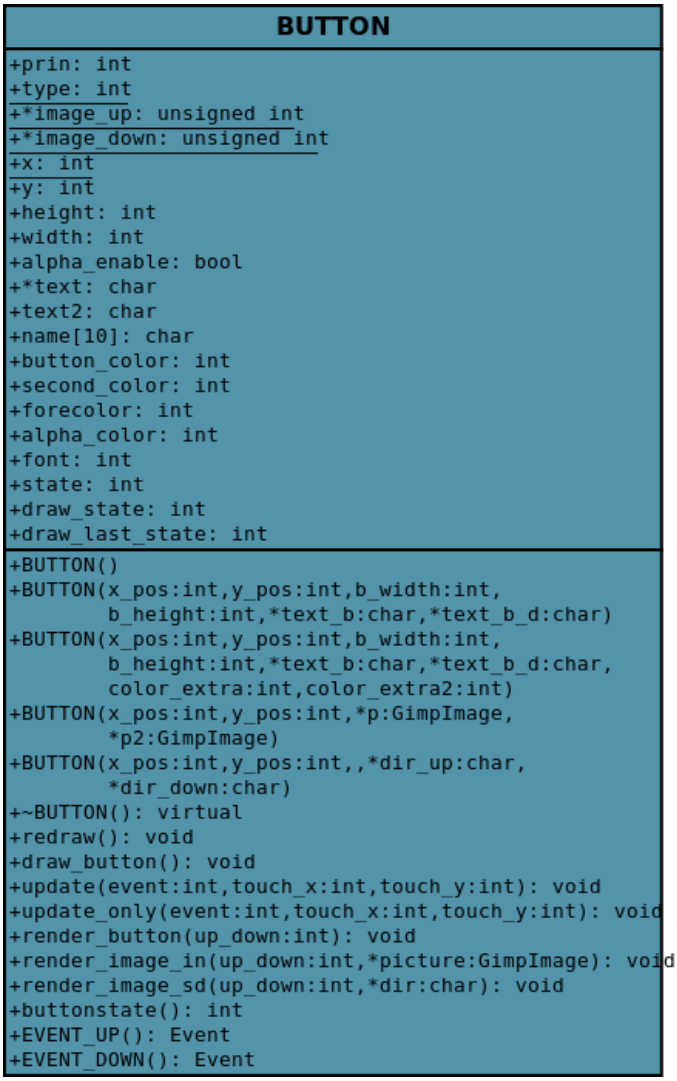
Cada objeto tiene su respectivo constructor y destructor recordando que en lenguaje C++ no existe un recolector de basura es obligación declarar un destructor.

DESCRIPCION DE LOS OBJETOS GENERADOS POR MEDIO DE DIAGRAMAS UML:

El Objeto Botón: el objeto botón está compuesto por atributos y métodos (ver FIGURA 36), con los cuales se puede modificar e interactuar con el objeto, para darle características de acuerdo a cada evento generado por la membrana touch (librerías modificada de terasic). El

Codigo 7 explica en un ejemplo el modo de uso del objeto botón.

FIGURA 36. Diagrama UML del objeto botón, generado en el software libre Día.



Codigo 7. Ejemplo uso clase BUTTON.

```
/* se declara la clase button y se construye el objeto button*/

BUTTON *button1=new BUTTON(0,200,200,100,"Button_up","Button_down");
/*el objeto construido tendra las siguientes características
posicion en x,y = 0,200
tamaño ancho,alto= 200,100
texto up,down = “Button_up“,“Button_down“
*/
```

```
void button1_up()
{
        printf("button1 up event \n");
}

void menu()
{
        int touch_x,touch_y;
        int event;

        /* se ata la función externa la cual se activa con el evento up del botón,     la
funcion puntero apunta a la funcion atada*/
        button1->EVENT_UP=&button1_up;

        while(1)// se inicia la interfaz indefinidamente
        {
        /* se obtiene los eventos de la interfaz touchscreen para enviarlos al
        receptor del objeto button1*/
        event=event_is_up(&touchscreen,&touch_x,&touch_y);
        button1->update(event,touch_x,touch_y);
        }
}
```

El objeto BUTTON, puede ser construido de diferentes maneras dependiendo de la necesidad del programador y de los constructores (ver Codigo 8) que la clase ofrezca en su estructura.

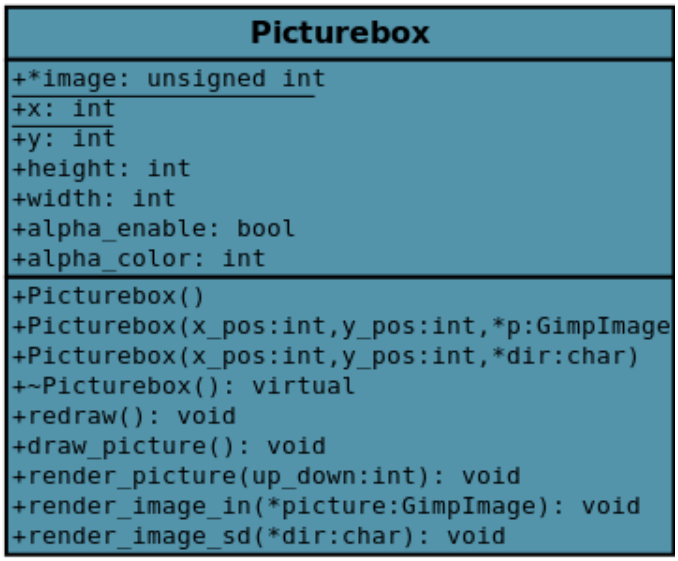
Codigo 8. Constructores de la clase BUTTON.

```
//CONSTRUCTOR ESTANDARD
BUTTON();

// CONSTRUCTOR 1
// Este constructor crea un botón estándar con dos labels diferentes, uno para up
// y el otro para down
BUTTON(int x_pos, int y_pos, int b_width, int b_height, char* text_b,char*
text_b_d);
```

```
// CONSTRUCTOR 2
// Este constructor crea un boton con dos labels diferentes, y con dos colores
// diferentes de fondo
BUTTON(int x_pos, int y_pos, int b_width, int b_height, char* text_b,char*
text_b_d, int color_extra,int color_extra2);

// CONSTRUCTOR 3
// Este constructor crea un boton con dos imagenes de fondo diferente
// traidas desde una estructura gimp
BUTTON(int x_pos, int y_pos,GimpImage* p,GimpImage* p2);

// CONSTRUCTOR 4
// Este constructor crea un boton con dos imagenes de fondo diferente
// traidas desde el directorio de una memoria SD externa
BUTTON(int x_pos, int y_pos, char *dir_up, char *dir_down);
```

- **El Objeto Picturebox:** Este objeto contiene atributos y métodos (ver
- FIGURA **37**) que le brindan características diferentes a la del objeto BUTTON, en este caso el Objeto Picturebox, solo guarda en sus atributos una imagen fija que puede ser cambiada por el usuario utilizando algunos de sus metodos. El
- Codigo 9 explica en un ejemplo el modo de uso del objeto Picturebox.

FIGURA 37. Diagrama UML del Objeto Picturebox, , generado en el software libre Día.



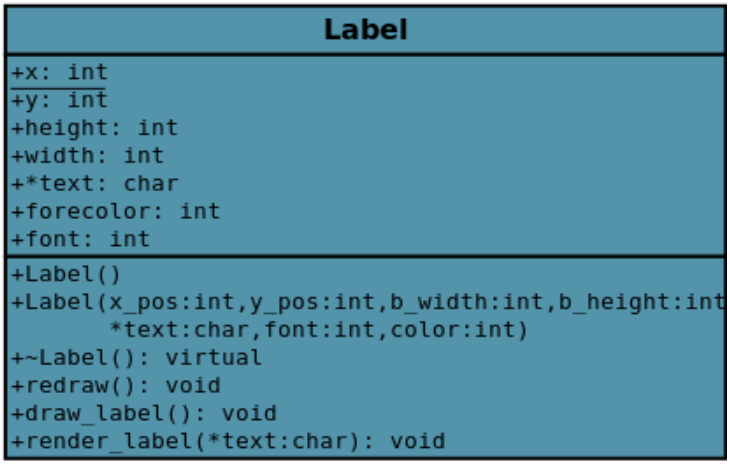
Codigo 9. Ejemplo uso clase BUTTON.

```
/* se declara la clase Picturebox y se construye el objeto Picture1*/

Picturebox *Picture1=new Picturebox(0,200,"sd/imagen.jpg");
/*el objeto construido tendra las siguientes características
posicion en x,y = 0,200
Imagen cargada desde la SD = “sd/imagen.jpg“
*/

void menu()
{
```

```
/* se dibuja la imagen creada*/
        Picture1->draw_picture();
        while(1)/* se inicia la interfaz indefinidamente */
        {
        // se escribe otro codigo
        }
}
```

- **El objeto Label:** Este objeto contiene atributos y objetos que le dan características especiales con respecto a otros objetos, este en él se visualiza texto con características de tamaño, color y fuente. El **Codigo 10** explica en un ejemplo el modo de uso del objeto Label.

FIGURA 38. Diagrama UML del Objeto Label generado en el software libre Día.



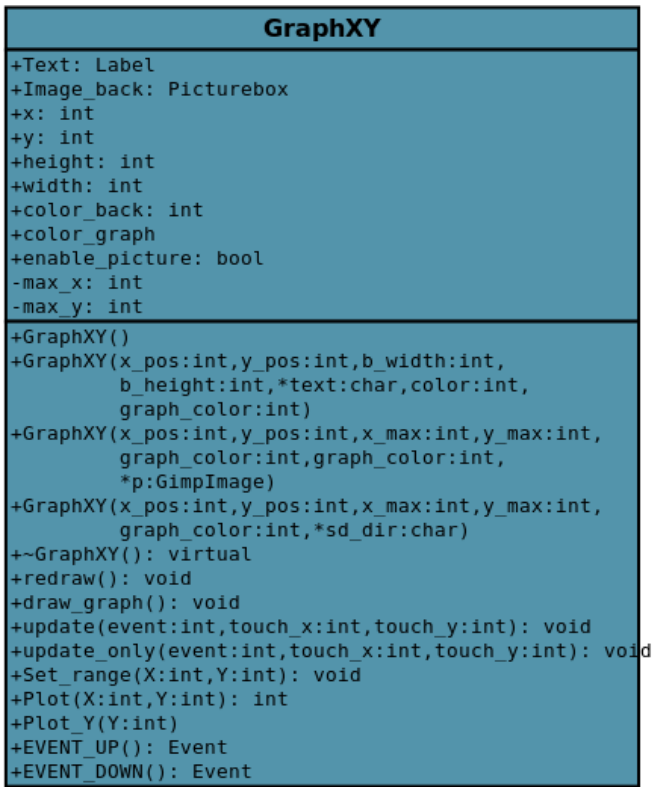
Codigo 10. Ejemplo uso clase Label.

```
/* se declara la clase Label y se construye el objeto Label1*/
Label *Label1=new Label(0,200,100,10,"FCV",coursans_10,0xFF0000);
/*el objeto construido tendra las siguientes características
posicion en x,y = 0,200
tamaño del objeto width,height = 100,10
color de fuente = ROJO(0xFF0000)
```

```
texto del label = “FCV“
*/
void menu()
        {
        /* se dibuja el label creado*/
        Label1->draw_label();
        while(1)/* se inicia la interfaz indefinidamente */
        {
        // se escribe otro codigo
        }
        }
```

- **El objeto GraphXY:** El objeto GraphXY, tiene atributos y metodos que le dan características especiales, este objeto es la composición de 1 objeto label y un objeto Picturebox, además que tiene funciones específicas para dibujar en la pantalla una gráfica de tiempo continuo en los puntos XY. El **Codigo 11** explica en un ejemplo el modo de uso del objeto GraphXY.

FIGURA 39. Diagrama UML del Objeto GraphXY generado en el software libre Día.



Codigo 11. Ejemplo uso clase GraphXY.

```
/* se declara la clase GraphXY y se construye el objeto */
GraphXY *Graph1=new GraphXY(0,200,600,200,"ECG 1",0x000000,0xFF0000);
/*el objeto construido tendra las siguientes características
posicion en x,y = 0,200
tamaño ancho,alto= 600,200
Titulo = “ECG 1“
Color de fondo NEGRO(0x000000)
Color de grafica ROJO(0xFF0000)
```

```
*/
void Graph1_up()
{
        printf("grafica up event \n");
}

void menu()
{
        int touch_x,touch_y,_i;
        int event;
        /* se ata la funcion externa la cual se activa con el evento up del boton, la
        funcion puntero apunta a la funcion atada*/

        Graph1->EVENT_UP=&Graph1_up;
        while(1)/*se inicia la interfaz indefinidamente*/
        {
        /* se obtiene los eventos de la interfaz touchscreen para enviarlos al
        receptor del objeto Graph1*/
        event=event_is_up(&touchscreen,&touch_x,&touch_y);
        Graph1->update(event,touch_x,touch_y);
        /* grafico un diente de sierra indefinidamente */
        for(_i=0;_i=10; _i++){
        /*Agrego un apunto a la gráfica en amplitud*/
        Graph1->Plot_Y(i);
        }
        usleep(1000);/*cada milisegundo grafico*/

        }
}
```

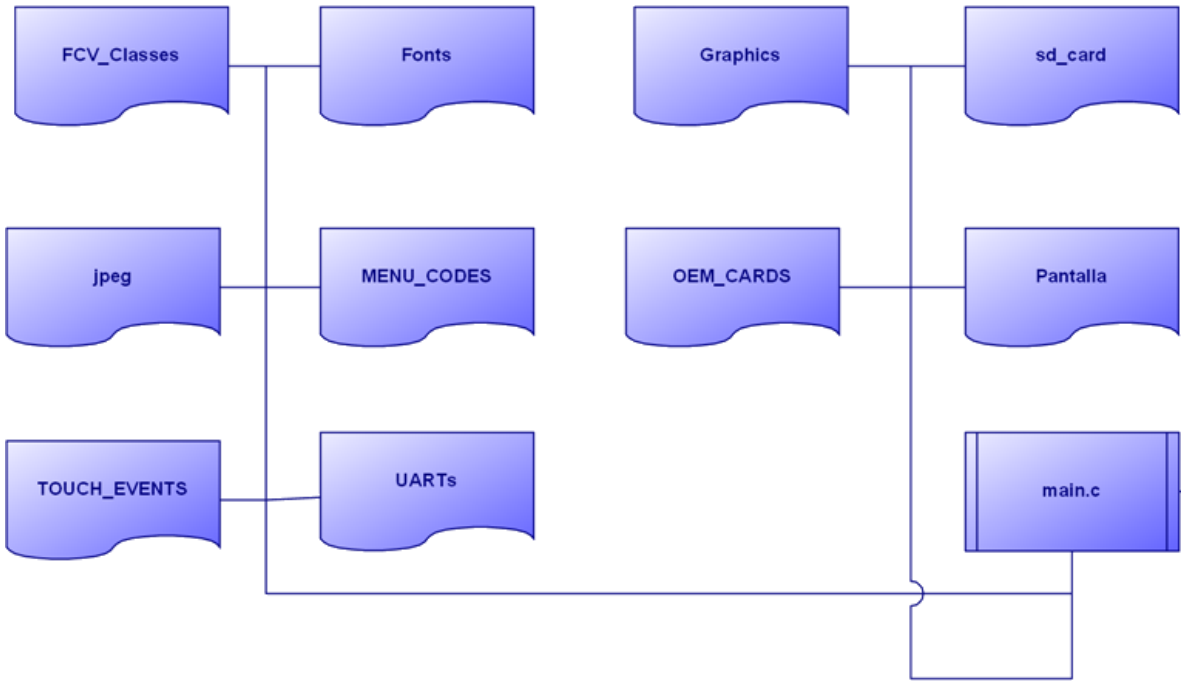
El codigo de los objetos escritos en lenguaje C++ son confidenciales ya que representan un desarrollo intelectual de la FCV. Cada objeto descrito se renderiza utilizando las librerías graficas de Terasic corp.

4.1.1.8 OPTIMIZACION Y ESTRUCTURACION SOFWTARE

La estructuración y optimización del software del sistema se hace con el propósito de facilitar al diseño de GUIs y de optimizar el rendimiento del procesador Nios II a la hora de ejecutar cualquier programa.

La primera etapa en la estructuración del codigo C y C++, es la organización de cada librería, driver o archivos en carpetas de acuerdo a su funcionalidad en el codigo base del programa.

FIGURA 40. Estructuración de librerías.



Los archivos se separaron en carpetas de la siguiente manera (ver **FIGURA 40**):

- FCV_Classes\
- Fonts\
- Graphics\
- sd_card\

- jpeg\
- MENU_CODES\
- OEM_CARDS\
- Pantalla\
- TOUCH_EVENTS\
- UARTs\
- main.c

DESCRIPCION DE CONTENIDO DE CADA CARPETA:

- **FCV_Classes:** Esta carpeta contiene los objetos de la GUI escritos en lenguaje C++, en ella están contenidas todas las clases de los widgets gráficos del sistema.
- **Fonts:** Esta carpeta contiene todas las fuentes o tipos de letras que utiliza el GUI para mostrar datos numéricos o mensajes en pantalla.
- **Graphics:** Esta carpeta contiene todas las librerías de dibujo dadas por Terasic y las hechas por el programador, estas librerías facilitan el dibujo de los objetos c++ en pantalla.
- **Sd_card:** Contiene los drivers del periférico de la memoria SD SPI y contiene los drivers para montar la memoria en formatos FAT16 o FAT32.
- **MENU_CODES:** Esta carpeta contiene los códigos de los MENU de la GUI que están enlazados a los objetos c++ gráficos, esta parte se hace un enlace de funciones C++ con funciones C.
- **OEM_CARDS:** Esta carpeta contiene los drivers de codificación y codificación de datos de cada tarjeta OEM.
- **Pantalla:** Esta carpeta contiene los drivers de control de video y de la membrana touchscreen.
- **TOUCH_EVENTS:** Esta carpeta contiene el codigo de identificación de los eventos touchscreen que suceden cuando el usuario interactúa con la membrana touch, esta librería fue escrita utilizando los drivers que están contenidos en pantalla, para generar los eventos UP, DOWN e IDLE en la interacción con el usuario.
- **UARTs:** Esta carpeta contiene los drivers necesarios para establecer comunicación por el puerto UART del sistema, en él se encuentra el control de interrupciones para el cambio de contexto software cuando se hace adquisición y envió de datos.
- **Main.c:** Este archivo contiene el codigo de inicialización del sistema y enlaza toda la estructura del software para que pueda correr adecuadamente la GUI diseñada.

En la segunda etapa se optimiza el software del sistema, esto se logra revisando a fondo la funcionalidad de cada librería escrita por el programador, en el caso del dibujo de los objetos gráficos para la GUI fue necesario optimizar el codigo de las clases C++ para que estos se dibujen solo si hay un cambio que represente un determinado evento, por ejemplo si el usuario oprime un boton este debe cambiar su forma de acuerdo al evento down teniendo en cuenta redibujar solamente si el estado del boton ha pasado a un estado diferente al down, esto acelera el sistema y permite que se puedan ejecutar otros procesos de mayor prioridad como la decodificación de datos.

Otro factor importante en el proceso de optimización, es aprovechar de manera adecuada la memoria RAM que ofrece el sistema, esto se hace liberando los recursos innecesarios que no se estén utilizando o no se vayan a utilizar por un determinado tiempo, como por ejemplo variables, vectores y objetos. Para darle espacio a otras variables liberando otras se utiliza lo que se conoce en lenguaje C como el manejo de espacios de memoria utilizando funciones como MALLOC(), REMALLOC() y FREE(). En el caso de liberar los objetos C++ se debe declarar lo que se llama un destructor de clase, el cual se encarga de liberar de memoria el espacio que utilice la misma.

Con la estructuración y optimización del software se logra desarrollar y diseñar aplicaciones ordenadas y confiables, que puedan ser interpretadas e implementadas de manera fácil por cualquier otro programador. A la hora de diseñar la GUI para el monitor de parámetros fisiológicos esto permite agilizar los procesos de diseño y asegurar el sistema sea estable y no entre a conflictos por manejo de lugares indeseados de memoria.

4.1.1.9 REGISTRO DE LA INTERRUPCION Y ADQUISICION DE DATOS UART

Las interrupciones permiten cambiar de contexto un proceso consecutivo para cumplir una determinada tarea mientras se pausa lo que el procesador este haciendo, finalizada la interrupción el procesador retoma la tarea anterior esperando para activarse de nuevo.

El procesador Nios II utiliza interrupciones (IRQ), estas pueden ser habilitadas o deshabilitadas por software para que el procesador realice una cierta actividad cada cierto tiempo o cada evento externo o interno, las interrupciones del procesador pueden ser activadas por un Timer interno o por un periférico debido a

un cambio externo dado por flancos de bajada, subida o nivel lógico (Altera, Nios II Software Developer's Handbook NII5V2-10.0, 2010).

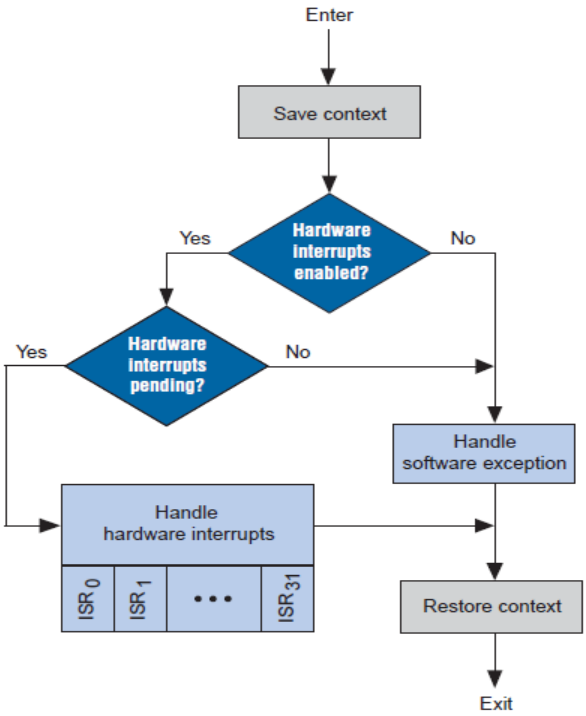
PARA REGISTRAR INTERRUPCIONES DEL PROCESADOR NIOS II SE DEBE:

- Registrar la interrupción y atarla a una ISR(Interrupt Service Routine).
- Escribir la rutina de interrupción ISR.
- Habilitar la IRQ.

Las subrutinas de ISR deben ser lo más cortas posibles para evitar latencias largas que pueden hacer que el procesador funcione de manera inadecuada.

Cada IRQ debe tener una prioridad que ayudara a definir que procesos son más importantes que otros para que el procesador los active de la manera adecuada. Una vez se registra cada IRQ y se escribe el ISR correspondiente, el sistema gestiona cada una por medio de un controlador interno de interrupciones, encargado de cambiar y resumir el contexto del proceso que se esté realizando en el momento (ver **FIGURA 41**).

FIGURA 41. Controlador interno de interrupciones para el manejo de excepciones del sistema.



Debido a que los datos de salida y entrada del puerto UART son asíncronos se hace imposible capturar y enviar todos los datos de manera adecuada sin perder datos. Para evitar la pérdida de datos se debe habilitar la IRQ del UART, se debe inicializar y registrar su ISR que se encargara de almacenar en un buffer los datos recibidos y de enviar de manera ordenada los datos de salida. Es importante guardar los datos de entrada y salida en un buffer para que otra funcion del codigo pueda decodificarlos y codificarlos por medio de algún protocolo de comunicación compatible (ver **Codigo 12**).

Codigo 12. Codigo para registro de la interrupción del puerto UART para enviar y almacenar datos de la manera adecuada.

```
#define RX_BUFFER_SIZE_1 4096
#define TX_BUFFER_SIZE_1 4096

unsigned char tx_buffer_1[TX_BUFFER_SIZE_1];
unsigned char rx_buffer_1[RX_BUFFER_SIZE_1];
unsigned short RxHead_1=0;
unsigned short RxTail_1=0;
unsigned short TxHead_1=0;
unsigned short TxTail_1=0;

//*********************************************************************
//*********************************************************************

// ISR del UART
void IsrUart1(void* context, unsigned int id)
{
        int sr;

        // captura el estado del puerto UART
        sr = IORD_ALTERA_AVALON_UART_STATUS(UART1_BASE);

        // Verifica si hay datos de entrada y los almacena en el buffer de //recepcion
        if(sr & ALTERA_AVALON_UART_STATUS_RRDY_MSK);
        {
        rx_buffer_1[RxHead_1] =
        IORD_ALTERA_AVALON_UART_RXDATA(UART1_BASE);
        IOWR_ALTERA_AVALON_UART_STATUS(UART1_BASE, 0);
        if (++RxHead_1 > (RX_BUFFER_SIZE_1-1)) RxHead_1 = 0;
        }
```

```
        //Verifica si hay datos en cola de salida y los envía por el puerto
        if(sr & ALTERA_AVALON_UART_STATUS_TRDY_MSK)
        {
        if(IORD_ALTERA_AVALON_UART_CONTROL(UART1_BASE) &
        ALTERA_AVALON_UART_CONTROL_TRDY_MSK);
        {
        if (TxTail_1 != TxHead_1)
        {
        IOWR_ALTERA_AVALON_UART_TXDATA(UART1_BASE,
        tx_buffer_1[TxTail_1]);
        if (++TxTail_1 > (TX_BUFFER_SIZE_1 -1)) TxTail_1 = 0;
        }
        else {IOWR_ALTERA_AVALON_UART_CONTROL(UART1_BASE,
        ALTERA_AVALON_UART_CONTROL_RRDY_MSK);
        }
        }
        }
}
//*********************************************************************
//*********************************************************************
// Funcion para conocer si el buffer esta vacío
unsigned char EmptyUart1()
{
        if(RxHead_1 == RxTail_1) return 1;
        return 0;
}
//*********************************************************************
//*********************************************************************

// Funcion para obtener los datos del buffer de entrada
unsigned char GetUart1(void)
{
        unsigned char rxChar;
        rxChar=rx_buffer_1[RxTail_1];
        if (++RxTail_1 > (RX_BUFFER_SIZE_1-1)) RxTail_1=0;
        return rxChar;
}

//*********************************************************************
//*********************************************************************

//Funcion para envió de datos por el puerto UART1
```

```
unsigned char PutUart1(unsigned char in_char)
{
        unsigned short size;
        unsigned int z;
        //se verifica el estado del puerto UART
        z = IORD_ALTERA_AVALON_UART_STATUS(UART1_BASE) &
        ALTERA_AVALON_UART_STATUS_TRDY_MSK;
        // se asegura que no exista datos en la cola de salida para
        //envier el caracter in_char
        if ((TxHead_1==TxTail_1) && z) {
        IOWR_ALTERA_AVALON_UART_TXDATA(UART1_BASE, in_char);
        }
        else
        {
        // Si la cola no estaba vacía se almacenan los datos en un
        //buffer de salida para enviarlos posteriormente
        if (TxHead_1 >= TxTail_1) size = TxHead_1 - TxTail_1;
        else size = ((TX_BUFFER_SIZE_1-1) - TxTail_1) + TxHead_1;
        if (size > (TX_BUFFER_SIZE_1 - 3)) return (-1);
        tx_buffer_1[TxHead_1] = in_char;
        if (++TxHead_1 > (TX_BUFFER_SIZE_1-1)) TxHead_1 = 0;
        z = IORD_ALTERA_AVALON_UART_CONTROL(UART1_BASE) |
        ALTERA_AVALON_UART_CONTROL_TRDY_MSK;
        IOWR_ALTERA_AVALON_UART_CONTROL(UART1_BASE, z);
        }
return(1);
}

//**********************************************************************
//**********************************************************************

int main()
{
// Inicializar el Puerto UART1
INIT_UART1();
// Registrar la ISR
alt_irq_register(UART1_IRQ,&context_uart1,IsrUart1 );
// Habilitar la IRQ
alt_irq_enable (UART1_IRQ);

        while (running)
        {
```

```
        // se imprime por el puerto UART la palabra HOLA
        PutUart1("H");
        PutUart1("O");
        PutUart1("L");
        PutUart1("A");

        while(!EmptyUart1())//Mientras que el Buffer de entrada este lleno
        {
        // se imprime por el puerto JTAG los datos de llegada del puerto
        //UART
        printf("DATA IN: %c",GetUart1());
        //EN ESTE ESPACIO DE CODIGO SE PUEDE PONER EL CODIGO DE
        //DECODIFICACION DE DATOS PARA CADA TARJETA OEM QUE
        CONFORMA LA GUI
        }
}
}
```

En el **Codigo 12** se implementó para registrar la interrupción del puerto UART1 descrito en el sistema por medio del SOPC Builder para el envió y recepcion de datos sin perder información.

4.1.1.10 GUI PARA LA VISUALIZACION DE PARAMETRO FISIOLOGICOS

Dado el resultado de las importaciones, solo llegaron las tarjetas SuntechMed y Mindray las cuales ofrecen los parámetros fisiológicos de ECG, TEMP, RESP y NIBP, por lo cual el parámetro SpO2 no entra hacer parte de ninguna interfaz programada hasta el momento.

Debido a que la tarjeta de desarrollo NEEK (ver FIGURA 2) solo dispone de un puerto RS-232 y no dispone de GPIOs (General Purpose Input/Ouput), además teniendo en conocimiento que para hacer la adquisición de datos se necesitarían 2 puertos Rs232 debido a que se utilizan 2 tarjetas OEM, la UEN Bioingeniería decidió hacer 2 GUI por aparte, una específica para la tarjeta Mindray y otra específica para la tarjeta SuntechMed.

4.1.1.11 INTERFAZ GRAFICA DE USUARIO MINDRAY

Utilizando las librerías de adquisición de datos de la tarjeta mindray codificados en el capítulo 4.1.1.2 IMPLEMENTACION DE CADA UNA DE LAS TARJETAS OEM DE SIGNOS VITALES, el sistema hardware definido y estructurado el HDL del capítulo 4.1.1.3 DESCRIPCION HDL DEL SISTEMA PARA LA GUI, UTILIZANDO EL SOPC BUILDER, las librerías graficas de Terasic corp**,** se procede a diseñar y programar una interfaz gráfica de usuario amigable, donde se visualicen los parámetros de ECG, RESP y TEMP. Cada parámetro debe tener un menu que configure los datos arrojados de la tarjeta OEM, en el caso de ECG y REP, se programa un menu en donde se puedan configurar opciones de ganancia, filtrado, selección de canal y de derivas de cada una de las señales visualizadas en pantalla.

Se procede a programar la interfaz y a crear los algoritmos de interacción con el usuario, teniendo en cuenta las funcionalidades de la tarjeta. Después de programada la interfaz se hace la verificación de interacción y funcionalidad, para entrar en la etapa de corrección y adaptación de la interfaz para generar una GUI eficiente con todas las funcionalidades que requiere la interacción con la tarjeta Mindray.

A medida de que se avanza en el diseño de la interfaz gráfica de usuario de la tarjeta mindray, se generan mejoras al sistema hardware, al driver de adquisición de datos y a los objetos C++ definidos en los capítulos anteriores, lo que trae como resultado 3 versiones de GUI diseñadas para visualizar los parámetros de ECG, RESP y TEMP. Esto sucede debido a que esta tarjeta facilita la entrega 3 parámetros fisiológicos, cada uno con configuraciones diferentes definidas en el manual del desarrollador de la tarjeta Mindray.

DESCRIPCION Y ANALISIS DE RESULTADOS DE LAS VERSIONES MINDRAY:

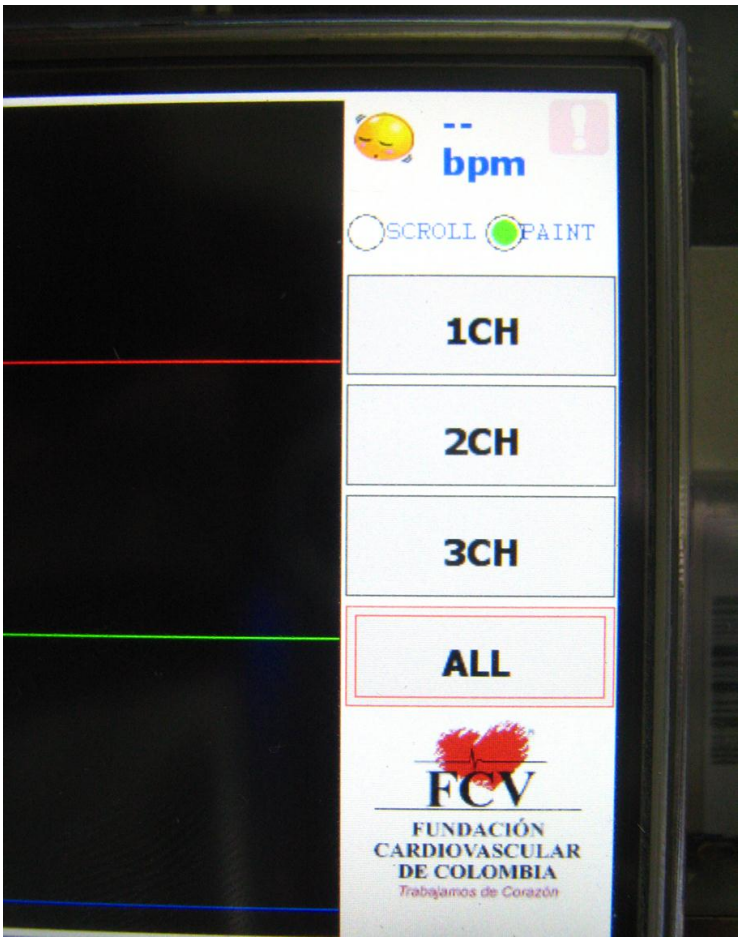
- **GUI Mindray Versión 1:** Esta versión es una de las mejores interfaces generadas, ya que su funcionalidad es rápida, llamativa y sencilla, en ella se observan algunos de los parámetros fisiológicos arrojados por la tarjeta mindray como lo son ECG y RESP. La visualización de datos es rápida y no tiene retraso alguno, sus botones son amigables, tiene dos opciones de visualización de graficas el tipo “overpaint“ que sobrescribe en tiempo a la gráfica cada vez que está completa un tiempo de 2.5 segundos(250 pixeles por segundo) y el tipo “scroll“ que corre los datos en el tiempo a la izquierda para darle paso a los nuevos datos que se adquieren por medio del driver de adquisición de datos.

Esta versión (ver

FIGURA 42) utiliza los botones para seleccionar el número de graficas que se desea visualizar en pantalla, en este caso se puede escoger de 1 a 3 canales de visualización, donde:

- 1CH: Se visualiza el canal 1 de ECG.
- 2CH: Se visualiza los dos canales de ECG.
- 3CH: Se visualiza los dos canales de ECG y 1 canal de Respiración.

FIGURA 42. Interfaz Mindray Version 1.

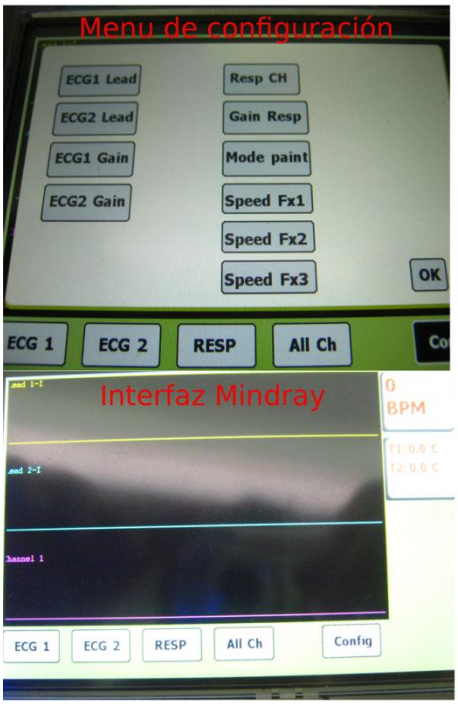


En esta versión se puede observar el ritmo cardiaco HR y 2 botones con los que se selecciona el modo de dibujo con el que se grafica(el modo “Overpaint“ y “Scroll“).

Una de las desventajas de esta interfaz, es la falta de optimización de su codigo software en cuanto a sus objetos visuales(botones), en este momento no se contaba un codigo que definiera un objeto C++ por medio de clases, en vez de eso, los objetos se improvisaban haciendo implementacion de estructuras y funciones en ANSI-C. Lo que hace que crear un solo objeto fuera demasiado complejo, largo e ineficiente.

- **GUI Mindray Versión 2:** Siendo la segunda versión (ver **FIGURA 43**) para la visualización de parámetros de la tarjeta mindray, esta presenta en su estructura muchas más opciones de configuración y datos de visualización en comparación con la primera versión, como:
    - 2 Canales de temperature.
    - 2 canales de ECG.
    - Ritmo cardiaco HR.
    - 2 canales de Respiración seleccionables.
    - 2 Modos de visualización de graficas scroll y overpaint.
    - Menu de configuración de los parámetros ECG y Respiración
    - Menu de selección de derivas.

FIGURA 43. Interfaz Mindray Version 2.

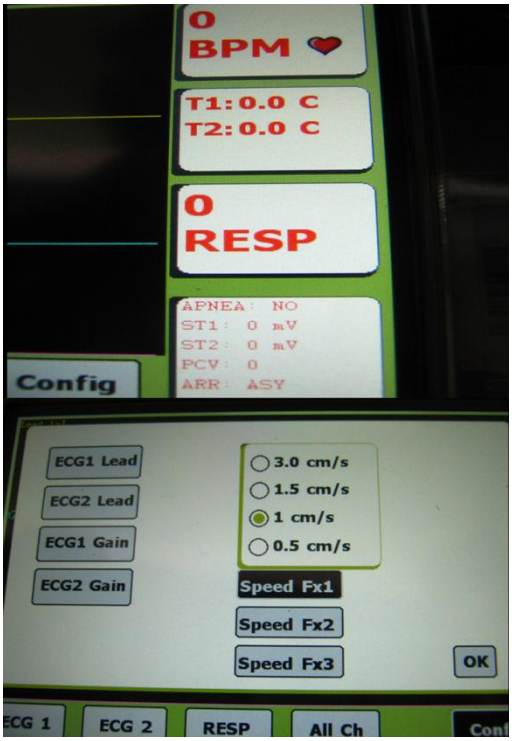


Los menús de configuración ofrecen opciones para configurar las amplitudes, velocidades, selección de canales y selección de derivas. Una ventaja de esta

nueva GUI, es que su codigo está mejor estructurado y sus menús son más animados en la interacción con el usuario, pero aún sigue siendo una desventaja el uso de estructuras en ANSI-C para representar objetos lo que hace que el codigo sea mucho más extenso y difícil de modificar para agregar y quitar nuevos elementos.

- **GUI Mindray Versión 3:** Esta última versión (ver **FIGURA 44**) de la interfaz mindray, es una versión modificada de la versión 2, por lo que la estructura del lenguaje no cambia significativamente. Se hace una optimización en cuanto a la estructura del codigo C y se agregan más opciones en la configuración de parámetros y visualización de los mismos.

FIGURA 44. Modificaciones de la versión 2 de la GUI mindray**.**



Esta nueva interfaz contiene:

- 2 Canales de temperatura.
- 2 canales de ECG.
- Ritmo cardiaco HR.
- Ritmo respiratorio.
- Segmento ST ECG.
- Arritmias Cardiacas.
- Alarma de apnea.

- 2 canales de Respiración seleccionables.
- 2 Modos de visualización de graficas scroll y overpaint.
- Menu de configuración de los parámetros ECG y Respiración
- Menu de selección de derivas.

Si se compara el número de opciones y parámetros que ofrece esta modificación con la versión 2, se puede observar que esta versión contiene en su interfaz más información en su estructura, lo que convierte esta interfaz gráfica de usuario en la versión definitiva de la tarjeta mindray.

CONCLUSIONES DE LA GUI MINDRAY:

La interfaz GUI construida para la visualización de parámetros fisiológicos de la tarjeta mindray, cumple con todas las expectativas pedidas por la empresa, ya que demuestra que el desarrollar y diseñar interfaces graficas de usuario basadas en sistemas hardware sobre FPGAs, es ideal para seguir trabajando en otros proyectos que necesiten sistemas embebidos dedicados que requieran de una interacción con el usuario por medio de pantallas LCD touchscreen, centrados sobre todo en los monitores de signos vitales que actualmente se están diseñando en la UEN Bioingeniería.

El tiempo transcurrido para generar la GUI mindray, trajo decisiones que ayudaron al mejoramiento continuo del sistema hardware y software, en cuanto al hardware se ve necesario la adquisición de datos por medio de interrupciones hardware que por medio del puerto UART, lo que trajo como consecuencia el estudio de las interrupciones en el sistema hardware Nios II para controlar el cambio de contextos por medio de software como se menciona en el capítulo 4.1.1.9 REGISTRO DE LA INTERRUPCION Y ADQUISICION DE DATOS UART; Otro factor de mejoramiento teniendo en cuenta el proceso GUI mindray es la optimización de la estructura software, el reordenamiento de librerías y el diseño de nuevos objetos utilizados para construir menús interactivos lo que hace que los estos pasen de ser simples estructuras ANSI-C a clases complejas en el lenguaje orientado a objetos C++ como lo dice el capítulo 4.1.1.7 OBJETOS C++ INTERFAZ GRAFICA DE USUARIO.

Otra modificación necesaria en el hardware se hace agregando el periférico de lectura de una memoria SD externa por medio del SOPC builder para la lectura de imagenes y de archivos desde dispositivos externos con el propósito no ocupar espacios innecesarios en la memoria flash del sistema, debido a que todas las imagenes utilizadas para la interfaz mindray se agregaron a la FLASH de la tarjeta

NEEK por medio de la interfaz JTAG, lo que hace que se desperdicien recursos de manera inadecuada y se reduzca el tiempo de vida de la memoria no volátil necesaria para almacenar la descripción hardware y el software del sistema.

El proceso de diseño de objetos C++ del capítulo 4.1.1.7 OBJETOS C++ INTERFAZ GRAFICA DE USUARIO, la estructuración y optimización del software del capítulo 4.1.1.8 OPTIMIZACION Y ESTRUCTURACION SOFWTARE y la descripción Hardware del sistema del capítulo 4.1.1.3 DESCRIPCION HDL DEL SISTEMA PARA LA GUI, UTILIZANDO EL SOPC BUILDER son resultados del proceso continuo de mejoramiento del diseño de la GUI de mindray.

4.1.1.12 INTERFAZ GRAFICA DE USUARIO SUNTECHMED

Las modificaciones de mejoramiento generadas a partir de la construcción de la GUI Mindray (ver Capitulo 4.1.1.11 INTERFAZ GRAFICA DE USUARIO MINDRAY) hacen que el diseño de la Interfaz gráfica de usuario de SuntechMed sea mucho más compleja, agradable a la vista y eficiente. Esta cuenta con la visualización del parámetro fisiológico NIBP (Presión no invasiva), donde se puede observar la Sístole, Diástole, Presión Media y Ritmo Cardiaco. Además la interfaz cuenta con opciones para la configuración de las lecturas hechas por la tarjeta OEM tales como tipo de paciente Pediátrico, Neonato o Adulto, contar con un menu de visualización de errores y control de la toma de presión para iniciar una medición o cancelarla.

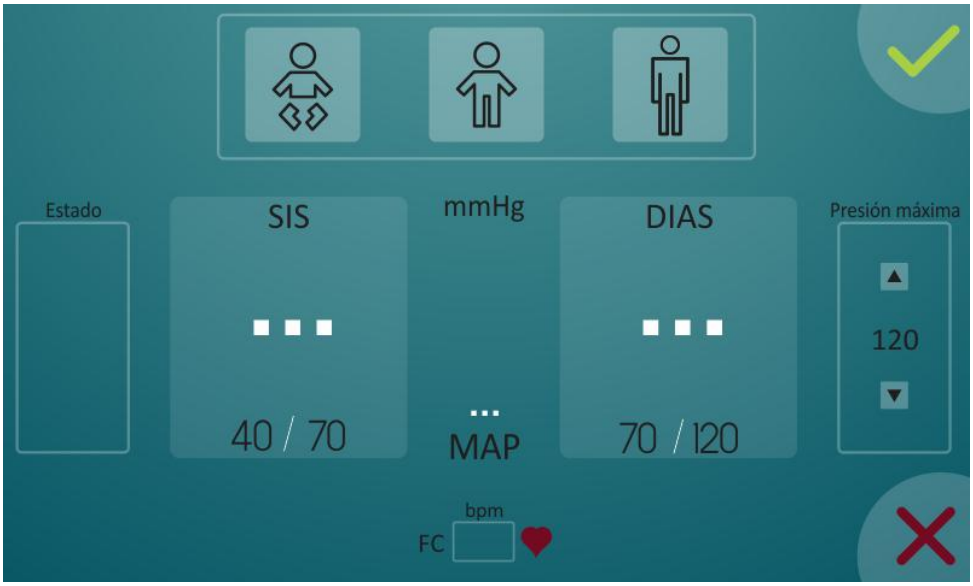
Esta GUI a diferencia de la Mindray está diseñada por el grupo de diseño industrial de la UEN Bioingeniería, lo que brinda una apariencia mucho más amigable al usuario y presenta más complejidad en su estructura, lo que apoya el proceso de optimización y estructuración del software que se muestra en el capítulo 4.1.1.8 OPTIMIZACION Y ESTRUCTURACION SOFWTARE.

Utilizando las librerías de adquisición de datos de la tarjeta suntechmed codificadas en el capítulo 4.1.1.2 IMPLEMENTACION DE CADA UNA DE LAS TARJETAS OEM DE SIGNOS VITALES, el sistema hardware definido y estructurado del capítulo 4.1.1.3 DESCRIPCION HDL DEL SISTEMA PARA LA GUI, UTILIZANDO EL SOPC BUILDER, las librerías graficas de Terasic corp**, un software estructurado y optimizado con librerías de objetos C++,** se procede a programar una GUI que aparece en la **FIGURA 45**.

Esta GUI cuenta con 7 botones de interacción cada uno con una funcionalidad diferente, en la parte superior de esta se encuentran 3 botones de selección para escoger el tipo de paciente al que se le hace la medición de NIBP, en la parte

derecha tiene 2 botones que ayudan al usuario a variar la presión máxima de referencia con la que se le indicara a la tarjeta los datos de configuración que debe tener en cuenta para empezar un toma de presión no invasiva, además en sus esquinas inferiores y superiores derechas se encuentran los botones “Start“ y “Stop“ con los cuales se ordena a la tarjeta empezar o cancelar la medición del parámetro NIBP, en la parte izquierda se observa una caja de texto que indica si la toma de datos fue exitosa, o existieron errores en el proceso.

FIGURA 45. GUI Suntech, Diseñada por los diseñadores industriales y programados por el desarrollador de software.



Esta interfaz también cuenta con espacios definidos para visualizar las mediciones arrojadas por la tarjeta OEM como lo son:

- Sístole.
- Diastole.
- Presión Arterial Media (MAP).
- Ritmo Cardiaco HR.
- Mensaje de error (Estado).

Los diagramas de flujo que se pueden visualizar en la **FIGURA 46**, **FIGURA 47**, **FIGURA 48** y **FIGURA 49**, representan el algoritmo programado para generar la interfaz gráfica de usuario.
El algoritmo que representa el diagrama de flujo fue el siguiente:

- Inicializar por medio de funciones software el Hardware del sistema.

- Dibujar la imagen de fondo de la pantalla utilizando codigo ANSI-C, debido a que esta permanecerá siempre estático.
- Inicializar o renderizar el menu SuntechMed por medio de los objetos C++, utilizando imagenes extraídas desde una memoria SD externa.
- Empezar la supervisión y adquisición de datos de la tarjeta Suntech, teniendo en cuenta los cambios generados por el usuario en la GUI y sus objetos gráficos C++, para registrar todo en la pantalla LCD.

FIGURA 46. Estructura del sistema de la GUI de SuntechMed.

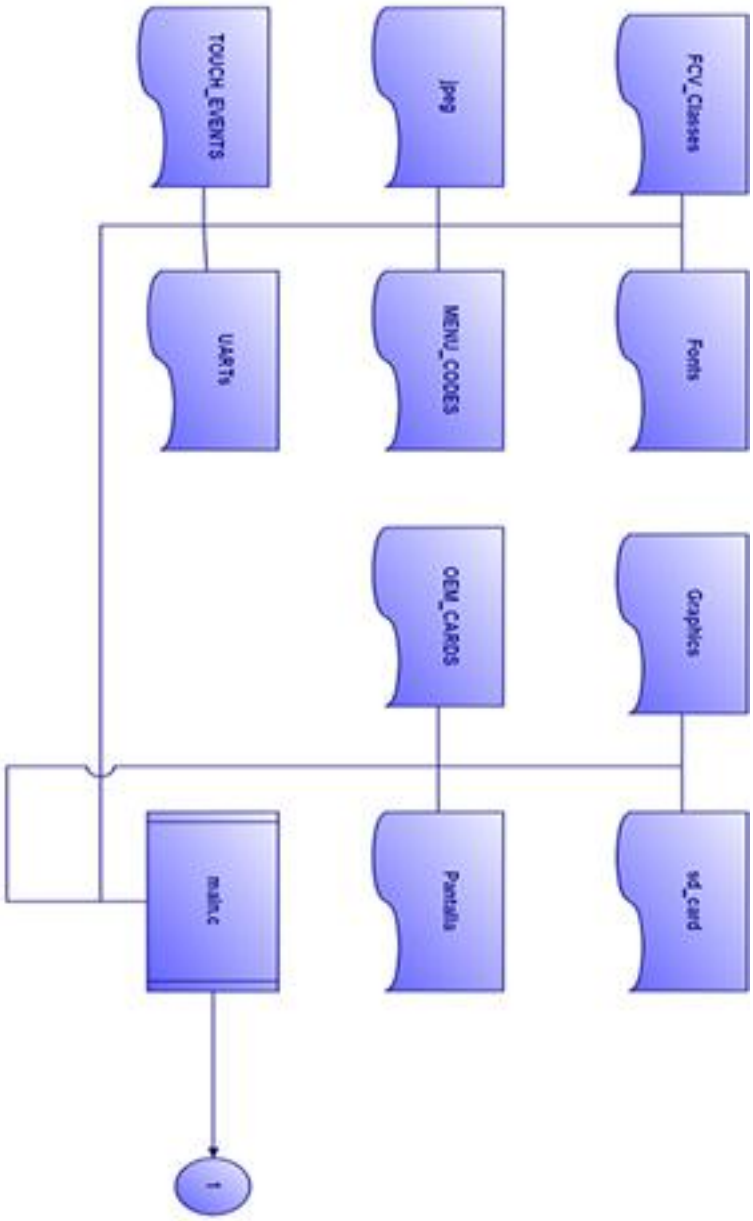


FIGURA 47. Enlazador de programa, Main.c, encargado de inicializar el Hardware y encargado de hacer los enlaces entre las diferentes funciones y procedimientos del programa.

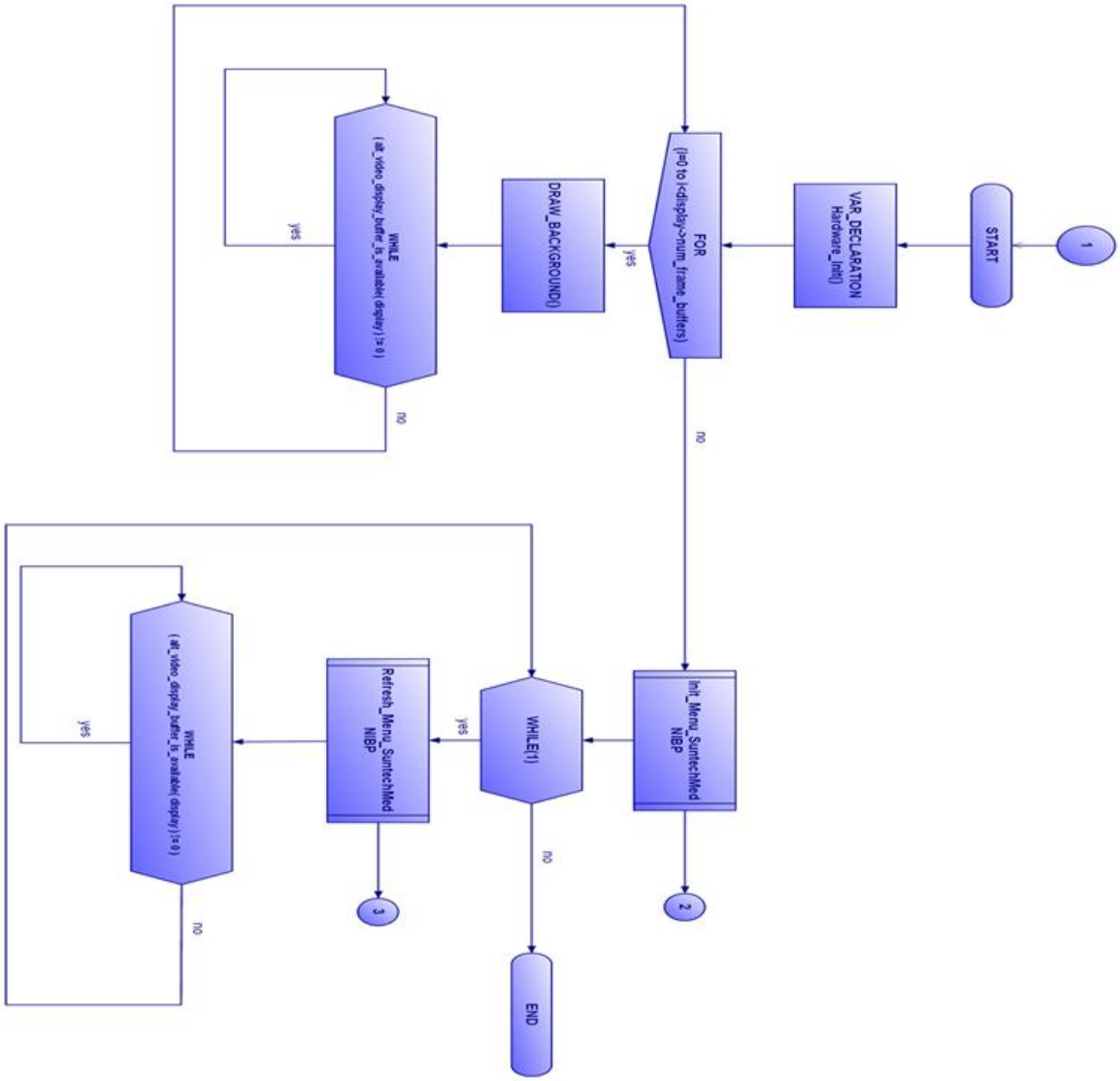


FIGURA 48. Procedimiento de inicialización del menu SuntechMed NIBP.

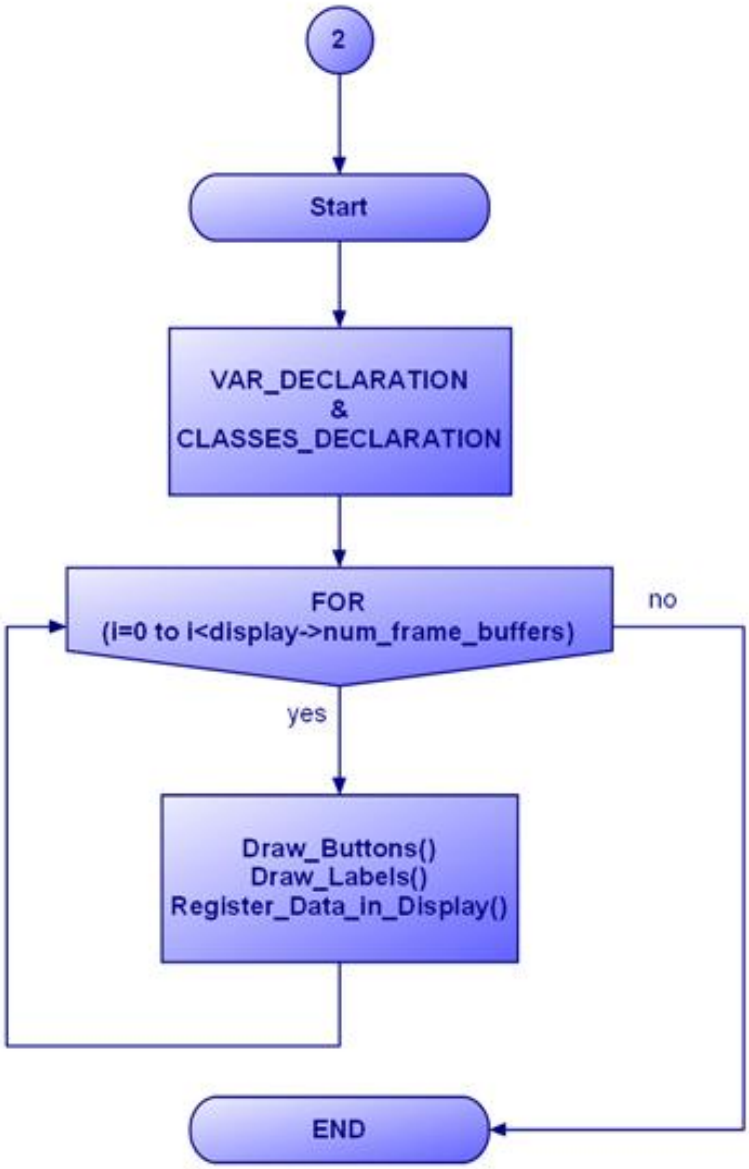
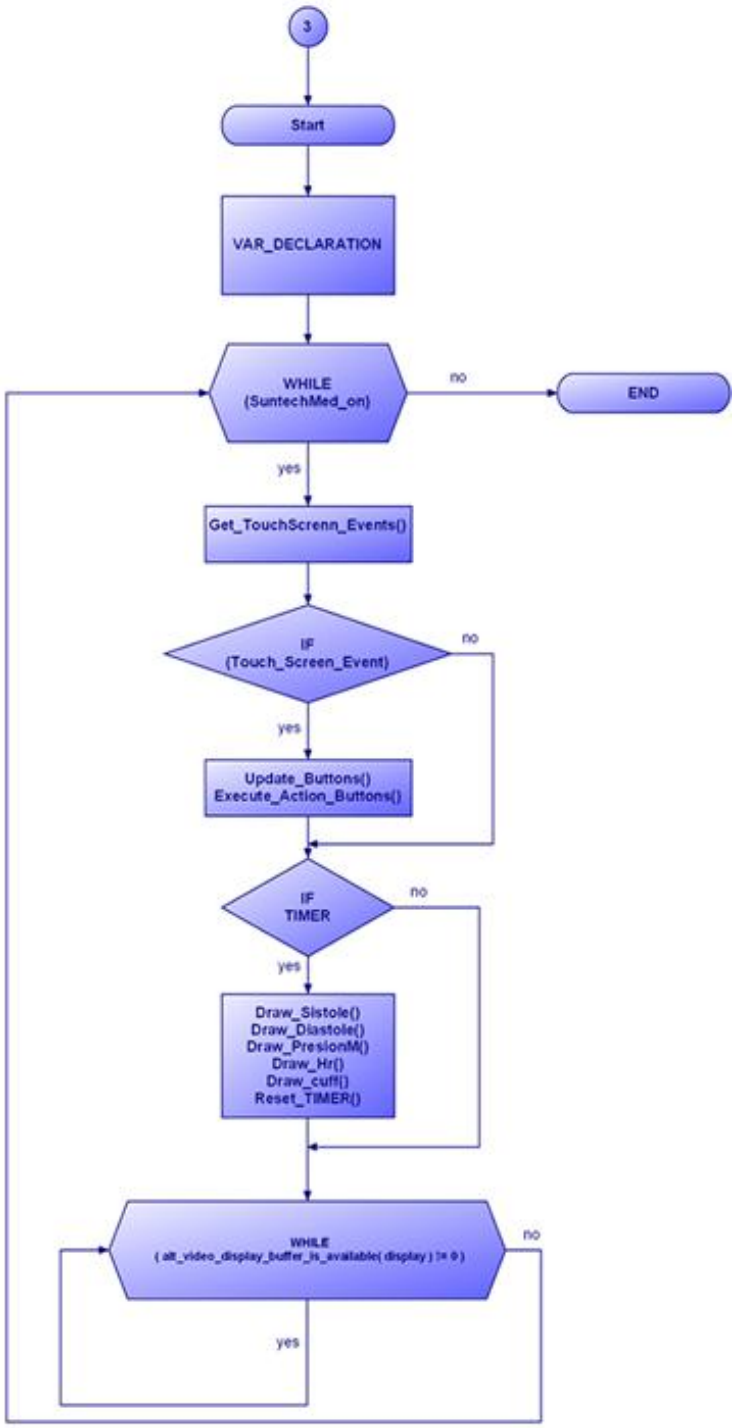


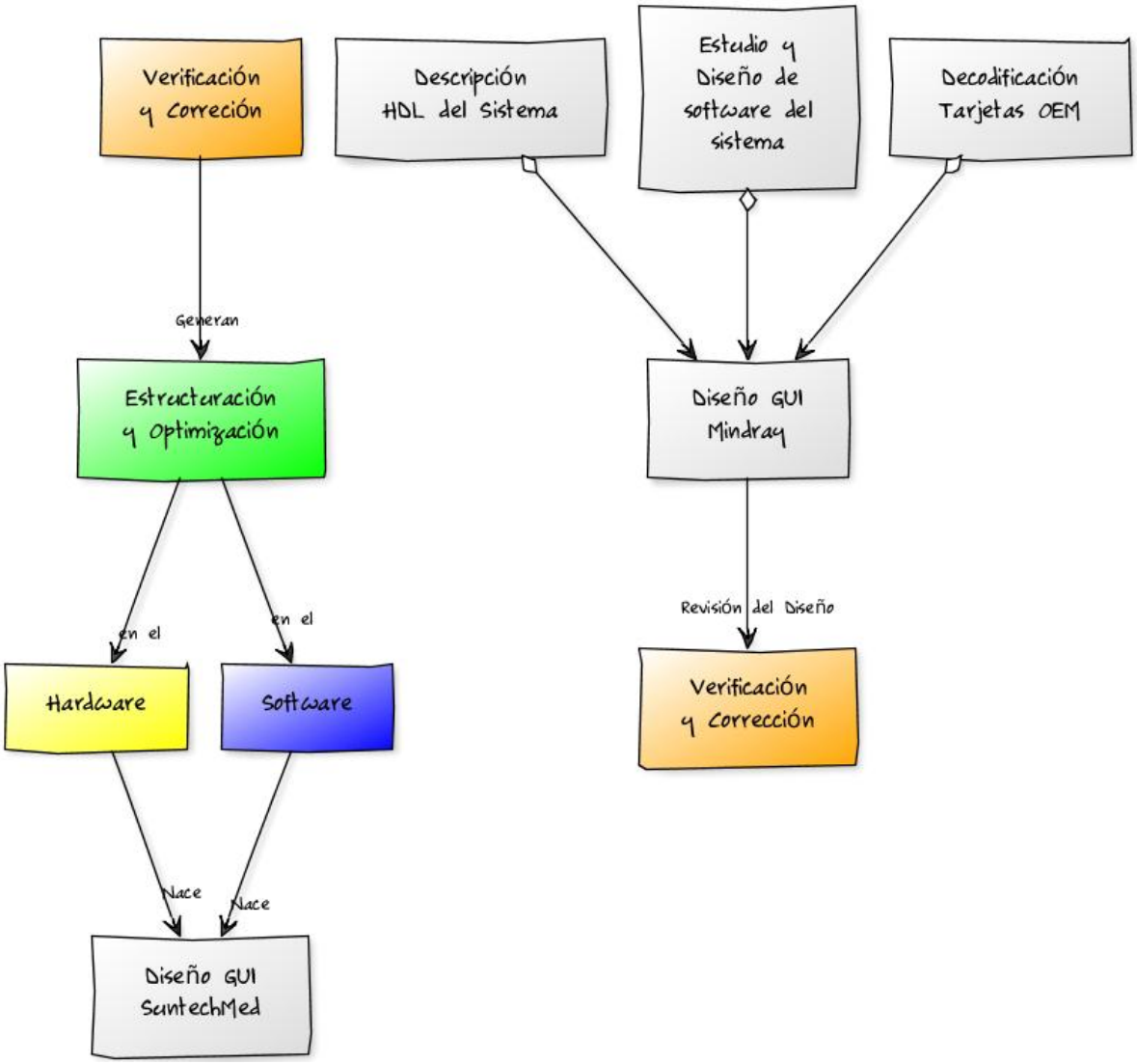
FIGURA 49. Ejecución del proceso menu SuntechMed, encargado de controlar y verificar los cambios en la GUI de acuerdo a los eventos Touch.



CONCLUSIONES DE LA GUI SUNTECHMED:

La interfaz gráfica de usuario SuntechMed representa un sistema de medición para la presión no invasiva y el trayecto de 4 meses (ver **FIGURA 50**) de trabajo continuo de investigación con una previa experiencia en el área de los sistemas embebidos basados en FPGAs. Esta interfaz es la base de futuros productos que representan para la UEN Bioingeniería un avance significativo que les puede permitir abrirse a mercados abiertos y competitivos en el área de los dispositivos médicos dedicados y embebidos.

FIGURA 50. Diagrama del trayecto investigativo para la construcción de Interfaces graficas de usuario utilizando sistemas embebidos basados en FPGAs- Diagrama hecho con la herramienta http://yuml.me/



5. APORTES AL CONOCIMIENTO

En la FCV, se han hecho monitores de signos vitales basados en PC convencionales, utilizando como herramienta de programación Labview para hacer las interfaces gráficas de usuario que hoy en día se están usando en varias clínicas a nivel nacional, esto ha traído varios problemas, uno de ellos es que el hardware utilizado está dedicado día y noche a trabajo continuo y su alto consumo de potencia hace que los elementos internos de boards, procesadores y ram se calienten haciendo corto el tiempo de vida de un equipo de esta clase, generando problemas en el cambio de repuestos. Otro problema con estos equipos es que las interfaces gráficas hechas en Labview se bloquean a consecuencia del hardware y por ser un software muy pesado que consume muchos recursos del sistema.

La FCV vio necesario cambiar el hardware y el software en sus monitores de signos vitales, en este momento con el uso de sistemas embebidos basados en FPGAs(NIOS II) se ha visto que se puede bajar el consumo de potencias en sus monitores y se puede desarrollar un producto dedicado, el cual tendría un tiempo de vida mayor que los actuales; su interfaz gráfica también podrá ser confiable y funcional evitando bloqueos del sistema; además brinda la posibilidad de cambiar rápidamente su hardware interno y software economizando recursos a la hora de hacer modificaciones.

Los ingenieros de la FCV, nunca habían utilizado FPGAs en ninguno de sus proyectos, conocían conceptos generales pero no sabían programarlas y hacer descripción de hardware por medio de Verilog HDL, en este momento se está haciendo capacitación a estos ingenieros para que se familiaricen con esta tecnología. Actualmente se está haciendo un gran aporte al conocimiento de la empresa, ya que el uso de esta tecnología brinda a sus productos y empleados:

- Avance en el conocimiento de nuevas tecnologías.
- Mejor posicionamiento en el mercado por el uso de tecnologías de punta.
- Abre la puerta a la fabricación de nuevos productos gracias al uso de esta tecnología.

Gracias a esta capacitación los Ingenieros pudieron diseñar con una CPDL un hardware acorde a las necesidades del diseño de una incubadora, ya que no encontraban en el mercado un chip que se adaptara a una función requerida para el diseño y de la manera que se tenía se gastaba mucho espacio en el PCB. Con la CPLD hicieron reducción en el diseño del PCB y describieron un hardware con la función específica que necesitaban.

También se hicieron nuevos aportes de conocimiento de la empresa gracias a que se contactaron nuevos distribuidores de tarjetas OEM para la medición de parámetros fisiológicos, generando de esta manera alianzas estratégicas, permitiendo a la FCV mejorar su posición en el mercado y tener un mayor grado de competencia.

Por medio del estudio de los protocolos de comunicación y las interfaces de cada una de las tarjetas OEM, se hace un aporte al conocimiento con la documentación de codificación y decodificación de datos de cada tarjeta de medición de parámetros fisiológicos, para que en futuros diseños los ingenieros de Bioingeniería puedan avanzar de manera rápida, para que no se gasten tiempos innecesarios en el proceso de interpretación de datos de las tarjetas OEM Mindray y SuntechMed.

Gracias al uso de procesadores embebidos(NIOS II) basados en FPGAs, se ha

Podido generar interfaces gráfica de usuario amigables y de fácil manejo, creando de esta manera librerías C/C++ propias de la FCV UEN Bioingeniería las cuales puedan ser utilizadas y mejoradas en otros proyectos. También se ha generado un aporte de conocimiento en arquitectura de computadores a la FCV UEN Bioingeniería, ya que por medio de la herramienta SOPC-Builder se pueden generar sistemas hardware rápidos, flexibles y modificables, teniendo en cuenta características en el diseño de arquitecturas computacionales confiables, para su correcta implementacion en productos que requieran un alto grado de confiabilidad y sean de alto riesgo.

GLOSARIO

AAMI: American National Standard, normas que rigen el diseño y producción de dispositivos médicos.
ALTERA: Marca de FPGAs.
ASIC: Dispositivo hecho de silicio o CHIP.
ANSI-C: Lenguaje de programación en C estándar.
C++: Lenguaje orientado a objetos.
CERN: European Organization for Nuclear Research, acelerador de partículas.
CLASE: Conjunto de atributos y metodos que definen un objeto en C++.
CLB: Bloque lógico configurable.
CPLD: Complex programmable logic device, elemento lógico programable complejo.
CPU: Unidad central de procesamiento.
DMA: Direct Access Memory.
ECG: Electrocardiografía.
FIFO: First Input First Output.
FPGA: Field Program Gates Arrays, arreglo de compuertas lógico programable.
GPIO: Propósitos generales, Input/Output.
GUI: Interfaz gráfica de usuario
HDL: Descripción hardware.
IP CORE: Modulo de descripción hardware con propiedad intelectual.
LAB: Conjunto de elementos lógicos.
LCD: Pantalla de cristal líquido.
LE: Elemento lógico o CLB.
MINDRAY: Compañía que fabrica tarjeas OEM de signos vitales.
MIPS: Millones de instrucciones por segundo.
NEEK: Nios II Embedded Evaluation Kit, tarjeta de desarrollo con FPGAs
NIBP: Presión no invasiva.
NIOS II: Procesador embebido de 32 bits de arquitectura RISC.
NIOS II IDE: Interfaz de desarrollo y compilación para el procesador Nios II.
QUARTUS II: Programa para la descripción de hardware de las FPGAs de Altera.
RAM: Memoria de acceso aleatorio.
RESP: Respiración.
SD: Secure Digital, formato de tarjeta de memoria.
SINCRONIZADOR: Modulo hardware que asegura el intercambio de datos entre dos o más dominios de reloj.
SOPC: System **o**n **P**rogrammable **C**hip **B**uilder.
SPI: Serial Pheriphelal interface, es un estándar de comunicaciones.
SPO2: Oximetría.

SUNTECHMED: Compañía que fabrica tarjeas OEM de signos vitales.
TARJETA OEM: Modulo electrónico de adquisición, filtrado y transmisión de datos.
TEMP: Temperatura.
UEN: Unidad Estratégica de Negocios.
UML: Unified Modeling Lenguage.
VERILOG: Lenguaje de descripción de hardware.

CONCLUSIONES

Con el trabajo realizado en la Fundación Cardiovascular de Colombia y específicamente en su unidad estratégica de negocios Bioingeniería, se cumplieron con todos los objetivos propuestos en el plan de trabajo, llegando a la meta principal de diseñar y desarrollar interfaces graficas de usuario para la visualización de parámetros fisiológicos por medio de sistemas embebidos basados en FPGAs utilizando la tarjeta de desarrollo NEEK. El trayecto recorrido para llegar a la cumbre de proyecto realizado con éxito, demostró que el uso de tecnologías de hardware reprogramable es una excelente opción para el diseño de productos médicos, que pueden brindar a la UEN Bioingeniería una ventaja estratégica en el mercado para brindar servicios confiables, de bajo consumo y modificables no solo en su software si no también en su hardware.

El trabajo realizado trajo a la empresa aportes al conocimiento significativos y nunca antes visto para la empresa, lo que hace que esta pueda abrir nuevas líneas de desarrollo donde se apliquen tecnologías de punta como lo son las FPGAs. Por otro lado la UEN Bioingeniería vio positivo e importante el avance que pueden tener a futuro si siguen utilizando sistemas embebidos en los productos que fabrican sacando del mercado los basados en PC convencionales.

BIBLIOGRAFIA

AAMI. (2007). *ANSI/AAMI/IEC.* 60601-1-2.

Altera. (2009). *Cyclone III Device Handbook, Volume 1.* Altera.

Altera. (2008). *Cyclone III FPGA Starter Board Reference Manual.* San Jose: Altera.

Altera. (2011). *Embedded Peripherals IP User Guide.* San Jose: Altera.

Altera. (2009). *Logic Elements and Logic Array Blocks CIII51002-2.2 Volume 1.*

Altera. (2008). *Media Computer System for the Altera DE2 Board.* Altera.

Altera. (2008). *My First FPGA Design Tutorial.* San Jose: Altera.

Altera. (2008). *My First Nios II Software Tutorial.* San Jose: Altera.

Altera. (2010). *Nios II Software Developer's Handbook NII5V2-10.0.* San Jose: Altera.

Altera. (2010). *SOPC Builder User Guide UG-01096-1.0.* San Jose: Altera.

CERN Trajectory Measurement System. (03 de 12 de 2007). Recuperado el 24 de 09 de 2011, de Alpha Data: http://portal.beam.ltd.uk/support/cern/

Frazer, R. (4 de 9 de 2008). *eetimes*. Recuperado el 24 de 09 de 2011, de http://www.eetimes.com/design/embedded/4007550/Reducing-Power-in-Embedded-Systems-by-Adding-Hardware-Accelerators

Ho, H.-C. S. (2009). *Soft DSP Design Methodology of Face Recognition System on Nios II Embedded Platform.* Fifth International Conference on Information Assurance and Security.

Ningfeng Huang, H. W. (2009). *A Nios II Based English Speech Training System For Hearing-impaired.* International Conference on Computer Engineering and Technology.

Qiu Chuanfei, Z. W. (2010). *Porting μC/GUI To LCD and VGA in Nios II System.* International Conference on Measuring Technology and Mechatronics Automation.

Sutherland, S. (2009). *Is SystemVerilog Useful for FPGA Design & Verification?* San Jose: Sutherland HDL, Inc.

Wang Wei, Z. G. (2010). *The Design and Implementation of High-Speed Data Acquisition System Based on NIOS II.* 2010 International Conference on Computing, Control and Industrial Engineering.

WANG Ziting, Z. C. (2009). *Research of Image Capturing and Processing System Based on SOPC Technology.* Fifth International Joint Conference on INC, IMS and IDC.

Xilinx. (2010). Celebrating Customer Innovation. *Xcell Journal* , 36.