

PRÁCTICA EMPRESARIAL EN LA UEE BIOINGENIERÍA DE LA FUNDACIÓN
CARDIOVASCULAR DE COLOMBIA: DISEÑO, OPTIMIZACIÓN E IMPLEMENTACIÓN
DEL SISTEMA SOFTWARE-HARDWARE E INTERFAZ GRÁFICA DE INTERACCIÓN
CON EL USUARIO DE UN DISPOSITIVO DE MONITORIZACIÓN VITAL

JOSE MANUEL VELANDIA ROJAS

UNIVERSIDAD PONTIFICIA BOLIVARIANA - SECCIONAL BUCARAMANGA
FACULTAD DE INGENIERÍA ELECTRÓNICA
ESCUELA DE INGENIERÍAS
BUCARAMANGA
2014

PRÁCTICA EMPRESARIAL EN LA UEE BIOINGENIERÍA DE LA FUNDACIÓN
CARDIOVASCULAR DE COLOMBIA: DISEÑO, OPTIMIZACIÓN E IMPLEMENTACIÓN
DEL SISTEMA SOFTWARE-HARDWARE E INTERFAZ GRÁFICA DE INTERACCIÓN
CON EL USUARIO DE UN DISPOSITIVO DE MONITORIZACIÓN VITAL

JOSE MANUEL VELANDIA ROJAS

LIBRO FINAL DE PRÁCTICA EMPRESARIAL PARA OPTAR POR EL TITULO DE
INGENIERO ELECTRÓNICO

Supervisor académico:
MSc. CLAUDIA LEONOR RUEDA GUZMÁN

Supervisor empresarial:
ING. LEONARDO ANDRÉS RODRIGUEZ SALAZAR

UNIVERSIDAD PONTIFICIA BOLIVARIANA - SECCIONAL BUCARAMANGA
FACULTAD DE INGENIERÍA ELECTRÓNICA
ESCUELA DE INGENIERÍAS
BUCARAMANGA

2014

CONTENIDO

	Pág.
INTRODUCCIÓN.....	16
1. ESTUDIO INICIAL Y METODOLOGIA DE TRABAJO	21
1.1 COMPRESIÓN DE LOS SISTEMAS PREVIAMENTE DESARROLLADOS	21
1.1.1 Extracción de la dinámica base.....	22
1.1.2 Uso de propiedades intelectuales (IP).....	25
1.1.3 Cuestionamientos iniciales.....	26
1.2 INVESTIGACIÓN Y FAMILIARIZACIÓN CON EL ÁREA DE TRABAJO.....	26
1.2.1 Problemas que impulsaron la investigación.	26
1.2.2 Evidencia de la investigación.	27
1.2.3 Contacto con el personal médico.	31
1.3 ESTUDIO DE TÉCNICAS DE DISEÑO Y DESARROLLO	31
1.3.1 Técnicas de desarrollo hardware.	31
1.3.1.2 Lenguajes de descripción hardware.....	32
1.3.2 Técnicas de desarrollo software.....	33
1.3.2.1 Problemática del desarrollo software.	33
1.3.2.2 Administración de versiones software.	34
1.3.2.2.1 La realidad sobre “SourceTree”.....	35
1.3.2.2.2 Comprensión de la plataforma “SourceTree”.....	36
1.3.2.2.3 Detalles y opcionalidades finales en cuanto a la administración software.	38
1.3.2.3 Solución de la problemática en base a la administración de ventanas.	38
1.3.2.4 Cursos de programación en línea.	38
1.3.2.5 Cuestionamientos previos al desarrollo (Metodología software).....	39
1.3.2.6 Abstracción de la dinámica software.	40
1.3.2.6.1 Comparativa de la dinámica entre códigos software.	41
1.3.2.6.2 Dinámica del código C++ (Objetos).....	42
2. DESARROLLO FORMAL SOBRE LOS SISTEMAS DEFINIDOS	44
2.1 INCURSIÓN EN LOS ENTORNOS DE DESARROLLO.....	44
2.1.1 Entornos de desarrollo hardware sobre FPGAs.....	44
2.1.1.1 Entorno de compilación y desarrollo de la lógica programable de la FPGA.....	44
2.1.1.1.1 Compilación y abstracción del sistema hardware del MSV.....	46
2.1.1.2 Entornos de desarrollo hardware de sistemas integrados.	49
2.1.1.2.1 Desarrollo sobre entornos de sistemas integrados (Qsys).	51
2.1.1.2.1.1 Organización del espacio de trabajo.	51
2.1.1.2.1.2 Armado del sistema.....	53
2.1.1.2.1.3 Interconexión explícita de los sistemas maestro-esclavo.....	55

2.1.1.2.1.4 Revisiones, instanciación y compilación del chip final.	57
2.1.2 Introducción a entornos de desarrollo software.	60
2.1.3 Edición software alejada de los entornos dispuestos.	62
2.2 ELIGIENDO EL SISTEMA OPERATIVO DEL PROYECTO SOFTWARE	62
2.2.1 Adquiriendo derechos sobre las librerías del sistema operativo disponible.	63
2.3 SOLUCIÓN DE ERRORES DE DESCRIPCIÓN HARDWARE.	63
2.4 COMPRENSIÓN Y CREACIÓN DE COMPONENTES SOBRE EL ENTORNO QSYS	67
2.4.1 Presentación del componente.	67
2.4.2 Creación y edición del componente.	69
2.4.3 Temporización del estándar de comunicación “Avalon Memory Mapped”.	74
2.5 DESARROLLO SOFTWARE SOBRE EL ENTORNO NIOS II ECLIPSE	82
2.5.1 Presentación del entorno.	82
2.5.2 Creación y configuración de proyectos.	86
2.5.2.1 Comprensión y uso de la distribución de memoria hardware prevista.	92
2.5.3 Programación y verificación de la lógica software embebida en los proyectos.	93
2.6 EDICIÓN SOFTWARE SOBRE LA HERRAMIENTA QT CREATOR	97
2.6.1 Presentación de la herramienta.	97
2.6.2 Edición software de las librerías controladoras.	98
2.7 COMPRENSIÓN SOFTWARE DE LA DINÁMICA DEL SISTEMA OPERATIVO.....	100
2.7.1 Uso del sistema operativo uC/OS II como herramienta útil en la ejecución de las tareas software.	101
2.7.2 Dinámica software de una GUI junto al sistema operativo uC/OS II y lenguaje orientado a objetos (C++).	102
2.7.3 Dinámica software de redes junto al sistema operativo uC/OS II y lenguaje orientado a objetos (C++).	105
2.7.4 Avances de desarrollo software en cuanto a interfaz gráfica de usuario (GUI).	106
2.8 GESTIÓN DE ADQUISICIÓN VÍA E-MAIL PARA COMPRA DE DIRECCIONES FÍSICAS DE RED O MAC IEEE STD 802.....	115
2.9 “BIOS” LA IMPORTANCIA DEL CONTROL A BAJO NIVEL: AUTOMATIZACIÓN FINAL DEL DISPOSITIVO	116
2.9.1 Concepto de BIOS.	116
2.9.2 Dinámica BIOS sobre la lógica del equipo médico.	117
2.9.3 Estructura física y de conexión BIOS sobre el hardware que abarca el equipo médico.	117
2.9.4 Lógica software de BIOS como Firmware base del equipo médico.	120
2.9.4.1 Entorno Atmel Studio como herramienta de desarrollo software.	121
2.9.4.2 Estructura de proyectos software en el entorno Atmel Studio.	121
2.9.4.3 Programación de la compilación software sobre el entorno Atmel Studio.	122
2.10 DEPURACIÓN Y TRABAJO CONTINUO EN LA CORRECCIÓN DE ERRORES .	125

2.11 CARGA DE BATERIAS EN BASE AL DISPOSITIVO FUEL GAUGE E INTERACCIÓN CON LA INTERFAZ GRÁFICA. 125

3. APORTE AL CONOCIMIENTO 127

4. CONCLUSIONES 129

BIBLIOGRAFÍA..... 131

LISTA DE FIGURAS

Pág.

Figura 1. Sistemas externos del MSV e interacción con el mundo físico.....	22
Figura 2. Esquema de correlación de las tareas, respecto a la tarjeta de procesamiento principal.	24
Figura 3. Pruebas en video - Verificación de alarmas y representación de valores estáticos en monitor comercial.	27
Figura 4. Pruebas en video - Verificación de límites de alarmas y modos gráficos de las ondas vitales.....	28
Figura 5. Pruebas en video - Verificación de sonidos, pausa de alarmas, iconografía de interfaz e iconografía externa.....	29
Figura 6. Pruebas en video - Sistema de tendencia gráfica y tabular de datos estáticos, guardado de eventos.	30
Figura 7. Lógica interna de las FPGAs	32
Figura 8. Esencia del sistema de descripción hardware “Verilog”.	33
Figura 9. Estructura de la administración de versiones.	35
Figura 10. SourceTree como aplicación dentro del sistema operativo Windows 8.	37
Figura 11. Ciclo de abstracción software sobre sistemas de procesamiento integrado. ...	41
Figura 12. Estructura de la clase y conceptos propios del código C++.	43
Figura 13. Quartus II, descriptor de interfaces de tipo hardware.	45
Figura 14. Entornos de desarrollo hardware Qsys y SOPC Builder	46
Figura 15. Ejemplo de desarrollo sobre la plataforma Quartus.....	47
Figura 16. Sistema hardware base del equipo de monitorización en representación RTL	48
Figura 17. Diseño y desarrollo modular de sistemas embebidos basados en tecnología FPGA.....	51
Figura 18. Adecuación del entorno Qsys, detalles base de la herramienta	52
Figura 19. Descripción gráfica “Qsys” de un chip de desarrollo sobre FPGA.	54
Figura 20. Estructura de conexión sobre el procesador Nios II.	55
Figura 21. Mapa de direcciones relativas Avalon.	57
Figura 22. Opciones funcionales de edición e Instanciación final sobre el entorno Qsys .	58
Figura 23. Generación forzosa de archivos por parte del entorno SOPC Builder	60
Figura 24. Entornos “Java” de desarrollo software, sobre procesadores (Nios II) basados en tecnología de la corporación Altera	61
Figura 25. Módulo “mailbox” e interconexión de procesadores sobre el entorno Qsys.....	68
Figura 26. Estructura de comunicación “mailbox” sobre el entorno Qsys.	69
Figura 27. Pestaña de descripción básica del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.	70

Figura 28. Pestaña de archivos de descripción del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.	71
Figura 29. Pestaña de parámetros del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.....	72
Figura 30. Pestaña de señales hardware del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.	73
Figura 31. Pestaña de Interfaces del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.	74
Figura 32. Dinámica modificada de comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.	75
Figura 33. Dinámica predeterminada de comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.....	78
Figura 34. Timing 1 - Configuración aleatoria de la comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.	79
Figura 35. Timing 2 - Configuración aleatoria de la comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.	80
Figura 36. Timing 3 - Configuración aleatoria de la comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.	81
Figura 37. Entorno gráfico de desarrollo Nios II Eclipse.....	83
Figura 38. Entorno gráfico de desarrollo Nios II Eclipse – Edición de proyectos.	84
Figura 39. Entorno gráfico de desarrollo Nios II Eclipse – Opciones de búsqueda y edición.	85
Figura 40. Entorno gráfico de desarrollo Nios II Eclipse – Creación de proyectos software en base a descripción hardware.	86
Figura 41. Entorno gráfico de desarrollo Nios II Eclipse – Proyecto software: carpeta de desarrollo y carpeta BSP básica de funcionalidades y definiciones propias del sistema que envuelve al procesador.....	87
Figura 42. Entorno gráfico de desarrollo Nios II Eclipse – Edición final de las características de ejecución software a través de la herramienta de edición BSP.....	89
Figura 43. Entorno gráfico de desarrollo Nios II Eclipse – Edición BSP, pestaña principal de configuración básica.	90
Figura 44. Entorno gráfico de desarrollo Nios II Eclipse – Edición BSP, pestaña de descripción de las diferentes regiones memoria del procesador.	92
Figura 45. Repartición de las diferentes regiones de memoria existentes para el uso compartido entre los procesadores y el hardware de video dedicado.	93
Figura 46. Interfaz gráfica de programación software de los procesadores Nios II.....	94
Figura 47. Ejemplo de programación no volátil sobre la herramienta “Nios II Flash Programmer”.	95
Figura 48. Verificación de la existencia de los procesadores hardware a programar de manera software dentro de la FPGA.....	96
Figura 49. Consola de comandos “Nios II command Shell”.....	97
Figura 50. Herramienta de desarrollo software “QT Creator”.	98

Figura 51. Parte del código fuente de la librería controladora sugerida para el módulo mailbox "mailbox.c".....	100
Figura 52. Dinámica de operación del sistema operativo uC/OS II de la compañía Micrium.	102
Figura 53. Dinámica de operación software del primer procesador.....	103
Figura 54. Dinámica de operación software del primer procesador.....	104
Figura 55. Dinámica de operación de red sobre el software del segundo procesador.	105
Figura 56. Antiguo menú de historial de alarmas.	106
Figura 57. Antiguo menú de tendencias gráficas y representación de las variables estáticas sobre el número cero (0).....	107
Figura 58. Antiguo menú técnico del monitor.	107
Figura 59. Nuevo menú de técnico con calibración de presión del monitor.	108
Figura 60. Nuevo menú de tendencias gráficas con mayor rango de tiempo y representación de las variables estáticas nulas a través del texto "- - -" como sucede con los monitores comerciales en el mercado.	108
Figura 61. Nuevo menú de historial de alarmas en búsqueda.....	109
Figura 62. Nuevo menú de historial de alarmas final.	109
Figura 63. Modificaciones al menú de paciente.	110
Figura 64. Modificaciones al menú de PANI (Presión Arterial no Invasiva).....	110
Figura 65. Modificaciones a los mensajes de ondas vitales.	111
Figura 66. Nuevo modo calibración del "Touch" o táctil resistivo.	111
Figura 67. Nuevas técnicas de evasión de errores en memoria SD.	112
Figura 68. Nuevas técnicas de evasión de errores en cuanto a configuración de edad. .	112
Figura 69. Nuevas técnicas de evasión de errores de lectura en archivos corruptos.	113
Figura 70. Creación de los menús de redes inalámbricas con funcionalidad real.....	113
Figura 71. Corrección de la funcionalidad de marcapasos de la tarjeta de adquisición. ..	114
Figura 72. Variadas verificaciones de orden con ayuda del software interactivo propio de la tarjeta de adquisición vital vía terminal USB-UART.	114
Figura 73. Variadas verificaciones de orden con ayuda del comportamiento típico en monitores comerciales de adquisición de signos vitales.	115
Figura 74. Dinámica de funcionamiento BIOS.	117
Figura 75. Espacio de la BIOS sobre el diseño PCB de la tarjeta electrónica principal del sistema de monitorización vital.	118
Figura 76. BIOS en proceso de programación - Conexión adecuada del programador Atmel.	119
Figura 77. BIOS en proceso de programación - Conexión inadecuada del programador Atmel.	119
Figura 78. Entorno Atmel Studio versión 6.0 - Edición del proyecto de nombre "main". .	121
Figura 79. Aplicando la configuración de programación en el entorno Atmel para con el chip de la BIOS utilizado.	123
Figura 80. Verificando el archivo de programación software final "main.elf".....	123

Figura 81. Verificando las conexiones hardware y disposición del chip físico desde el entorno Atmel.	124
Figura 82. Ejecutando y verificando la respectiva programación vía programador Atmel USB.	124
Figura 83. Información de porcentaje del monitor de signos vitales sobre la interfaz gráfica en el menú varios.	126

GLOSARIO

SISTEMA: cualquier conjunto de más de una regla que establecen la lógica de funcionamiento de los entes implicados.

LÓGICA: estructura de funcionamiento de un sistema que puede ser entendida por el pensamiento del ser humano.

ENTORNO/PLATAFORMA: ambiente de trabajo compuesto por una serie de características, normas u opcionalidades propias del mismo, se caracteriza por facilitar la tarea del trabajador en cuanto a visualización rápida e interactiva inmediata, así como interfaz gráfica de usuario en muchas ocasiones.

INTERFAZ: es ese puente o conexión de dos sistemas o mundos, conexión en la cual se relacionan o se crea "interfaz" entre los mismos para lograr un contacto u o dinámica más afín en la visión práctica de su homólogo.

GUI: (interfaz gráfica de usuario) es una interfaz o puente de relación entre un sistema ya sea bien informático, embebido entre otros, siendo concedido u o accedido a través de un entorno gráfico siempre, el cual la mayoría de los seres humanos, para este caso, estamos acostumbrados; esto con el fin de hacer esta relación con los diferentes sistemas mucho más intuitiva con el usuario humano final.

EMBEBIDO: referido a aquel sistema computacional que pretende cumplir una respectiva y específica labor a través de la construcción y desarrollo de hardware y o software limitado-especializado para aplicaciones que requieran alta calidad estabilidad y confiabilidad en sus sistemas.

FIRMWARE: conjunto de instrucciones de máquina que describen una serie de acciones de más bajo nivel que la aplicación prestada por un software como tal, ya que son aquellas acciones que le preceden.

BIOS: concepto que abarca una serie de firmware, hardware y lógica de manejo de las diferentes instrucciones de acceso básico hacia un sistema físico específico y desde una perspectiva ligada a factores de interés de este tipo. Se asemeja a una interfaz que facilita la interacción física con los dispositivos de este tipo.

DISEÑO: conjunto de normas, reglas o definiciones que abarcan una herramienta tal y como se conoce y se maneja; el diseño trata de explicar las razones o el porqué de la forma final de los sistemas para así obtener el mayor aprovechamiento de lo que se desea con lo que se tiene dispuesto para el desarrollo de dicha herramienta.

DESARROLLO: proceso continuo de cambios sobre una herramienta con un fin de diseño específico.

IMPLEMENTACIÓN: acción real que abarca un cambio específico previamente desarrollado sobre las bases de diseño; un desarrollo abarca múltiples implementaciones.

PRUEBAS: se incluye en el proceso de desarrollo como una fase extra a la implementación, la cual pretende mejorar o depurar dicho cambio implementado para tener plena seguridad de funcionalidad sobre el mismo.

SECUENCIA: dinámica de funcionamiento que especifica una acción seguida de otra y nunca de forma paralela; de esta manera una secuencia mantiene la premisa de ejecución sobre las bases de la interrupción de eventos y cola de datos, ya que no existe la ejecución de una nueva instrucción mientras una instrucción previa se encuentre en etapa de proceso.

PARALELISMO: característica propia de los sistemas ensamblados en hardware independientemente, en donde un único sistema puede ejecutar varias acciones al mismo tiempo, ya que precisamente se fundamenta en una serie de sub-sistemas independientes, que se encuentran en espacios físicos independientes sobre el mismo sistema final e inclusive pueden o no depender del mismo pero siempre poseen o sostienen una mínima relación de orden para su relación sistemática.

SINCRONISMO: es un proceso que se realiza para establecer cierta característica de secuencialidad u orden en su mayoría a través del dominio del tiempo; esto con el fin de poseer un verdadero control u organización sobre sistemas desorganizados o independientes, como sucede con el paralelismo que soportan algunos de estos sistemas.

HARDWARE: estructura física y palpable de los sistemas.

SOFTWARE: estructura intelectual que se traduce en acciones para el hardware de los sistemas que envuelve; generalmente se asocia con la lógica programable que se compila y descarga en los diferentes dispositivos electrónicos, ya que finalmente son mandatos que se traducen en acciones verificables por el ser humano a través del hardware intrínseco de los sistemas.

ALTERA: compañía fabricante de múltiples dispositivos semiconductores comerciales entre otros servicios y soporte.

ATMEL: compañía fabricante de múltiples dispositivos semiconductores comerciales entre otros servicios y soporte.

MSV: abreviatura de monitor de signos vitales.

SGC: abreviatura de sistema de gestión de la calidad.

SGC: abreviatura de presión arterial no invasiva.

UEE: abreviatura de unidad estratégica empresarial.

VALIDACIÓN: en esta práctica se relaciona con el proceso de verificación funcional, efectiva y carente de fallas del dispositivo MSV bajo desarrollo, con el fin de ofrecer un soporte verídico, fiable y de peso de la funcionalidad del mismo para con el usuario final.

VERIFICACIÓN: proceso que aplica una serie de pruebas sobre pruebas que garanticen siempre la verdad o veracidad de las cosas que se pretenden someter a dicho proceso.

METROLOGÍA: área científica que tiene el objeto de verificar y establecer normas de medición mundialmente aceptadas para con los diferentes sistemas a comparar.

INTERNET: serie de todos los dispositivos electrónicos de red interconectados a través de los diferentes sistemas hardware y lógica software de protocolos de comunicación establecidos para este fin, con el propósito de que exista cierta interacción entre los dispositivos implicados.

PROTOCOLO: serie de secuencias de interacción que permiten una comunicación puente entre las diversas lógicas propias de cada dispositivo independiente; puente de comunicación que hace las veces de traductor universal sobre las bases de un único lenguaje o estándar asociado y aceptado por los dispositivos implicados en el uso del protocolo en cuestión.

RED: se refiere a partes separadas o pequeños grupos de interacción comunicativa entre los diferentes dispositivos electrónicos de red implicados en dicha red. Grupos o sectores de Internet asociados que finalmente conforman a la Internet.

PROGRAMACIÓN: conjunto de descripciones y algoritmos que conforman la lógica de funcionamiento de un sistema. Programar hace referencia a la acción de escribir dicha programación sobre una memoria electrónica destinada para este fin, en la cual se leen los algoritmos por el sistema que procesamiento que después las ejecuta.

LENGUAJE DE PROGRAMACIÓN: descripción literal y o numérica que ofrece una interfaz entre la lógica de instrucciones que entienden los seres humanos, a extensos lenguajes de máquina poco intuitivos, versátiles o molestos para amplios desarrollos que abarcan tiempos limitados.

FPGA: dispositivo semiconductor electrónico caracterizado principalmente por permitir el cambio de su lógica hardware interna a manera de programación descriptiva de la misma, siempre y cuando dicho dispositivo soporte tal lógica descrita. Cada fabricante posee una serie de entornos y herramientas para lograr el desarrollo sobre estos dispositivos al igual que sucede con los microcontroladores y se destaca su flexibilidad en cuanto a lógica física reconfigurable en comparación a otros dispositivos de dinámica netamente software.

MICROCONTROLADOR: dispositivo semiconductor electrónico caracterizado por permitir una serie de programaciones y ejecuciones software sobre un dispositivos de

procesamiento, memorias, periféricos de comunicación externa y otros adicionales, hardware que generalmente no se puede modificar en su estructura como en una FPGA.

DE BAJO NIVEL: referencia a básico, generalmente orientado a instrucciones y lógica de los lenguajes de programación, en donde se describe cada vez más específicamente cada paso, estructura y mínimo detalle posible de tales lenguajes.

DE ALTO NIVEL: referencia a versátil y rápido, generalmente orientado a instrucciones y lógica de los lenguajes de programación, en donde se pretende describir de manera resumida las diferentes instrucciones de funcionamiento de los lenguajes de más bajo nivel, generalmente se caracteriza por incluir compiladores que traducen una instrucción poco detallada, en instrucciones de bajo nivel que entiende la máquina, instrucciones que en la mayoría del tiempo no son evidenciadas por el programador.

EQUIPO DE TRABAJO: reunión de una serie de personas con conocimientos e ideas desordenadas de todo tipo, que pretende orientar las habilidades de sus integrantes a un solo fin, repartición de tareas y las mejores prácticas y metodología de trabajo posible; más en un ambiente de desarrollo de dispositivos electrónicos programables.

VERSIONES: conjunto de implementaciones parciales probadas y verificadas sobre una base medible, las cuales son organizadas y almacenadas para la posteridad con fines exploratorios que devienen del error humano y la carencia de perfección en los procesos.

SOPC BUILDER: entorno de desarrollo hardware basado en la unión de modularidad hardware prediseñada, con fines a la implementación de sistemas más confiables y rápidos.

QSYS: entorno de desarrollo póstumo al entorno SOPC BUILDER, que plantea una perspectiva más clara para el desarrollador, con mayores prestaciones en cuanto a la integración de las nuevas tecnologías y disminución de errores o BUGS de su entorno predecesor.

BUG: traduce “bicho o insecto” en ingles conocido por uso común en entornos de desarrollo de programación y se refiere a esa falla o afectación de un sistema, al punto de afectar su funcionamiento por completo; estos fallos muchas veces se evitan realizando procedimientos fuera de lo estándar y otras veces no es tan sencillo y limitan el trabajo de quien los enfrenta.

CULOMBIO: unidad de carga que se proporciona como referenciación de los elementos que almacenan energía eléctrica, esta es una medida derivada de la capacidad de carga de un electrón y sus unidades son “A*s” (Amperio segundo).

RESUMEN

TITULO: PRÁCTICA EMPRESARIAL EN LA UEE BIOINGENIERÍA DE LA FUNDACIÓN CARDIOVASCULAR DE COLOMBIA: DISEÑO, OPTIMIZACIÓN E IMPLEMENTACIÓN DEL SISTEMA SOFTWARE-HARDWARE E INTERFAZ GRAFICA DE INTERACCION CON EL USUARIO DE UN DISPOSITIVO DE MONITORIZACION VITAL

AUTOR: JOSE MANUEL VELANDIA ROJAS

FACULTAD: FACULTAD DE INGENIERÍA ELECTRÓNICA

DIRECTOR: MSc. CLAUDIA LEONOR RUEDA GUZMÁN

Esta práctica fundamenta su estudio e investigación en el campo de los sistemas embebidos más modernos, de alta capacidad y bajo costo, que pueden encontrarse hoy en día en el desarrollo de nuevas tecnologías, además de software embebido y dispositivos alternos.

El proyecto sobre el cual se fundamenta este trabajo, se encuentra bajo desarrollo y producción en las instalaciones de la Fundación Cardiovascular de Colombia y abarca la creación de un monitor de signos vitales (MSV) para su uso interno, así como venta nacional e internacional del equipo resultante.

Se ha hecho necesario retomar estudios sobre dispositivos de lógica digital programable (FPGA), sistemas embebidos, electrónica de potencia, sincronización de señales, concepto de interfaz gráfica de usuario en base a programación básica, programación referida a objetos y una corta etapa de diseño de PCBs sobre el software Altium Designer para el entendimiento base de la conectividad entre tarjetas e identificación de problemas de tipo físico que tuvieron cabida en el proyecto.

La idea del proyecto se centra en una tarjeta de adquisición de signos vitales siendo reconocida por una lógica de recepción y configuración en la FPGA que procesa las distintas señales sobre un dispositivo interactivo compuesto por una pantalla táctil resistiva en la cual se presentan las distintas señales adquiridas y las distintas configuraciones a las que puede acceder el usuario a través de una interfaz gráfica de usuario GUI. Lo anterior abarca una serie de lógica paralela para la adquisición de los datos, señales de video propias en RGB, montaje de procesadores embebidos partiendo de lógica hardware programable de propiedad intelectual de la compañía Altera (Versatilidad de aplicación software), entre otros agregados de portabilidad como lo es el control de carga de la batería y otros más externos de comunicación vía Internet y redes 3G que poco se mencionan para esta práctica.

PALABRAS CLAVES:

GUI, PROGRAMACIÓN, OBJETOS, ADQUISICIÓN, MONITOR, SIGNOS VITALES, DISEÑO, DESARROLLO, FPGA, DESCRIPCIÓN HARDWARE.

VoBo DIRECTOR DE TRABAJO DE GRADO

ABSTRACT

TITLE: BIOENGINEERING STRATEGIC ENTERPRISE UNIT OF THE CARDIOVASCULAR FOUNDATION OF COLOMBIA - GRADUATION PRACTICE: DESIGN, OPTIMIZATION AND IMPLEMENTATION OF SOFTWARE-HARDWARE SYSTEM FOR GRAPHIC USER INTERFACE FACILITIE OF A DEVICE FOR MONITORING VITAL

AUTHOR: JOSE MANUEL VELANDIA ROJAS

FACULTY: ELECTRONIC ENGINEERING FACULTY

DIRECTOR: MSc. CLAUDIA LEONOR RUEDA GUZMÁN

This practice based his study and research on embedded systems field always looking of the most modern, high capacity and low cost devices, which can be found nowadays on the go of new technologies development, in addition to embedded software and alternative devices.

The project on which this paper is based, is under development and production inside the Cardiovascular Foundation of Colombia place and covers the creation of a vital signs monitor (MSV) for use on the organization plant itself, as well as national and international sale of the resulting medical equipment.

It was necessary to resume studies about digital programmable logic devices (FPGA), embedded systems, power electronics, signal synchronization, concept of graphical user interface based on object-referenced and basic software programming, finally a short stage of PCB design on Altium designer software for database connectivity between boards and physical trouble identification that took place in the draft understanding.

The idea of the project focuses on an acquiring vital signs board being recognized by a receptional and configurational hardware logic on the FPGA, which processes the various signals on an interactive device comprising by a resistive touch screen in which the different signals acquired and different configurations for user input can be accessed through a graphical user interface GUI. This comprises a series of parallel logic for data acquisition, own video signals in RGB format, installation of embedded processors over programmable hardware logic from Altera's company intellectual properties (Versatility of software application), among other aggregates of portability as it is controlling battery charge and others more external about communication via Internet and 3G networks which are little mentioned for this practice.

KEYWORDS:

GUI, PROGRAMMING, OBJECTS, ACQUISITION, MONITOR, VITAL SIGNS, DESIGN, DEVELOPMENT, FPGA, HARDWARE DESCRIPTION.

VoBo THESIS DIRECTOR

INTRODUCCIÓN

La UEE (Unidad Estratégica Empresarial) Bioingeniería, hace parte de la FCV (Fundación Cardiovascular de Colombia) como la sub-empresa dedicada a la elaboración, producción, calibración, validación, prestación de servicios de salud, soporte técnico, adecuación de dispositivos médicos, así como el diseño y desarrollo de los mismos; esta labor tiene como fin no solo aportar en el desarrollo biomédico interno en beneficio del hospital de la organización, sino lograr un desarrollo de impacto tecnológico en la región, como ya se puede apreciar a través de la construcción actual del hospital internacional de la FCV y su amplia gama de servicios a la mano de la comunidad.

Uno de los objetivos más importantes de la organización siempre ha sido obtener reconocimiento y validación ante las diversas normas y estándares nacionales e internacionales, mientras que cada colaborador pueda aportar con calidad al usuario final a través de sus conocimientos profesionales de la mano con las políticas que se manejan internamente. Desde esta perspectiva el área de bioingeniería pretende alcanzar una visión en la que no solo a través de prestación de servicios de salud la organización puede verse beneficiada y reconocida, sino también en la medida que las capacidades que poseen los profesionales en la región puedan aportar para ser reconocidos como un centro tecnológico biomédico cada vez más especializado y profesional, desarrollar tecnologías propias que pudiesen ayudar a miles de personas, mientras los nuevos desarrollos se abren paso para competir en el mercado contra los bajos precios de países netamente productores, esto a través de las nuevas tecnologías que se implementan en el área; a este objetivo principal le apunta el proyecto del monitor de signos vitales y su interfaz gráfica de usuario que se plantea en esta práctica, un proyecto innovador que busca reconocimiento del área para sacar adelante la UEE de bioingeniería y ser tenidos en cuenta más profesionalmente y así de la mano con el apoyo de la organización, proponer mejores proyectos de mayor complejidad y de mayor calidad.

En la actualidad se posee un prototipo funcional del MSV e inclusive una pequeña producción de prueba, de la cual se pretenden abarcar todas las fallas, cosas que se pasaron por alto y en resumen reunir toda la información necesaria del equipo, para hacerlo realmente comercial y depurar por completo sus funcionalidades.

A lo largo de este tiempo, que inclusive supera el periodo de esta práctica empresarial, se han venido haciendo múltiples investigaciones, inversión de tiempo en situaciones no planeadas, autogestión del nuevo conocimiento, cursos externos y demás; también se han afrontado múltiples problemas proyectados en esta práctica de grado, tiempo que ha sido de gran ayuda para encontrar tanto debilidades que debe superar cualquier profesional, como fortalezas que se deben seguir cosechando para así transmitir dicho conocimiento en pro del desafío que supone este inmenso proyecto en el que se han invertido tantos esfuerzos de un equipo de trabajo comprometido.

En este punto del desarrollo actual del monitor de signos vitales, se tiene una sólida base hardware y mejoras de todo tipo en cuanto a estabilidad del sistema, nuevos desarrollos funcionales y más rediseños con pruebas e implementaciones finales que pretenden

mejorar el rendimiento actual del monitor en cuestión. Es entonces momento para finiquitar algunos desarrollos y generar algunos instructivos al conocimiento referente a la nueva herramienta de diseño hardware “Qsys”, para luego ingresar en el campo de desarrollo software, necesario para abordar muchas de las tareas funcionales del equipo, ya que el mundo hardware no se encuentra muy acoplado para todos los procesamientos que se pretenden realizar, se hace importante sacar ventaja también de la secuencialidad que proporciona la interacción software con el equipo de monitorización.

En la primera etapa del proyecto se abordaron temas teóricos referidos a la utilización de sistemas funcionales y comprensión física del montaje básico del monitor, temas teóricos respecto al desarrollo software que hace parte de la segunda etapa de esta práctica y uso de sistemas hardware, diseño y desarrollo sobre FPGAs y embebidos de todo tipo.

Hubo un espacio bastante amplio para la familiarización con el MSV en cuestión y los diversos entornos de desarrollo, así como investigación y referenciación con sistemas comerciales de monitoreo existentes en el hospital de la organización (ICF), tiempo mucho mayor al que aborda esta práctica, pero pudo ser administrado en su momento; esta aclaración se realiza para dar una idea al lector acerca de la importancia de la investigación y el planteamiento de los requerimientos.

Una vez adquiridos los conocimientos y haber absorbido el desarrollo hardware, la segunda etapa del proyecto se centró en adquirir esa habilidad de software que permitirá modificación de todo tipo en la supresión de errores de interfaz entre otros, errores tanto complejos como evidentes saldrán a la luz de la mano con el trabajo del área de validación y metrología de bioingeniería, quienes serán los encargados de poner a prueba el equipo y de ver aquello que genere no conformidades o fallas críticas al sistema del equipo médico.

Este último ha sido un proceso bastante largo debido a la inmensidad del código y adentra los desarrollos en adquirir ese nivel requerido de experticia desde el inicio de la práctica, lo cual permite abordar la responsabilidad de las mejoras y fallas que el equipo vaya presentando a lo largo de este proceso.

Objetivo general

- ❖ Diseñar, optimizar, e implementar la estructura del sistema software-hardware básico para la interfaz gráfica de usuario, de un dispositivo de monitorización vital, utilizando conocimientos teóricos y prácticos de tipo programático, digital y manejo de datos, en la línea de desarrollo tecnológico de sistemas embebidos. Generación y corrección de librerías en código C/C++.

Objetivos específicos

- ❖ Implementar y describir el sistema modular software-hardware que guiara las tareas del dispositivo de monitorización, sistema operativo y entornos de desarrollo innovadores aun no utilizados para este tipo de tarjetas de desarrollo en base a los conocimientos del estudiante e investigaciones iniciales requeridas.
- ❖ Optimizar el sistema para reducir las posibles fallas en módulos de video existentes, implementar señales de video más estables que eviten limitaciones de frecuencia y ruido interno.
- ❖ Diseñar e implementar nuevos dispositivos de comunicación interna más eficientes.
- ❖ Integrar los diferentes periféricos de video, comunicación local, envío de señales al exterior del procesamiento software por medio del softcore Nios II.
- ❖ Rediseñar y programar el software embebido en el procesador principal, gestionar los gráficos e interfaz de ventanas del sistema actual, en un entorno optimizado de mayor resolución y solucionando problemas de estabilidad.

Actividades

- ❖ Estudiar las diversas interfaces de monitores ya desarrollados para estructuración e implementación en el monitor De signos vitales.

Se hace imprescindible estudiar la estructura de monitores actuales como guía en el proceso de creación del nuevo dispositivo de monitorización en pro de agilizar el proceso de diseño y desarrollo del mismo.

- ❖ Investigar métodos de descripción hardware y software avanzados, facilidades de uso entre sistemas operativos y demás pautas que definirán el desarrollo de ahora en adelante.

Se deben retomar conocimientos base para aplicar diseños avanzados que generen mayor desempeño, robustez, sincronía y confianza al momento de ser implementados e integrados con otros diseños; se deben definir las herramientas (entornos, códigos, sistemas operativos, interfaces físicas) que se van a utilizar desde un inicio, evitando cambios inesperados que afecten tanto el desempeño del sistema que puedan a su vez generar demoras a los tiempos establecidos para el desarrollo del sistema de monitorización.

- ❖ Diseño y descripción de las bases hardware embebidas para el dispositivo en cuestión, implementación, rediseños y pruebas finales.

Esta etapa abarca pruebas de cada módulo hardware por separado y de ser necesario, implementación funcional con los demás módulos, detección de fallas, optimización y rediseño de lógicas actuales, creación de nuevos módulos, e integración interactiva con el sistema gráfico y adquisitivo global.

- ❖ Implementación software en el procesador Nios II, generalidades, detalles de funcionamiento y guías de uso en el entorno previamente definido.

Esta etapa incluye el uso del hardware existente hasta el momento, siendo administrado por software de forma básica por el procesador embebido en las diferentes tareas que deben ser implementadas en tiempo real.

- ❖ Diseño de la interfaz gráfica básica aplicada a través del uso de librerías software orientadas a objetos. Manejo de “sprites” en memorias, carga no volátil de las distintas descripciones.

Esta etapa incluye el desarrollo software, uso de librerías y convivencias de códigos C y C++, administración de copiosos masivos de memoria para generar una interfaz relativamente fluida intuitiva e interactiva con el usuario, además de coherente respecto los diferentes módulos de aplicación hardware, controladores, drivers, táctiles de pantalla y demás capas físicas que componen el dispositivo, pruebas e implementaciones experimentales, verificaciones de funcionamiento, rediseños finales y entrega final de interactividad usuario-interfaz.

Cronograma de actividades

Actividades.	Semana.																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Estudiar el monitor previo.	█	█	█	█	█	█			█	█	█	█	█	█			
Estudiar métodos de descripción avanzados y definir las herramientas a utilizar a lo largo del desarrollo.		█	█	█					█	█	█	█	█				
Diseño y descripción de las bases hardware embebidas para el dispositivo de monitorización, sobre una específica tarjeta de desarrollo basada en tecnología FPGA.				█	█	█	█	█									
Implementación software en el procesador Nios II, manejo del entorno previamente definido.									█	█	█	█					
Diseño de la interfaz gráfica básica interactiva con el usuario, uso de librerías software orientadas a objetos, manejo de memoria y carga no volátil de las distintas descripciones.												█	█	█	█	█	█

1. ESTUDIO INICIAL Y METODOLOGIA DE TRABAJO

Antes de iniciar a realizar cualquier tipo de desarrollo respecto al monitor de signos vitales referido a esta práctica, se hace un previo estudio su estructura, concepto, bases, planeación y lo más importante, su electrónica y funcionamiento en forma global, esto con el fin de evitar el choque de información que suele suceder cuando se realiza la organización de la información concerniente, después surge la necesidad de ingresar en un nuevo tema y dado a la falta de conocimiento del mismo, se terminan extraviando los conceptos, se cometen errores, se pierde tiempo y se termina necesitando del apoyo y la información que puedan facilitar los compañeros de trabajo en momentos poco oportunos; lo anterior podría abarcar bastante tiempo y dedicación poco disponibles en momentos adecuados, orillando al estudiante a ser un mal autodidacta y a atrasarse en su cronograma, sin embargo son estrategias de aprendizaje que se pretenden evitar interviniendo de manera ordenada y oportuna a cada labor que se vaya presentando, todo en función de aprovechar mejor los espacios, en la medida que estas pautas definen la metodología de trabajo.

En la investigación que se realizó de mano de los ingenieros electrónicos Sherneyko Plata Rangel y Jose Pablo Pinilla Gómez, desarrolladores actuales del monitor en cuestión, se extrajo un vasto conocimiento de todo lo que refiere al sistema electrónico y la tarjeta de adquisición, diseños completamente propios de todo el sistema embebido, tarjetas electrónicas, programación hardware en FPGAs, software embebido de interfaz gráfica de usuario (GUI), utilización de licencias de software en el uso de propiedades intelectuales de la compañía Altera, manejo muy básico del software Altium Designer para diseños de PCB en el proceso de la recolección de la información necesaria para incursionar en los diferentes sistemas físicos, derechos de uso en cuanto a las librerías de uC/OS II como sistema operativo en la tarea de compilación de la plataforma embebida y demás material que se ira absorbiendo con el paso de la lectura.

1.1 COMPRESIÓN DE LOS SISTEMAS PREVIAMENTE DESARROLLADOS

Entrando un poco más en materia acerca de los sistemas que componen a detalle el monitor, se inicia con el avistamiento físico del mismo, del cual se extraen los dispositivos clave más externos y se determina el funcionamiento base a gran escala del equipo, extracción de conocimiento que pretende resolver en la Figura 1 que representa la composición del monitor en cuestión y su interacción con el medio, así se tiene una idea más clara de lo que se posee, función y lo que se pretende hacer con esto.

1.1.1 Extracción de la dinámica base.

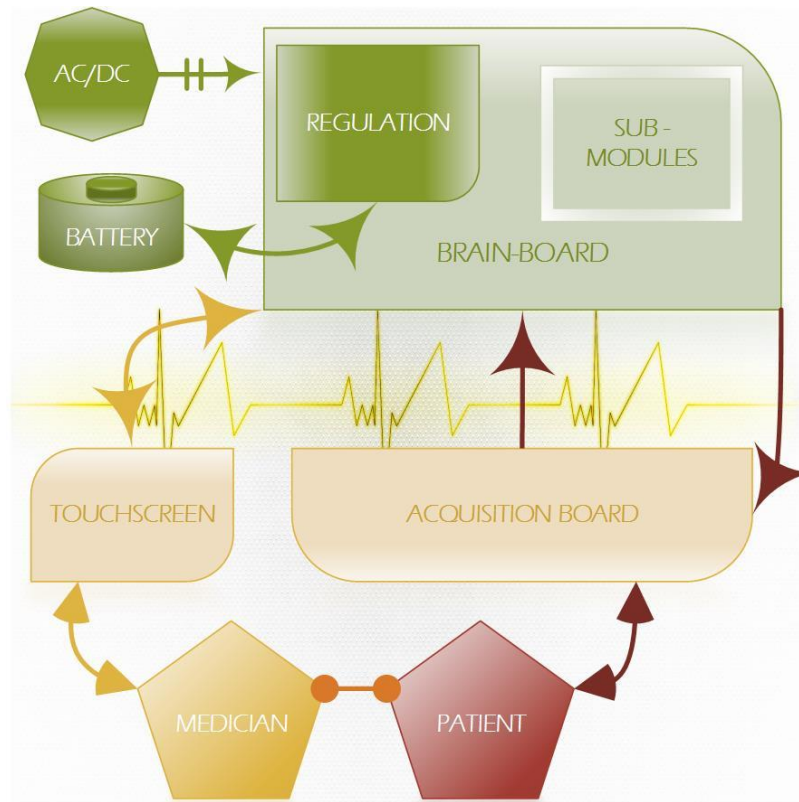


Figura 1. Sistemas externos del MSV e interacción con el mundo físico.

Como se puede apreciar en la Figura 1, el proyecto del monitor de signos vitales (MSV) trata de abarcar el concepto de una tarjeta de adquisición de signos vitales siendo montada o ensamblada en una tarjeta electrónica para cumplir múltiples labores de procesamiento (Mainboard); un sistema gráfico de pantalla táctil pretende que el usuario (Personal asistencial médico) pueda acceder a todas las funcionalidades y opcionalidades ofrecidas por dicha tarjeta de adquisición, que en esencia, pretende trabajar sobre un determinado tipo de pacientes a través de los conocimientos profesionales que pueda aportar dicho personal asistencial al sistema de monitorización vital en general; cabe aclarar que la tarjeta de adquisición de signos vitales es un tipo de tarjeta completamente comercial que ha sido probada con ciertos estándares médicos especificados en su respectiva hoja de datos (la cual es proporcionada por su fabricante) y por ende no deviene ningún desarrollo funcional de la misma por parte de los ingenieros, sin embargo se debe diseñar el sistema de comunicación con la misma y ser muy cuidadoso con ciertos aspectos para con el sistema embebido global y la integridad del usuario final (médico - paciente).

Este concepto viene acompañado por ciertos agregados desde el punto de vista electrónico y programático que poseen la mayoría de monitores comerciales, en los que se pretende dar al usuario cierto nivel de estabilidad y portabilidad con el uso de baterías, sub-módulos de comunicación, así como realimentación visual y auditiva, registro de datos en tarjeta de memoria SD que aporten al seguimiento de tendencia fisiológica médica del paciente y resolución de problemas legales que puedan surgir respecto a fallas propias del equipo, problemas de configuraciones erróneas (factor humano), menús de orientación para el usuario final y otros extras que ayudan a manipular el monitor de manera más precisa y confiable; lo anterior a través de la interacción organizada que proporciona la interfaz gráfica de usuario (GUI) del dispositivo.

Otro punto importante al momento de diseñar la interacción gráfica que se da a través de la pantalla táctil, es evitar el índice de desorganización que pueda reflejarse en el usuario debido a dicha interfaz y así sin importar la complejidad que pueda abarcar el monitor, el usuario final siempre suponga un mínimo de ajuste necesario al momento de su uso en pacientes y se refleje la espontaneidad de manejo del dispositivo, este es un factor crucial para que el usuario desee adquirir el producto, es un índice de organización de los elementos que debe ser analizado a detalle y que pocos monitores de signos vitales en el mercado poseen, ya que los menús se tornan algo complicados si no se tiene la suficiente experticia en su manejo, como ya se ha evidenciado por los distintos colaboradores del área, el estudiante, e inclusive las opiniones y experiencias del mismo personal asistencial, lo cual supone uno de los más grandes retos de este proyecto, ya que plantear este concepto sobre un medio de programación embebido tan poco explorado es claramente incierto.

En este punto ya se tiene la idea base y el concepto respecto al cual se desarrollará un monitor de signos vitales, sin embargo hay que ser un poco más específico respecto al desarrollo electrónico se refiere, ya que aunque se posee un concepto y unos requerimientos que se reparten en funciones para con el equipo de trabajo, no se tiene mayor idea del tema específico en cuanto a descripciones del sistema y dispositivos electrónicos a usar; es por ello que hay que darle paso al estudio del monitor en cuanto a los sistemas electrónicos específicos.

En la Figura 1 se pudo evidenciar la existencia de una tarjeta cerebro (Mainboard) que alberga una serie de sub-módulos y regulación de alimentación para el equipo en general, sin embargo quien lleva el verdadero control de esta serie de elementos como sub-módulos, la sincronización para el video de pantalla, la adquisición de datos, comunicación interna y externa, parte del control de carga de la batería entre otros, es el control referido al procesamiento que realiza una FPGA sobre casi todos los periféricos físicos base explicados con anterioridad y muchos otros que se verán a través del desarrollo de esta práctica. Para entender de forma más clara la función de dicho procesamiento se plantea la Figura 2.

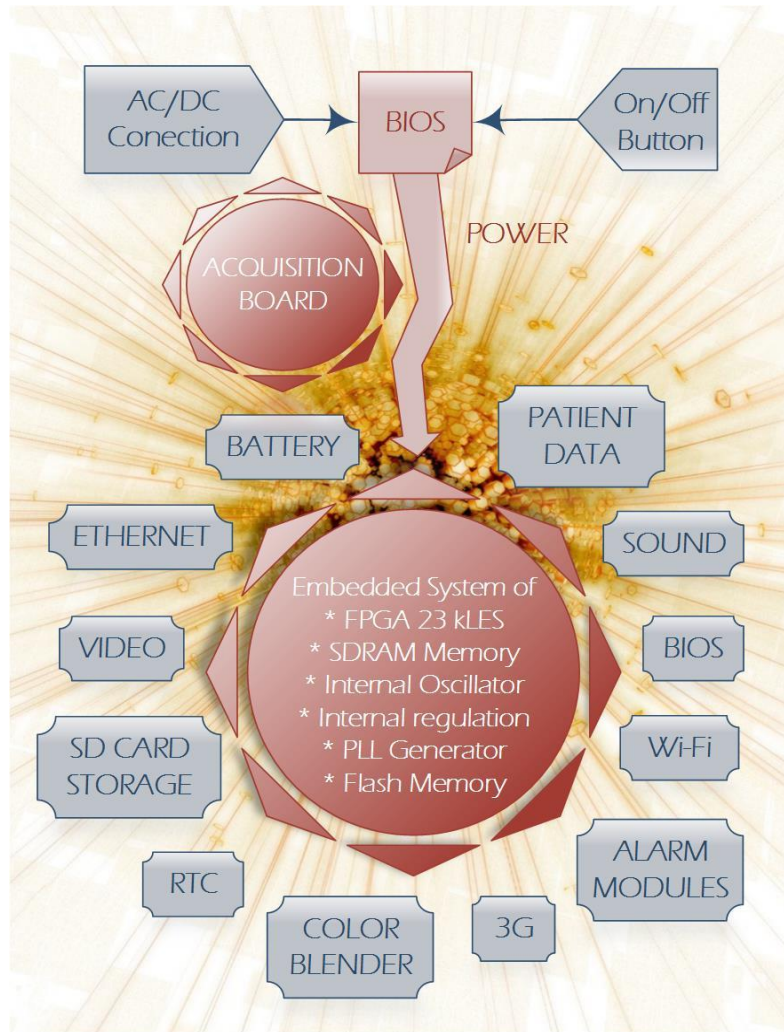


Figura 2. Esquema de correlación de las tareas, respecto a la tarjeta de procesamiento principal.

Como se puede apreciar en la Figura 2, la FPGA no actúa por completo sola, ya que el sol representativo de en medio del esquema, plantea toda una tarjeta de desarrollo dotada de múltiples memorias, oscilador y regulación propia embebida además de su respectiva FPGA de cierta capacidad lógica; todos estos dispositivos embebidos dentro de la tarjeta, son un puente crucial para evitar gasto de lógica interna, evasión del ruido dado a la conexión interna de la memoria volátil de alta frecuencia (SDRAM), generadores de oscilación estables, memoria no volátil prefabricada y conectada internamente, quienes a su vez evitan cierto gasto de pines libres que requerirán los demás dispositivos externos con los que se tendrá que comunicarse dicho módulo de procesamiento principal (FPGA); interacción con el exterior ejemplificada a través de las puntas del sol de la Figura 2.

El funcionamiento normal del sistema se basa en una secuencia de tareas, en donde la FPGA al igual que la tarjeta externa de adquisición de signos vitales, serán alimentadas y

encenderán gracias a la acción ofrecida por un microchip que cumple la función de bajo nivel conocida como BIOS del sistema, encargada de la carga inicial y procesos base de todo sistema embebido, esta a su vez otorgará control sobre la alimentación y el encendido del equipo una vez este sea energizado correctamente, es entonces donde la FPGA podrá tomar el control de múltiples periféricos y módulos externos a través de sus múltiples pines de entrada, salida y de bi-dirección, todo esto mientras su descripción programática sea correctamente cargada a la misma a través de una serie de memorias flash no volátiles previamente tratadas por los ingenieros electrónicos y el estudiante en práctica; por otra parte la información no volátil será volcada a la memoria volátil del sistema conocida como SDRAM, quien gestionará un procesamiento mucho más rápido y funcional referente al desarrollo de programación software, gestión propia de los procesadores softcore descritos en hardware (propiedad intelectual de la compañía Altera) quienes junto a otros módulos de video dentro de la misma FPGA y memorias, logran una interfaz gráfica perfectamente funcional, fluida y en armonía con todos los periféricos y descripciones hardware que componen el sistema.

Cabe aclarar que el microchip que fue designado a ocupar su función como BIOS del sistema, es un chip de la compañía Atmel perfectamente comercial y por ende también conlleva cierto desarrollo en programación software, esta programación es llamada a funcionamiento a través de una pequeña memoria no volátil construida internamente dentro de este chip, así que no se requirió de otra memoria externa para cumplir con el objetivo de plantear una BIOS completamente funcional; sin embargo se aclara que las funcionalidades y el desarrollo en este tipo de dispositivos conllevan un vasto estudio de sus respectivas hojas del datos, las cuales son proporcionadas por el fabricante y se analizaron en el transcurso de la práctica, esto con el fin de corregir problemas, arreglar y agregar funcionalidades que surgieron en medio del desarrollo de todo el equipo médico por la generación de nuevos requerimientos emergentes de los cuales no se está exento.

1.1.2 Uso de propiedades intelectuales (IP).

Este es el punto en donde los mundos hardware y software se encuentran para complementar sus faltas y hacer un sistema completo con todas las ventajas disponibles y las desventajas que vayan surgiendo de las cuales se puede aprender a aplicar en proyectos futuros; igualmente poner en conocimiento la flexibilidad que se evidencia al trabajar con dispositivos de la compañía Altera, ya que en cuanto a sus propiedades intelectuales y desarrollos propios se refiere, siempre existe un uso verificable en comparación a otras compañías, las cuales obligan al usuario a depender de una licencia para cualquier mínima prueba de las funcionalidades de sus desarrollos intelectuales; en este aspecto la compañía Altera no se cierra y permite el uso restringido pero funcional de sus sistemas, lo cual resulta ser bastante útil en situaciones que conllevan gastos financieros no contemplados, liberando así cierta presión y encadenamiento a licencias, más aún en sistemas que atraviesan por sus desarrollos iniciales sin una garantía total de funcionamiento, como lo evidenciaría cualquier desarrollo de sistemas embebidos.

1.1.3 Cuestionamientos iniciales.

Ya se tiene el objetivo, ya se tiene el sistema, ya se tienen las funcionalidades y ya se tiene una base más clara de la situación del desarrollo electrónico; sin embargo es el momento para preguntar, ¿Porque FPGA? y entrar en contexto a una corta etapa de investigación en cuanto a ventajas y desventajas, capacidades y necesidades que posee el proyecto, con el fin de tener aún más claro que se puede y que no se puede hacer con los recursos predispuestos; esto se da ante la necesidad de evitar problemas que surgen con los nuevos requerimientos, en donde un nuevo planteamiento que abarca un respectivo ajuste genera compromisos que se son imposibles de cumplir ya sea por los recursos, falta de conocimientos actuales o por la presión que generan decisiones apresuradas en términos de responsabilidad con las labores asignadas sobre el equipo de trabajo, acción que se traducirá en una cola de labores inconclusas que afectan la fluidez del proyecto y la calidad final vista por los directivos, que desde el desarrollo es evidenciada por todo el equipo de trabajo y el producto en general.

De la investigación que se plantea sobre el dispositivo de procesamiento principal se ajustan las siguientes dudas sencillas pero claves para el entendimiento del mismo, junto con otros conceptos que deben ser asimilados desde un comienzo.

¿Qué es una FPGA?

¿Para qué se utiliza una FPGA?

¿Cómo se logra el desarrollo en una FPGA?

¿Cuáles son los aspectos que destacan el desarrollo sobre una FPGA?

¿Cómo ha sido la evolución de las FPGA?

¿A qué se refiere un sistema embebido?

¿Cómo se aplica el concepto de sistemas embebido a este proyecto?

¿Cuáles son las ventajas y desventajas de un sistema embebido?

¿Desventajas de los sistemas embebidos?

1.2 INVESTIGACIÓN Y FAMILIARIZACIÓN CON EL ÁREA DE TRABAJO

Existe una etapa intermedia que plantea definir los requerimientos que debe tener el monitor para asemejar su funcionamiento comercial, dado que en su momento no se tuvieron en cuenta algunos factores importantes; este trabajo ayuda a generar ideas en la creación, diseño y modificación de los sistemas que se abarcaran más adelante, en función de distribuir las tareas necesarias para el equipo de trabajo en cuestión.

1.2.1 Problemas que impulsaron la investigación.

En el proceso de definición de los requerimientos del proyecto existieron situaciones que no se analizaron muy claramente, no se definieron como debió en su momento o sencillamente no se pensaba que pudiesen afectar al sistema, sin embargo se evidenció

como a puertas de depurar el proyecto, ese bajo nivel de planeación que en cierto punto afectaron los recursos, el tiempo y calidad de los sistemas. Establecer una planeación clara para un proyecto, es un proceso tan fundamental como en el desarrollo exitoso del mismo y toda esta problemática orillan al equipo de trabajo a redefinir ciertos requerimientos en base a un proceso de investigación-definición-solución.

Dado que el MSV de la FCV pretende a futuro ser un sistema igualmente competitivo-comercial, la idea de comparar el monitor en cuestión con los equipos de la clínica, surge como una rápida salida que pretende abordar con precisión cada detalle fallido, indefinido o faltante en el equipo; de esta manera, una de las tareas previamente asignadas al estudiante se cumple a cabalidad y se ve reflejada en las múltiples visitas comparativas realizadas en la clínica, informes de registro de los datos investigados, así como pruebas visuales y auditivas ejecutadas en su momento para el sistema de gestión de la calidad (SGC) de la organización.

1.2.2 Evidencia de la investigación.

En las imágenes comparativas, se evidencia al monitor comercial en la parte izquierda y al monitor FCV (prototipo funcional) en la parte derecha más su respectiva descripción.

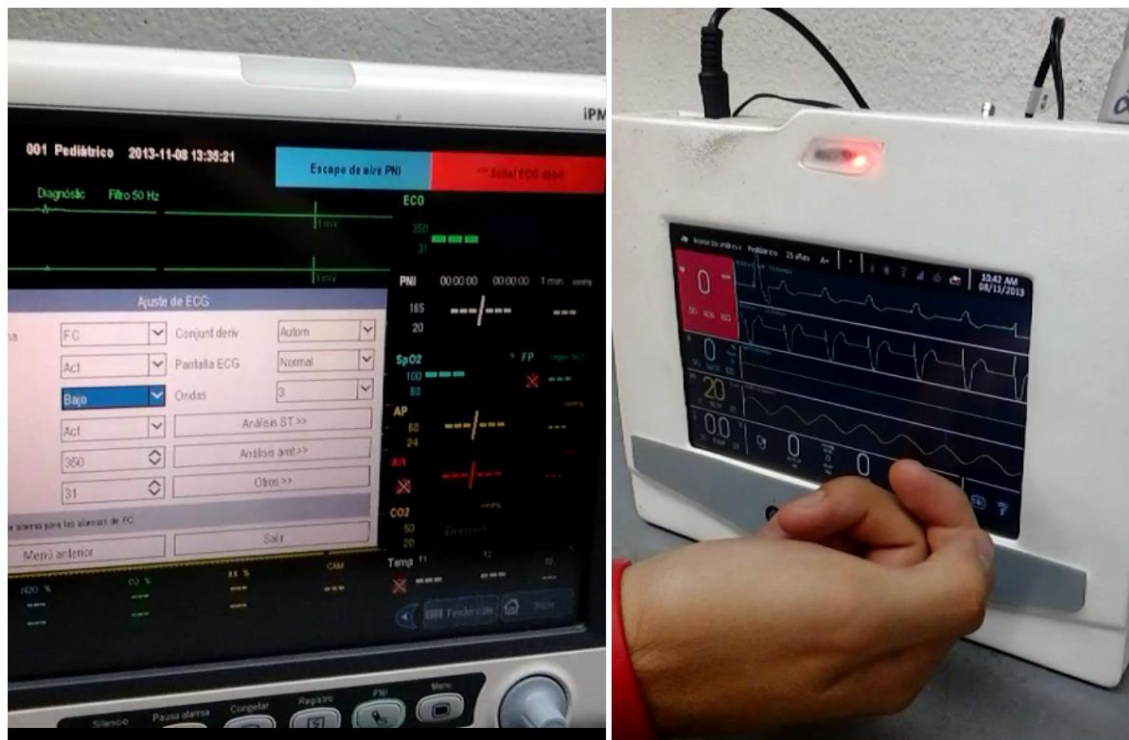


Figura 3. Pruebas en video - Verificación de alarmas y representación de valores estáticos en monitor comercial.

En la prueba de la Figura 3, se pudo verificar como las alarmas poseían un nivel determinado el cual no cambiaba su configuración por parte del usuario a menos que el paciente entrara en estado crítico, lo cual se tenía implementado pero no de la manera que el monitor comercial lo proponía. También se evidenció la representación escrita de alarmas técnicas (como lo son las caídas de sensor o fugas de PANI), lo cual no poseía el equipo de desarrollo en absoluto, por otro lado la representación gráfica de los valores estáticos en reposo, siempre se establecía a través de líneas horizontales de la forma “---” para el monitor comercial y no a través de números cero (0) como se refleja en el monitor de desarrollo; de esta manera se pudieron recoger varias percepciones importantes en el diseño y desarrollo de los cambios funcionales y de interfaz pertinentes, evitando siempre que los nuevos cambios tuviesen una intervención negativa sobre los desarrollo existentes.

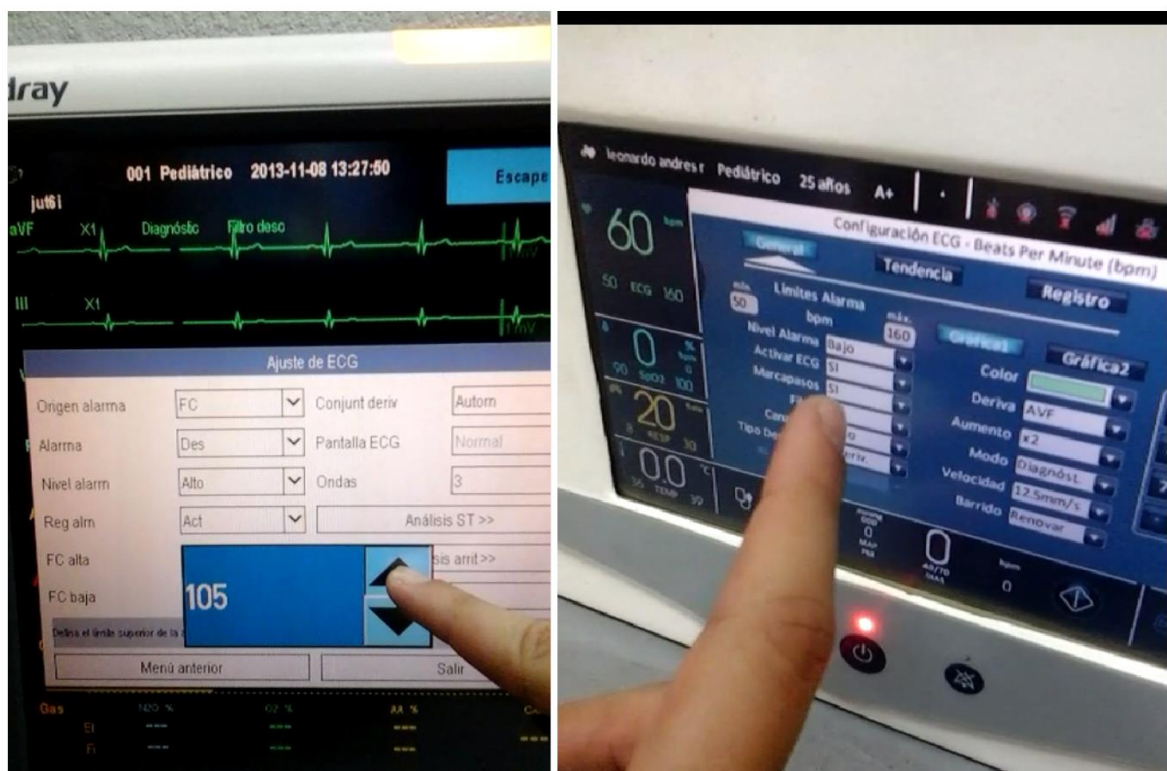


Figura 4. Pruebas en video - Verificación de límites de alarmas y modos gráficos de las ondas vitales.

En la representación de la Figura 4, se pudo verificar cada límite de alarma y configuración predeterminada que tuviese el monitor comercial con respecto a los ajustes que conlleva el cambio de tipo de paciente, lo cual pudo ser validado junto a la información referida en los respectivos manuales disponibles de estos equipos comerciales, esto en comparación a la opcionalidad de configuración en cuanto a límites que ofrecía en monitor de la FCV y el correcto ajuste de los mismos; por otro lado también se verificaron los diferentes modos de velocidad, barrido, amplitud y modo de onda que pudiese existir para cada parámetro, en donde cada cual podía informar de su situación a

través de un mensaje superior al parámetro concerniente, cosa que aunque ya estaba implementada en el monitor local, existía una escasez de información poco funcional para el proyecto, lo que conlleva al aumento de ciertos mensajes informativos y optimización de los espacios a través de técnicas de pestañeo visual para los campos necesarios.

Otras observaciones y optimizaciones respecto a datos de alarma y comportamiento de los mensajes informativos distribuidos por toda la pantalla se llevaron a cabo a través de las múltiples reuniones con el equipo de trabajo.



Figura 5. Pruebas en video - Verificación de sonidos, pausa de alarmas, iconografía de interfaz e iconografía externa.

La prueba de la Figura 5 arroja un resultado muy estético en cuanto a iconografía necesaria por el diseñador industrial a cargo del proyecto. Respecto al diseño auditivo-visual de la interfaz, se evidenció la necesidad de realizar muchos cambios de tipo funcional ya que el monitor de desarrollo no poseía ambas opciones de sonido (Silencio y Pausa de alarmas), sino que la función de pausa temporal de sonido realizaba la labor de silencio del sonido de forma implícita, sin embargo se decidió que el equipo médico debía

garantizar seguridad al paciente y por ende no debía poseer dicho botón de silencio permanente, por otro lado la iconografía debía ser cambiada dado que la campana de silencio era usada para representar la función de pausa temporal de alarmas, la cual le correspondía el icono triangular de líneas punteadas, lo cual concuerda con las normas para este tipo de iconografía médica en monitores.

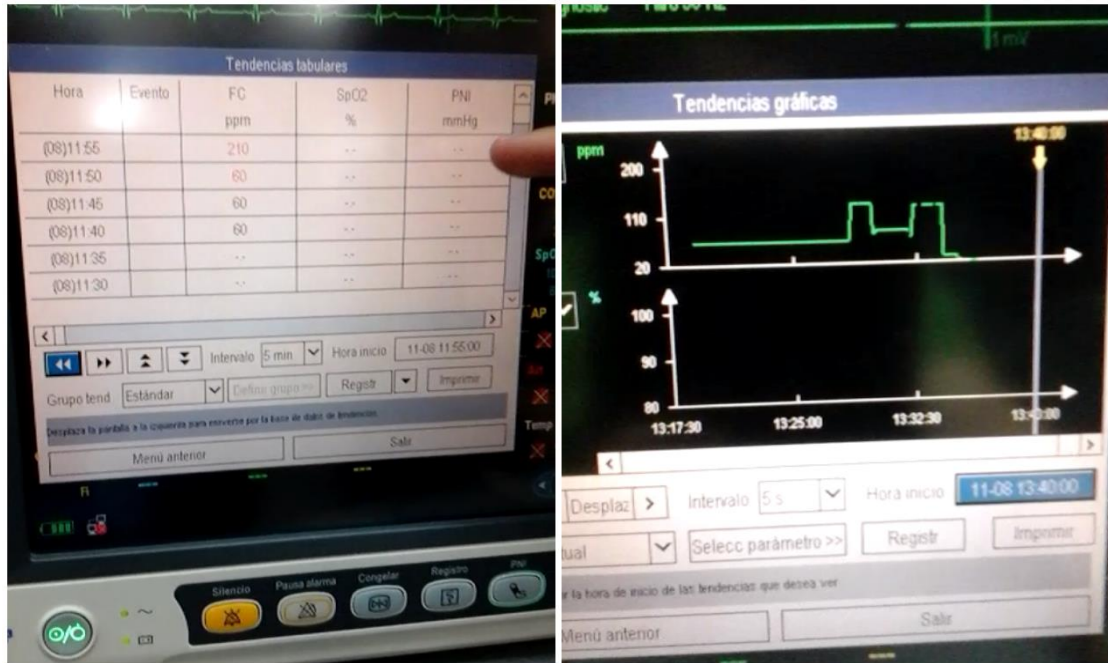


Figura 6. Pruebas en video - Sistema de tendencia gráfica y tabular de datos estáticos, guardado de eventos.

Respecto a la Figura 6 se evidenció un sistema de registro en memoria de datos del equipo prototipo, es un tema relativamente sencillo que plantea una monitorización más completa con la inclusión de guardados continuos de los datos estáticos del paciente, así como eventos peligrosos sobre un dispositivo de almacenamiento masivo, registro que puede ser visualizado a través de tablas, las cuales pueden ser representadas también en forma gráfica sobre el eje un tiempo versus escala. Por otro lado el monitor comercial posee una representación numérica y tabular sobre una serie de menús comunes para todos los parámetros que se encuentren bajo monitorización y un menú tabular para guardar los eventos peligrosos que hicieron alarmar al monitor en cuestión.

A diferencia del monitor prototipo hay ciertas falencias de velocidad en el guardado y la lectura del dispositivo de almacenamiento masivo (SD), situación que se optimiza junto con algunas falencias gráficas de la tendencia, nuevas estrategias de guardado y visualización de los eventos de manera más completa.

Estas son algunas de las muchas pruebas médicas ejecutadas, las cuales fueron analizadas antes de su implementación por el equipo de trabajo para obtener una idea

clara de lo que se realiza en el mercado y solo implementar las mejoras convenientes, necesarias e inmediatas, no saturar el equipo y retrasar los desarrollos, situación que se había venido presentando por la planeación errónea previamente ejecutada, junto al tiempo que consume la corrección de los errores que surgen por desarrollo de nuevos requerimientos.

1.2.3 Contacto con el personal médico.

Se hace notar que el lugar en donde se desarrolla esta práctica es el centro tecnológico de la Fundación Cardiovascular de Colombia ubicado en el casco urbano de Floridablanca-Santander y las pruebas que se evidenciaron en este documento respecto a manejo de monitores comerciales, se realizaron en la clínica de la misma organización; allí tuvieron lugar múltiples ensayos en los cuales el estudiante pudo tener contacto y opiniones propias del personal, esto ciertamente ayuda a centrar aún más los requerimientos y tener un enfoque claro de las percepciones del entorno medico en el cual debe su desempeño y para el cual se desarrolla el equipo médico que abarca este proyecto en general.

1.3 ESTUDIO DE TÉCNICAS DE DISEÑO Y DESARROLLO

En este punto se ha analizado la teoría necesaria para comprender el uso de este tipo de sistemas embebidos, sin embargo ha de saberse que la investigación y el estudio práctico, van de la mano para alcanzar la meta que supone este proyecto, lo cual abarca una serie de desarrollos de todo tipo por parte del estudiante. Es entonces donde se empezarán a definir los campos de trabajo y las técnicas que se utilizaran sobre los mismos.

1.3.1 Técnicas de desarrollo hardware.

Repaso en interconexión de sistemas, lenguajes de programación hardware, técnicas de sincronización, aplicación de lógica combinatorial y manejo de memoria a través de los diversos recursos presentes en la FPGA.

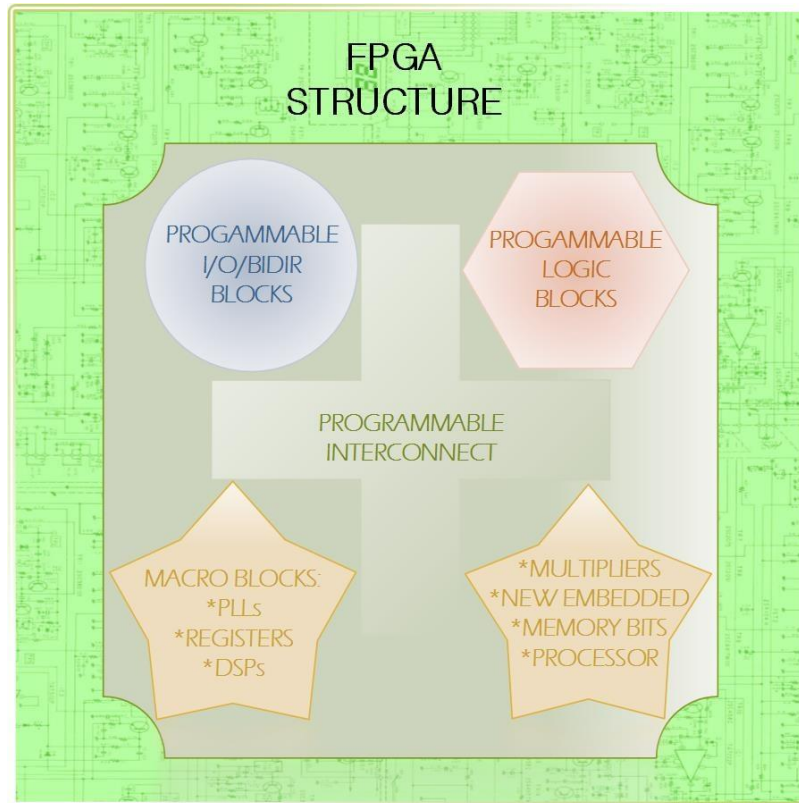


Figura 7. Lógica interna de las FPGAs

La Figura 7 representa la modularidad accesible dentro de una FPGA a través de bloques programables y motiva la investigación para un correcto manejo de estos bloques y su uso en los desarrollos que devengan del proyecto en cuestión. El manejo de estos recursos se hace crucial en la medida que casi se acaban los mismos y se requiere una optimización inmediata que además mejore la estabilidad física de los dispositivos.

1.3.1.2 Lenguajes de descripción hardware.

Con respecto a lenguajes de descripción se refiere, se decidió utilizar Verilog HDL como descriptor base de la lógica y abstracción hardware de este proyecto debido a su alto índice intuitivo que deviene a su similitud con el conocido lenguaje de programación software "código C", facilidad de manejo experimentado de forma personal desde hace ya mucho tiempo en la academia, así como su amplia gama de desarrollos globales, que sin duda alguna influyen para disminuir el costo de desarrollo en ingeniería referido al concepto de "NRE : Non Recurring Engineering".



Figura 8. Esencia del sistema de descripción hardware “Verilog”.

Como se puede observar en la Figura 8, el nivel de abstracción de este tipo de descripciones hardware es relativamente sencillo e incluye un orden bastante predecible y comparable a las diversas descripciones software a través de la implementación de subfunciones (Módulos) que agregan variadas funcionalidades al sistema embebido en general.

1.3.2 Técnicas de desarrollo software.

Una vez superados los cuestionamientos, los requerimientos y el conocimiento hardware como tal, es momento de entrar en una muy exhaustiva investigación en cuanto a técnicas de programación, abarcar por completo otros temas de procesamiento y comunicación a través del código software “C” y un tema derivado tan poco visto en el proceso de formación a lo largo de todos estos años, como lo es la programación referida a objetos “C++”, la cual abarca todo el desarrollo actual de la interfaz gráfica de usuario (GUI). Ha de aclararse que variados temas de procesamiento y comunicación, no han sido enteramente desarrollados a través del código C, sino que incluyen cierta flexibilidad del código C++ como una mezcla y aprovechamiento de ventajas y desventajas de ambas descripciones.

1.3.2.1 Problemática del desarrollo software.

Dado al bajo conocimiento del tema software más complejo (C++), se debió adoptar un conocimiento foráneo para alcanzar todas las competencias necesarias que concluyan en una formación en programación referida a objetos muy completa, que a su vez permite abordar los temas de interfaz gráfica de manera fluida en el desarrollo e implementación

de nuevos requerimientos gráficos de tipo software, como así lo implican las investigaciones realizadas a monitores comerciales, fallas, inestabilidad en el sistema, así como otros factores internos que en su momento tendrán lugar gracias a la intervención de los diferentes procesos reguladores, como lo es el área de validación y metrología de la UEE bioingeniería.

Sin duda el estudio en programación es bastante amplio y complejo dado a la inmensidad del código que debe absorber el estudiante con mucho más de cuarenta mil (40.000) líneas de código junto a aquellas que aún no se anexan, sin embargo se deben renovar primero los conocimientos en cuanto al código C base e ir abordando la parte del código híbrido (C y C++) junto a arreglos o requerimientos sencillos que vayan surgiendo por parte del tutor de acompañamiento o supervisor a cargo (jefe directo); es por ello que se ha decidido abarcar por completo los códigos C y C++ a través del conocimiento tanto local implícito en los ingenieros electrónicos asociados, como externo a través de cursos online.

El tema del estudio software es tan amplio, que debe abarcarse en paralelo con los desarrollos de ahora en adelante y se debió ser muy cuidadoso para no mezclar los nuevos conocimientos y correcciones, con errores garrafales que impliquen un daño grave al sistema software y a su vez sean imposibles de resolver o revertir, situación que tiene un plan de apoyo en la herramienta "SourceTree".

1.3.2.2 Administración de versiones software.

Antes de intervenir al tema netamente software, ha de saberse que el equipo de trabajo tiene a su disposición un administrador de versiones llamado "SourceTree" recientemente implementado por el ingeniero Jose Pablo Pinilla, el cual es una increíble herramienta de desarrollo que le da completa flexibilidad a todos los movimientos del resto del equipo de trabajo electrónico en el cambio de cualquier tipo de descripciones y modificación de archivos sin incurrir en un proceso de intercambio vía complejos servidores locales; esto cambia por completo la perspectiva para diseño y desarrollo (D&D) y propone una nueva estrategia en esta área, la cual pretende disminuir el tiempo de subida de los avances, guardados remotos y locales de cambios, paralelismo de desarrollo que incluye a todos los ingenieros implicados, así como la resolución inteligente de conflictos por línea de código o modificación de archivos que se presenten gracias a las opcionalidades que ofrece dicho administrador, el cual se implementa de manera básica mientras se explora de forma más avanzada por los implicados y con el paso del tiempo su funcionalidad suplementaria.

La herramienta "SourceTree", fue investigada, implementada y usada a través del interés común de evitar los previos y complejos procedimientos para administrar las versiones por parte de los ingenieros, esto al momento de comprimir y subir todo el material, aun para desarrollos muy efímeros, esto sin mencionar la imposibilidad de trabajar paralelamente en desarrollos completos, junto a otras mejoras que agregan este tipo de herramientas de control de versiones. "SourceTree" sin duda alguna, permite gran flexibilidad de desarrollo y aunque sus funcionalidades no se conocen muy a fondo, día a día se adquiere gran experticia en el tema y en general se transmite a todo el equipo encargado, efecto para el

cual se muestra a continuación parte de los avances adquiridos y evidenciados a través del uso de este tipo de herramientas a disposición de los interesados.

1.3.2.2.1 La realidad sobre “SourceTree”.

SourceTree es realmente una aplicación fachada de escritorio que proporciona la compañía “Atlassian”, como una herramienta con completa detección y gestión de cambios en los archivos que se encuentren dentro de un directorio asignado por el usuario, esta aplicación pretende gestionar las acciones que los ingenieros reflejan en la modificación de los diversos códigos de descripción que trabaja el equipo electrónico D&D, al tiempo que gestiona conexión de tipo cliente a través de protocolos como Git o Mercurial, hacia a los diversos servidores dispuestos con este tipo de protocolos; dado que por el momento se trabaja a través de la compañía “Atlassian” para sostener este administrador de versiones, se decidió optar por el servidor gratuito-limitado “Bitbucket” de la misma compañía, como servidor principal y remoto de los cambios realizados por el equipo de trabajo a través de los servicios prestados gratuitamente por la aplicación “SourceTree”, al tiempo que se evitan problemas de compatibilidad en los intercambios.

Ha de aclararse que para acceder gratuitamente a dicho servidor, los integrantes deben poseer cuenta en el mismo y no exceder los cinco integrantes por repositorio (Espacio de trabajo en el servidor) que esté siendo manipulado. Para hacer más énfasis y entender un poco más el funcionamiento del mismo, se plantea la Figura 9.

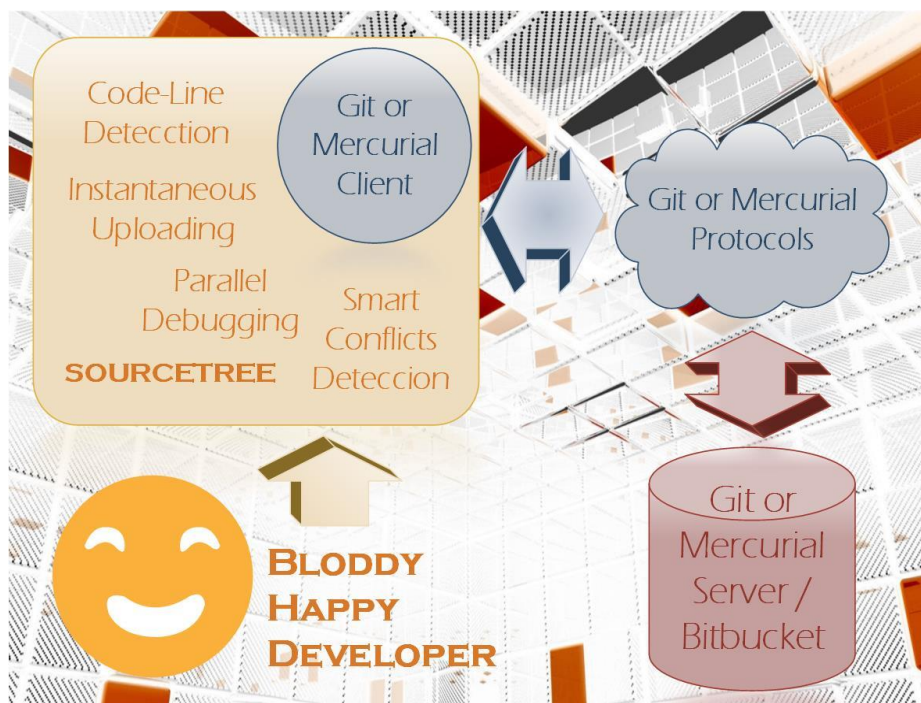


Figura 9. Estructura de la administración de versiones.

Como se puede observar en la Figura 9, todo funciona a través de SourceTree, esta es una aplicación con interfaz gráfica de usuario que proporciona relativa sencillez a todo lo que envuelve al concepto de administración de versiones, imprimiéndole ese toque de experticia a cualquier usuario alejado de la consola de comandos que cualquier usuario Mac o Windows puede tener a la mano y el cual recomiendo altamente para proyectos tanto complejos que incluyan gran cantidad de descripciones de todo tipo, como sencillos en los que se desee tener un control óptimo de los avances. SourceTree posee un cliente de tipo Git y Mercurial con los que puede acceder a los servidores respectivos que tengan este tipo de protocolos a la mano; más específicamente el protocolo Git y el servidor Bitbucket se utilizaron en este caso dado a ciertos criterios que no tendrán discusión en esta práctica.

Entrando en un contexto más real de las cosas, el desarrollador solo tendrá que preocuparse por saber utilizar SourceTree y todas sus funcionalidades (Las cuales se ponen a prueba y se utilizan cada día de trabajo), y el software se encargara de la comunicación con el servidor, sin embargo para estar un poco más claros con respecto al manejo de los espacios de trabajo o repositorios, ha de saberse que un repositorio es aquel casillero o espacio dentro del servidor que se habilita para determinar el trabajo sobre el marco de un respectivo proyecto y aunque solo pueden ingresar cinco personas por casillero, el usuario puede poseer los casilleros que desee y con una cantidad muy variada de personas de manera gratuita, mientras que estas se encuentren repartidas correctamente por estos casilleros; esta razón es completamente ventajosa para todo tipo de proyectos, sin embargo siempre cabe la opción paga que permite cada vez más usuarios y en cierto punto, el acceso ilimitado a todos los usuarios que desee el administrador de dicho repositorio, junto con soporte de uso por parte de la compañía que preste el servicio de acceso a un respectivo servidor propio.

1.3.2.2.2 Comprensión de la plataforma “SourceTree”.

Como se dijo anteriormente y con el fin de ir con a un contexto más real del uso, la herramienta luce como una aplicación cualquiera en el marco de trabajo de la plataforma/OS (Operative System) Windows, como se muestra en la Figura 10.

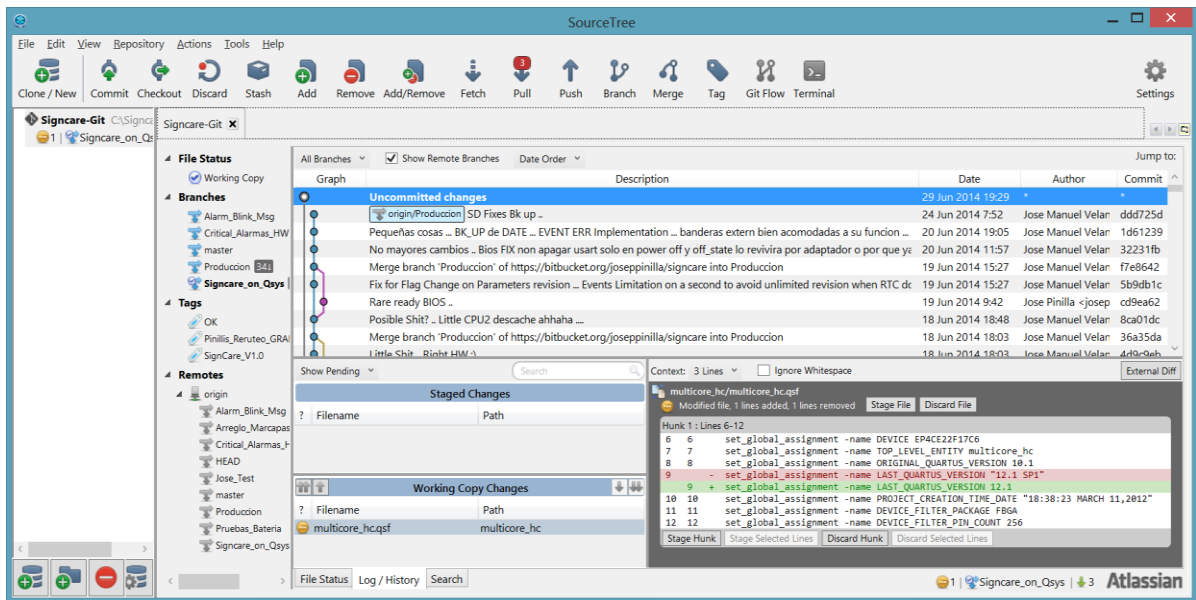


Figura 10. SourceTree como aplicación dentro del sistema operativo Windows 8.

En SourceTree, el desarrollador se va a encontrar con una aplicación, en donde se moverá constantemente a través de unas ramas que se indican, creen o modifiquen por parte del usuario, las cuales relacionan los desarrollos del grupo de trabajo a través de “Commits” o subidas de cambios al servidor, en donde cada punto de la rama, es un nuevo cambio ya subido a dicho servidor de manera remota y aunque de manera local podría lograrse, no sería accesible por los usuarios en otros computadores conectados al servidor y podría causar conflictos con otras subidas, los cuales poseen solución inteligente.

Para dar un ejemplo verificable, se puede apreciar en la Figura 10 que hubo una subida de cambios el 19 de junio de 2014 a las 9:42 de ese día por el ingeniero Jose Pinilla sobre la rama roja, luego el practicante Jose Velandia realizó una subida al servidor a las 15:27 hora militar en ese mismo día sobre la rama azul, en el mismo minuto el practicante pudo realizar un mezclado de ambos desarrollos dentro de un solo punto de la rama local azul llamada “Producción” a las 15:27 horas, sin mayor inconveniente dado a que no hubo conflictos en los cambios realizados para ambas ramas y así los cambios pudieron ser subidos remotamente guiado a través de la palabra “Origin” y más exactamente sobre la rama de “Producción” (Origin/Producción), rama que ya ha crecido en tres puntos o commits nuevos hasta la fecha y que por el momento se encontraba en el 24 de junio de 2014 en su última versión según la figura.

Por otro lado, en la Figura 10 también se puede observar que el usuario se encuentra posicionado en otra rama diferente a la rama de “Producción” llamada “Signcare_on_Qsys”, esto gracias a la información que proporciona uno de los paneles izquierdos de la aplicación; también se puede observar que se han realizado unos cambios que aún no se suben al servidor o “Uncommitted changes”, a través de una rama grisácea temporal que sobresale con un punto blanco que resalta la ubicación actual del

usuario; también se puede verificar que el usuario se encuentra atrasado tres commits de avance dentro de la rama en la que se encuentra, esto gracias a la información que brinda la flecha superior (Pull), muy probablemente debido a cambio subidos por otros usuarios o al reversar commits, para cualquier caso se puede revisar esta información a través del botón de actualización “Fetch”.

1.3.2.2.3 Detalles y opcionalidades finales en cuanto a la administración software.

El ejemplo inicial es una de las tantas y sencillas maneras de ir realizando cambios sin depender de otros, además se está tratando con una interfaz muy intuitiva y funcional alejada de las molestas líneas de comando en pantalla negra como se contempla para esta administración de versiones. Como un agregado se pudo ver en la Figura 10 lo siguiente:

- ❖ Un panel izquierdo en donde se muestra el repositorio Git actual que se está trabajando, así como una pestaña superior media que lo indica.
- ❖ Un nuevo panel izquierdo central de guía para saber si se está en una copia de trabajo, que ramas locales se poseen, que etiquetas se han heredado y los avances locales que quiso mantener el usuario al alcance.
- ❖ Herramientas superiores de subidas, bajadas y cambios de posición en cuanto a avances, dentro de las múltiples ramas que se extienden por todo el repositorio al antojo del usuario.
- ❖ Menús desplegados superiores con mayor opcionalidad para cambios del mismo repositorio, creaciones de repositorio, autenticación de la cuenta en Bitbucket entre otras cuantas opciones de red.

1.3.2.3 Solución de la problemática en base a la administración de ventanas.

Una vez abarcados una serie de ejemplos del administración de versiones que utilizará el equipo de trabajo a través de SourceTree, facilidad, posibilidad de falla y reconstrucción de avances anteriores, junto a otros factores que impulsan cada detalle del desarrollo programático, organización y de la accesibilidad grupal, se crea cierta tranquilidad de uso para continuar con la metodología del sistema software que se ha venido partiendo con el paso de la lectura.

1.3.2.4 Cursos de programación en línea.

Como se expresó en líneas anteriores, se tuvo que abordar una completa descripción de los lenguajes de programación software que se usaron como base en la comprensión a fondo de cada detalle de la extensa descripción software que se posee; en igual proporción, el estudiante no puede descuidar sus responsabilidades en la realización de estos desarrollos de tipo software, mientras evita el desgaste del apoyo prestado en cuanto a conocimientos programáticos; es por ello que a partir del par de cursos online ya

mencionados, el estudiante ha venido guiando sus conocimientos en la implementación de nuevos detalles, cambios de funcionalidades y requerimientos en general (los cuales tendrán su lugar y explicación más a fondo al final de esta práctica), en base a la redención del conocimiento recibido gratuitamente por dichos cursos, sobre las bases de una exposición verificable en el desarrollo de este documento.

1.3.2.5 Cuestionamientos previos al desarrollo (Metodología software).

Es cierto que la descripción completa de lenguajes de programación software de la que se viene hablando, se tomó tanto de los conocimientos grupales, como de los conocimientos aportados en gran mayoría por el equipo web conocido como "Wibit.net"; a través de ello se pudo absorber gran cantidad de conocimientos en cuanto a programación de todo tipo se refiere, útiles para el desempeño profesional y con ello, una comprensión global de la esencia de las descripciones funcionales de todo tipo. Es entonces donde surgen cuestionamientos básicos que debiese preguntarse todo desarrollador antes de abarcar una respectiva descripción programática, cuestionamientos de forma y de orden que le ayudarán a establecer una forma de trabajo mucho más eficiente y precavida al mismo tiempo.

❖ ¿Para qué se utiliza una descripción de tipo software?

R: El uso de un sistema de descripción software se realiza con múltiples motivos muy válidos para los sistemas, que casi siempre obedecen a requerimientos de flexibilidad de manejo y ahorro hardware de todo tipo. Sobre la premisa de plantear una respuesta verdaderamente funcional, se hará un ejemplo clave de entendimiento en donde se tiene lo siguiente:

Se plantea una situación en donde se da la necesidad de realizar cierto número de operaciones matemáticas que llegaran a un cierto resultado por ahora irrelevante; se trata de una suma, una multiplicación, una potencia y una multiplicación nuevamente.

En motivo de hacer una comparativa, lo que plantearía una lógica hardware sería optar por un módulo de suma, uno de multiplicación, uno de potencia y finalmente otro de multiplicación, en donde siempre que cada una de las entradas de los sistemas cambien, se tendrá una respuesta en la salida de todas aquellas operaciones casi inmediatamente después de dichos cambios en las entradas, pero con un gasto bárbaro de recursos hardware no reprogramables e inútiles para realizar otro tipo de funcionalidad.

Para el caso de un sistema software, se plantearía todo el problema en el uso de un único modulo sumador y esto sería suficiente, siempre sabiendo que cada operación que deba realizarse, tomará una repercusión en tiempo que se ira acumulando al realizar cada suma posible de manera secuencial sobre el sistema, esto es posible recordando como las multiplicaciones, son realmente la unión de múltiples sumas y como las potencias, son realmente la unión de múltiples multiplicaciones que se traducen igualmente en sumas.

Es allí donde el sistema software concreta su victoria sobre el sistema hardware, ya que propone utilizar el mismo módulo hardware para múltiples labores, lo cual nunca imagina el usuario final; solo basta una idea, una compilación validada de la idea a lenguaje de bajo nivel y por último una descripción funcional que podrá ser descargada y almacenada sobre una memoria tipo ROM (Read Only Memory) quien guardará dicha descripción; esta descripción podrá correr libremente sobre un respectivo procesador, quien con la ayuda de la velocidad de datos proporcionada por una memoria de tipo RAM (Random Access Memory), podrá disminuir la brecha de tiempos entre los procesos secuenciales y la percepción física del tiempo que posee el usuario, emulando así la función hardware sobre un concepto netamente software, el cual no siempre se observa exento de fallas y ralentización de los procesos en la vida real, lo que se interpreta como una molestia para el usuario, aunque no deja de ser útil y funcional para el desarrollador en la creación de sistemas interactivos, mucho más cuando se cuenta con un nivel de abstracción software de tipo medio como lo es el código C, completamente portable y compatible con las millones de descripciones software que la humanidad puede prever, evadiendo así los gastos de ingeniería que refleja el concepto de la “NRE” previamente mencionado.

❖ ¿En que difieren los múltiples lenguajes de descripción de tipo software?

R: Realmente no difieren en su finalidad, ya que todos poseen una razón de compilación a ese lenguaje de bajo nivel (ej. lenguaje ensamblador), el cual se traduce en instrucciones que ejecutaran los sistemas de forma secuencial, sin embargo cada lenguaje si posee su propio nivel de abstracción y complejidad a manera de entendimiento para el programador y el compilador para el cual fue diseñado; en otras palabras, “es una manera de hacer lo mismo pero con una estrategia diferente”, es entonces donde entra a jugar el concepto de “Metalenguaje”, el cual es usado para interpretar un lenguaje de más alto nivel conocido como “Lenguaje Objeto”; en ese orden de ideas, el lenguaje de programación o código C, java, C++ entre otros, son realmente tipos de lenguajes objeto que poseen la capacidad de ser interpretados por su respectivo metalenguaje (ensamblador) frente a una respectiva arquitectura hardware de instrucciones propias de cada fabricante de dispositivos de procesamiento integrado.

1.3.2.6 Abstracción de la dinámica software.

En medida de sostener los conceptos hasta ahora plantados en este documento, se propone la Figura 11 de función software, con el fin de resumir las varias facetas que posee la estructura software sobre un concepto más sencillo.

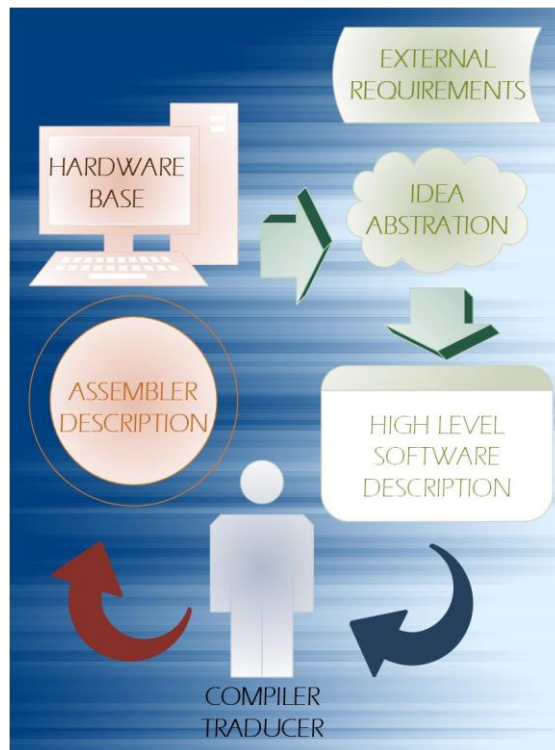


Figura 11. Ciclo de abstracción software sobre sistemas de procesamiento integrado.

Como se observa en la Figura 11, todo el desarrollo software se basa en la retroalimentación continua de nuevas ideas sobre un modelo hardware; este ciclo está abordado por diversas aplicaciones desarrolladas en la actualidad para el desarrollo sobre computadores de escritorio, que a su vez sirven como herramientas para la compilación, traducción y edición de las descripciones en su mayor parte, es entonces sobre la nube de abstracción de la idea, en donde se centra la labor específica del desarrollador encargado de cerrar el lazo, proporcionando así el pensamiento integral del sistema respecto a complejos factores externos.

Respecto al desarrollo de programación orientada a objetos C++, se plantea una abstracción de la idea un poco más estructurada a la abstracción software global propuesta, como se ejemplifica a continuación.

1.3.2.6.1 Comparativa de la dinámica entre códigos software.

El código de descripción software C está llamado a utilizar los recursos de sistemas hardware de manera más amplia y completa, con el mínimo gasto de procesamiento en comparación a otro tipo de lenguajes de más alto nivel como java, sin embargo el código C proporciona tan amplio manejo de los recursos a tan bajo nivel, que prácticamente todo debe hacerse manualmente y a través de extensas descripciones basadas en funcionalidades sencillas, que por otro lado permiten entender más a fondo la dinámica de descripción software.

El nacimiento del código C++, pretende como su nombre lo dice, sumarle o aventajar por ciertas características extra (referidas a objetos), al ya existente código C base; esto va a definir claramente un avance importante en cuanto a organización y descripción de los sistemas a través de objetos para el desarrollo sobre de la inmensa interfaz gráfica, sin embargo es un tipo de código que debe ser administrado con prudencia, ya que muchas veces se pueden pasar por alto ciertos errores que no afectan inmediatamente la estabilidad del sistema y una vez se han detectado dichas fallas de estabilidad, es muy tarde tanto para corregirlas, como para solucionarlas y es entonces donde el ciclo software previamente explicado se alarga en tiempo, comprometiendo así la calidad y la credibilidad de los desarrollos.

1.3.2.6.2 Dinámica del código C++ (Objetos).

La característica fuerte del código C++, está orientada a la funcionalidad que proporcionan los objetos, a través de las “clases” quienes se pueden describir como un tipo estructura de programación software, la cual podrá ser adoptada por el desarrollador para disminuir el costo de ingeniería en el planteamiento de múltiples y complejos algoritmos de descripción propia de objetos de interfaz. La “clase” referida a la estructura que posee el código C++, es básicamente un arreglo programático que pretende describir a través de funciones, variables y arreglos típicos del código C base, a un único objeto que pueda proporcionar múltiples actividades software sobre el mismo arreglo; este nuevo tipo de estructura difiere un poco respecto a la estructura del código C base, en la manera como se incluyen librerías, se aparta memoria para un determinado puntero o clase y otros no muy apartados de las premisas que ya posee el código C y con la ventaja de la utilización híbrida de los mismos (Uso de códigos C y C++ sobre la misma descripción software funcional y complementaria) bajo ciertos criterios que se deben respetar.

Para entender un poco la estructura de ambos códigos, se plantea la Figura 12 que define los agregados que el código C++ posee sobre el código C.

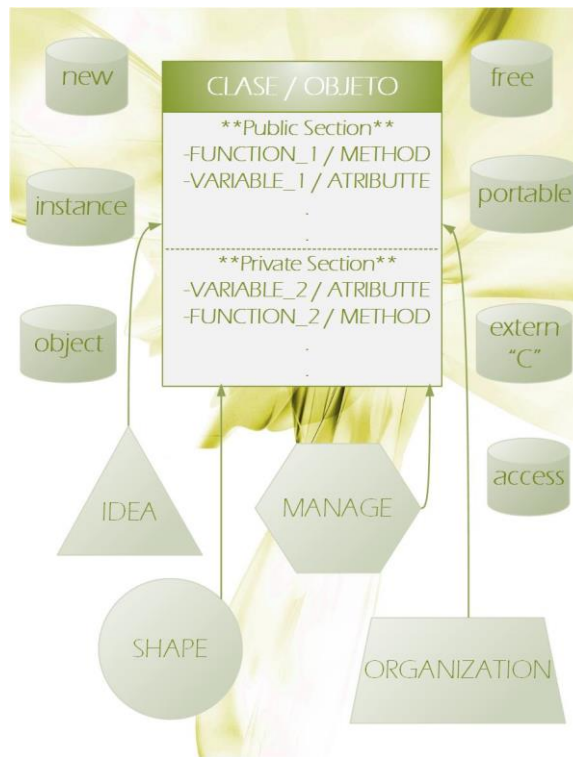


Figura 12. Estructura de la clase y conceptos propios del código C++.

Como se observa en la Figura 12, la clase se compone de métodos y atributos de todo tipo y extensión, se encierra sobre conceptos como portabilidad, instanciación, asignación de memoria, accesibilidad y es completamente moldeable a través de las ideas y los conceptos de administración, forma, organización entre otros que el desarrollador pueda incluir en la implementación de dicha estructura.

2. DESARROLLO FORMAL SOBRE LOS SISTEMAS DEFINIDOS

Ya se posee un conocimiento bastante amplio en todos los campos de aplicación no solo sistemas software y hardware, sino el campo electrónico en general dado a los diversos desarrollos previamente establecidos por los ingenieros en la organización. Esta etapa centra el estudio de esta práctica sobre los sistemas de descripción hardware de una manera muy completa en la resolución de inconvenientes que se hayan venido detectando durante todo este tiempo, así como en la implementación innovadora sobre interfaces funcionales mucho más actualizadas, nuevos entornos y por ende soporte más completo, que pretenden dar el desarrollo de los nuevos sistemas.

2.1 INCURSIÓN EN LOS ENTORNOS DE DESARROLLO

2.1.1 Entornos de desarrollo hardware sobre FPGAs.

Retomando viejos conceptos de implementación de descripción hardware, ha de recordarse como el fabricante provee las herramientas de desarrollo y el usuario provee los conocimientos que resultaran en la creación de sistemas completos; he allí que lo primero que se debe realizar antes desarrollar con la ayuda de las respectivas herramientas, se remite a un previo y completo reconocimiento de las mismas, así se evitara pérdidas de tiempo de todo tipo, así como falta de conocimientos que terminen en daños de funcionalidades al momento de la optimización que se tiene planteada para estos sistemas.

2.1.1.1 Entorno de compilación y desarrollo de la lógica programable de la FPGA.

En primer lugar se posee el entorno de desarrollo hardware para todo tipo de FPGAs y dispositivos de la compañía Altera conocido como "Quartus", este entorno permite abordar el desarrollo hardware desde múltiples alternativas como lo es el diseño modular gráfico, programático, de bloques, secuencial, etc., y posee a su vez múltiples ventanas y opciones sobre el aprovechamiento, compilación y generación de sistemas hardware que dichos dispositivos tengan para ofrecer al usuario; entorno que se aprecia a continuación en la Figura 13.

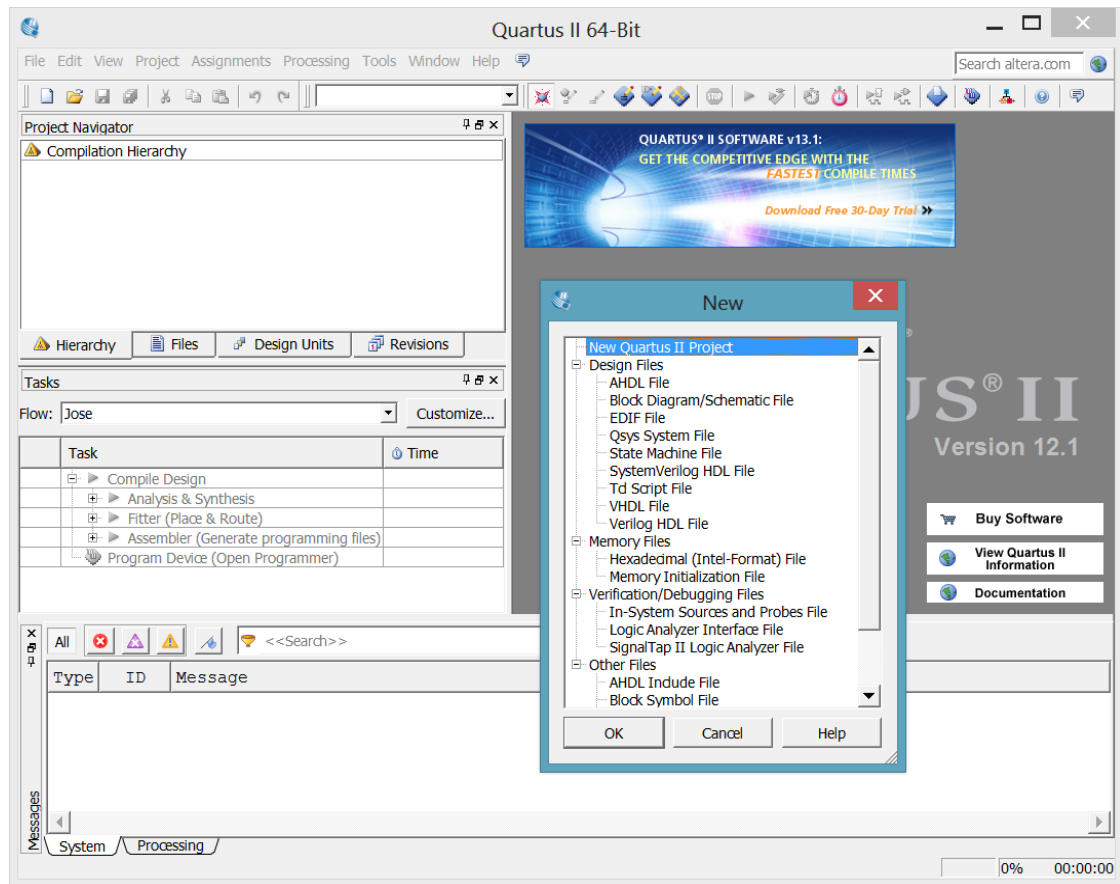


Figura 13. Quartus II, descriptor de interfaces de tipo hardware.

Por otro lado se tiene un entorno auxiliar que deriva de la plataforma “Quartus” conocido como SOPC Builder, el cual ayuda ampliamente en la creación de sistemas hardware de complejidad relativamente alta, a costo de tiempos relativamente bajos a los habituales para este tipo de sistemas; aunque esta herramienta es bastante buena y permite gran cantidad de aprovechamiento, el estudiante debe poder asimilar la información ya descrita a través de este entorno auxiliar, he implementar una sistema que implique mayor calidad partir de un nuevo entorno de desarrollo llamado “Qsys”, el cual pretende realizar la misma función que el conocido “SOPC Builder”, pero abordando cualidades de uso y descripción mucho más amplias, innovadoras y con mayor soporte por parte de la compañía Altera sobre este tipo de desarrollos, quienes a su vez han declarado la plataforma SOPC Builder como obsoleta para los años que vienen.

Tanto el viejo como el nuevo entorno de trabajo deben ser absorbidos en conocimiento, para así abordar el planteamiento de un sistema más actual, el cual proporciona grandes ventajas que aún no se alcanzan a apreciar por el usuario, pero si por el desarrollador y el equipo en general. Dichos entornos en cuestión lucen tal y como se observan en la Figura 14.

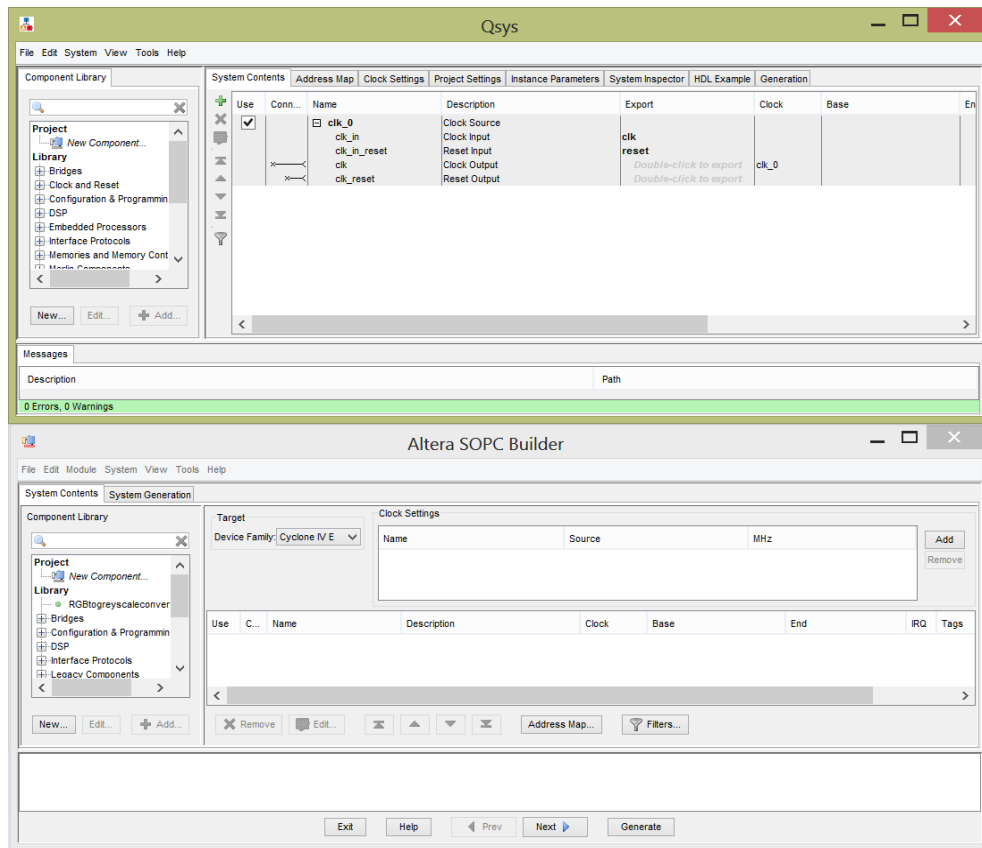


Figura 14. Entornos de desarrollo hardware Qsys y SOPC Builder

En una rápida comparativa, se puede ver que ambos entornos son muy parecidos en cuanto a su espacio de trabajo referido a un menú de componentes prediseñados, así como el panel de montaje de dichos componentes; sin embargo ha de saberse que la representación final en dicho panel, es mucho más completa y precisa en el entorno Qsys, sin mencionar el orden y pestañas con opcionalidad más claras acerca de la información del sistema (pestañas visibles en la parte superior del entorno), por lo que se hace natural e intuitivo ingresar en dichas pestañas para definir rápidamente la situación de los desarrollos, así como realizar un cambio rápido de los sistemas que lo requieran sin perder de vista factores que merecen precaución en su modificación. La mejora gradual se va a ver reflejada en el transcurso de los avances ejemplificados en este documento.

2.1.1.1.1 Compilación y abstracción del sistema hardware del MSV.

Retomando la base hardware que proporciona la plataforma Quartus II y conociendo los previos desarrollos realizados por los ingenieros, se tiene al sistema del MSV local siendo modelado en su estructura interna gracias a la modularidad programable que posee una FPGA y más específicamente, la tarjeta de desarrollo De0-nano usada en este proyecto, descrita en esencia como un módulo embebido de variados aditamentos embebidos y una FPGA de la serie Cyclone IV de la compañía Altera, quienes en conjunto ayudaran a

prototipar y generar un sistema modificable mucho más rápidamente, como lo son los módulos de memorias RAM y ROM para las bases del sistema hardware que administrara el funcionamiento del sistema software en sus estructura funcional física, módulos de video y demás módulos comunicación y realimentación interna y externa.

Los desarrollos que se describen en el párrafo anterior pueden ser evidenciados a través ciertas características que describen a los sistemas en una FPGA en cualquier caso de desarrollo típico y funcional sobre el equipo de monitorización, mas no la liberación explícita de la información dado a la confidencialidad que exige la organización.

A continuación un ejemplo de base hardware montado sobre la plataforma Quartus, en donde se alcanza apreciar sobre la pestaña “Hierarchy” una columna “Entity”, con una serie de módulos que derivan del módulo principal (multicore_hc), como subsistemas asociados que agregan funcionalidad y modularidad al sistema integrado en general y que pretenden dar a conocer la relación jerárquica de los mismos entre sí; por otro lado se refleja un panel derecho, con parte de la descripción hardware en formato Verilog HDL, junto a otros módulos aguardando en pestañas adjuntas al mismo; finalmente se muestra un nuevo panel izquierdo central con las opcionalidades de compilación hardware adoptadas para dicho diseño, así como un panel inferior que describirá los errores, precauciones e informaciones que tengan lugar en compilación a través de mensajes.

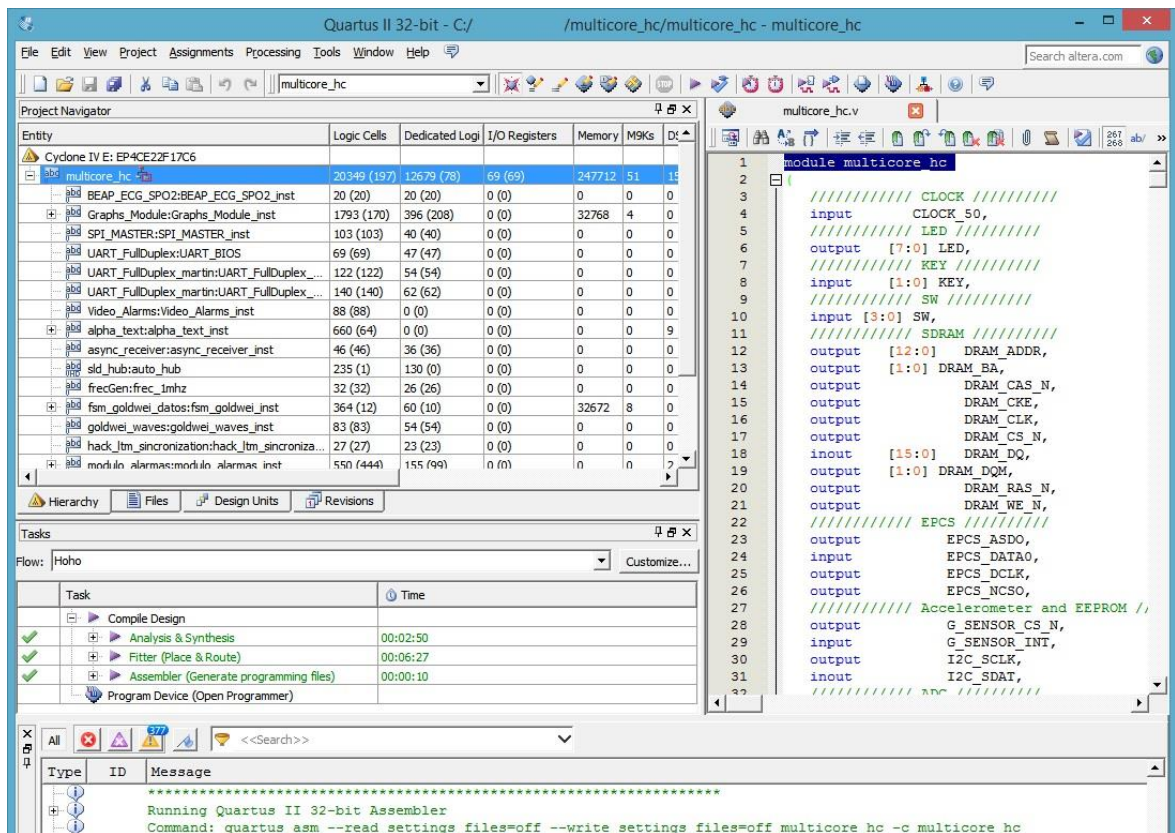


Figura 15. Ejemplo de desarrollo sobre la plataforma Quartus

En este punto se entiende claramente como el desarrollo hardware puede ser logrado a través de una serie de descripciones modulares, que serán compiladas a través de una plataforma como lo es Quartus (para los dispositivos de la compañía Altera), sin embargo ¿Qué es lo que logran todas estas descripciones, compilación y trabajo en la creación del sistema hardware base?; la respuesta a esta pregunta se encuentra en la misma característica propia de las FPGA en cuanto a su reprogramación hardware, en donde cada que se compila una nueva descripción de este estilo, se genera un nuevo sistema que reprogramará los bloques físicos y conexiones que alberga una FPGA internamente, formando así el sistema hardware que tanto se espera obtener para cada modificación. El sistema del MSV actual luce en su base hardware sobre la abstracción modular de tipo RTL (Register Transfer Level), tal y como se muestra a continuación; esto se logra a través de una de las opcionalidades de compilación que posee el entorno Quartus II.

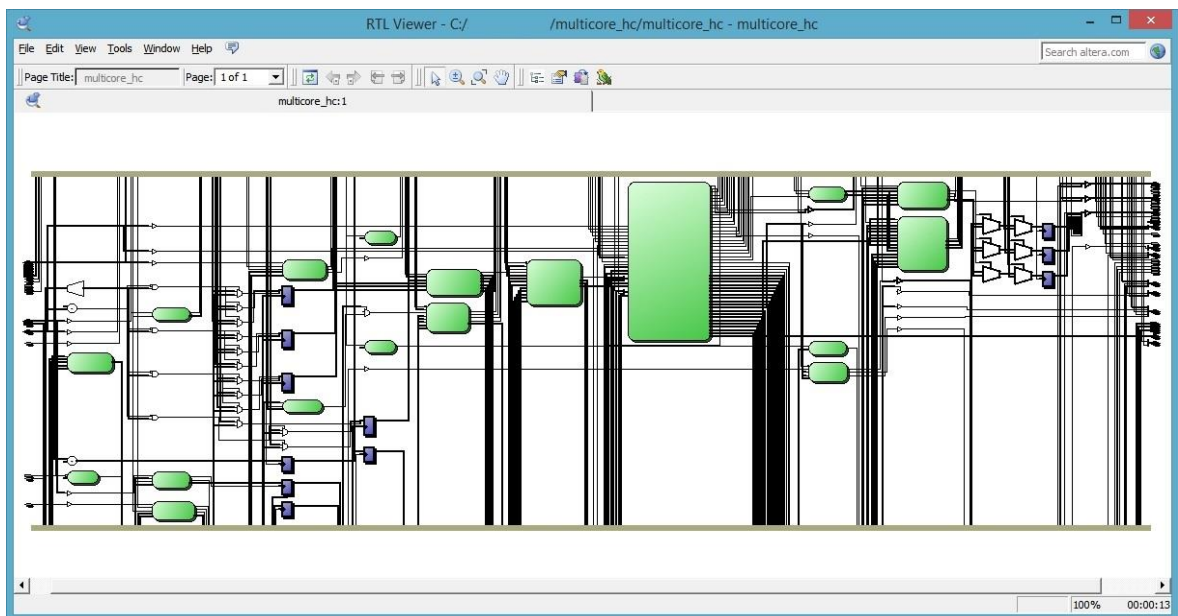


Figura 16. Sistema hardware base del equipo de monitorización en representación RTL

Claramente se observa a través de la Figura 16 como todas las descripciones modulares previamente avistadas a través de código Verilog HDL en un formato de jerarquía, tienen la función de sostener una descripción válida sobre el sistema físico que se alberga la FPGA a través de las conexiones y circuitos entre módulos como se observó en el previo esquema de tipo RTL.

Ha de notarse que los módulos de color verde soportados por una respectiva descripción de tipo Verilog HDL (archivos de extensión *.v) entre otras, no siempre se sostienen por ellos mismos, sino que contienen muchos más módulos internamente que enriquecen la funcionalidad de cada descripción con mayor número de descripciones disponibles, es decir, solo se está observando una pequeña parte de la modularidad hardware existente dentro de este proyecto; nótese también como el sistema global posee no solo módulos y conexiones, sino también interfaces de entrada y salida que entraran en contacto con el

medio físico externo a la FPGA que sea programada (Cyclone IV montada sobre la tarjeta de desarrollo De0-nano disponible comercialmente en el mercado).

Una última apreciación que impulsó el desarrollo hardware por parte del estudiante, se especifica sobre el módulo de color verde más grande que se observa en la Figura 16, el cual hace referencia al chip proporcionado por el entorno SOPC Builder, base física de funcionamiento para el sistema software, el cual se encuentra completamente descrito a través de lenguaje hardware y es allí donde corren las variadas tareas secuenciales que se han venido tratando últimamente en la resolución de múltiples requerimientos relacionados con interfaz gráfica de usuario y algoritmos software.

2.1.1.2 Entornos de desarrollo hardware de sistemas integrados.

Ya se tiene la suficiente experticia para abarcar los sistemas de reprogramación hardware de manera más fluida sobre una FPGA, es por ello que se propone la migración de las descripciones montadas a través de la plataforma SOPC Builder ya obsoleta, sobre la nueva plataforma Qsys de mayores prestaciones. Este trabajo comprende el estudio de los módulos ya implementados en la plataforma SOPC Builder como una copia referida al mismo, solo que se debió estudiar a detalle la dinámica de estos sistemas desconocidos en su mayor parte, para lograr plasmar una copia mejorada y a su vez disponer de un amplio conocimiento funcional en el momento de existir problemas típicos en el desarrollo de los sistemas embebidos que se resolvieron en su momento.

Dado a los antecedentes que evidencian la falta de conocimiento sobre el entorno Qsys, se realizan las investigaciones respectivas que guían el desarrollo de la nueva base hardware que proporciona el funcionamiento para dicho sistema software, al tiempo que se cumplen con las responsabilidades referidas a cambios de programación que deban implementarse respecto a la interfaz de usuario (GUI).

Con el fin de definir más claramente el concepto que envuelve a los entornos de integración de sistemas, como lo es Qsys en el desarrollo de nuevas tecnologías embebidas bajo descripción de tipo hardware, se plantearan las siguientes preguntas como base del conocimiento para los sistemas que están por venir:

- ❖ ¿Qué ventaja al entorno Qsys sobre otros entornos de desarrollo hardware?

Es básicamente el nivel de abstracción que posee lo que lo hace tan atractivo a este tipo de entornos de integración de sistemas, utilizando el concepto de NoC (Network on a Chip), la compañía Altera plantea la creación de sistemas hardware en minutos sobre bases de comunicación hardware de estándar industrial, en una red completamente funcional que utiliza subsistemas de la compañía misma; este concepto plantea disminuir los tiempos de diseño, verificación y por sobre todo, compilación respecto a la estructura planteada por el entorno de integración “SOPC Builder” (características que he podido evidenciar personalmente), garantizando así un desempeño mucho mayor en cuanto a velocidad, estabilidad y rendimiento.

❖ ¿Por qué se hace necesario utilizar el entorno de desarrollo Qsys?

La herramienta Qsys al igual que SOPC Builder, poseen tanto la característica como el poder de implementar sistemas embebidos a través de un par de clics, utilizando por supuesto la modularidad estandarizada y desarrollada previamente por la corporación Altera, la cual es una tremenda ventaja que ahorra tanto gastos de ingeniería “NRE”, así como fallas y errores humanos.

Una segunda razón muy verídica para el uso de este tipo de sistemas, es referida al uso de procesadores embebidos, decodificadores del color para el video con lectura directa de la memoria RAM, depuradores UART, así como los diferentes controladores de memorias para el sostenimiento del sistema software, todo lo anterior completamente descrito en lógica de FPGA y con un par de clics sobre el entorno.

En conclusión, el estilo de desarrollo Qsys, es la manera rápida de unir módulos como si fuesen unas sencillas piezas de LEGO para armar poderosos sistemas embebidos, posee sus protocolos de comunicación serial a través de interfaces de maestro-esclavo y a su vez permiten abordar rápidamente los temas de desarrollo en procesadores y por ende ampliar ciertos conocimientos sobre la estructura de los sistemas micro procesados; todo sobre la única base hardware que envuelve al concepto de las FPGA en la creación de sistemas al antojo del usuario.

Con el fin de abstraer todo el desarrollo práctico que se ha venido relatando hasta el momento, referido a las diferentes descripciones hardware que posee el sistema de monitorización vital y así poder relacionar los desarrollos a través de sus respectivas plataformas de diseño, se planteará la Figura 17 para resumir y aclarar las dudas acumuladas hasta este punto sobre diseño modular.

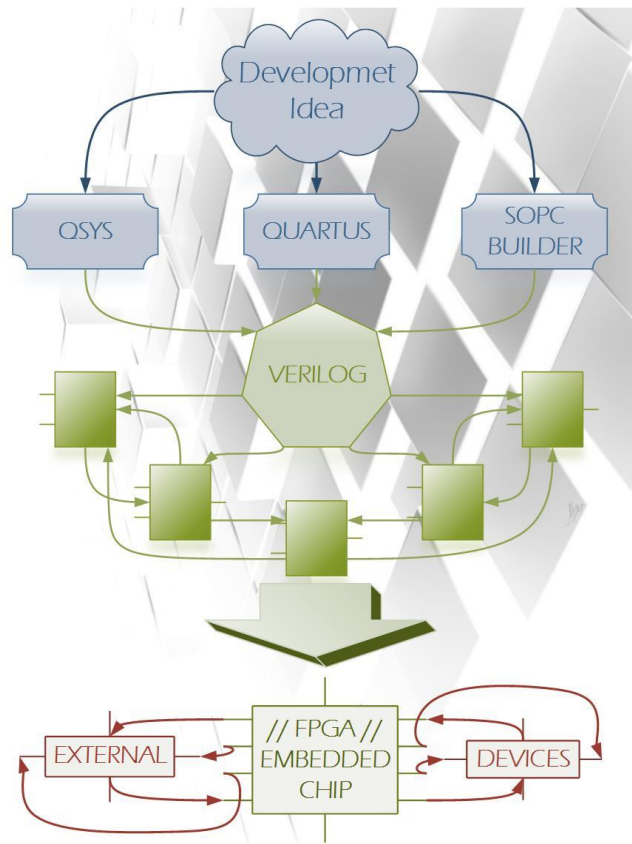


Figura 17. Diseño y desarrollo modular de sistemas embebidos basados en tecnología FPGA

Como se puede observar en la Figura 17, todo lo referente a herramientas y plataformas de diseño, tienen el fin de generar los sistemas hardware se enlazaran a través de la flexibilidad de conexión y compuertas que supone la lógica dentro de una FPGA y que a su vez actúan como esqueleto de los sistemas de adquisición, sistemas de audio, sistemas de video, sistemas de ejecución secuencial de tareas (software) entre otros.

2.1.1.2.1 Desarrollo sobre entornos de sistemas integrados (Qsys).

En calidad de evidenciar los desarrollos, se pretende ejemplificar como sucede exactamente el diseño de estos sistemas “LEGO” completamente automatizado y simple, en la creación de chips embebidos que se implantaron sobre la FPGA.

2.1.1.2.1.1 Organización del espacio de trabajo.

Esta etapa centra el trabajo del desarrollador en la definición del entorno de trabajo y sus funcionalidades, archivos fuente, abstracción, topología de diseño y adecuación de los recursos disponibles sobre la FPGA.

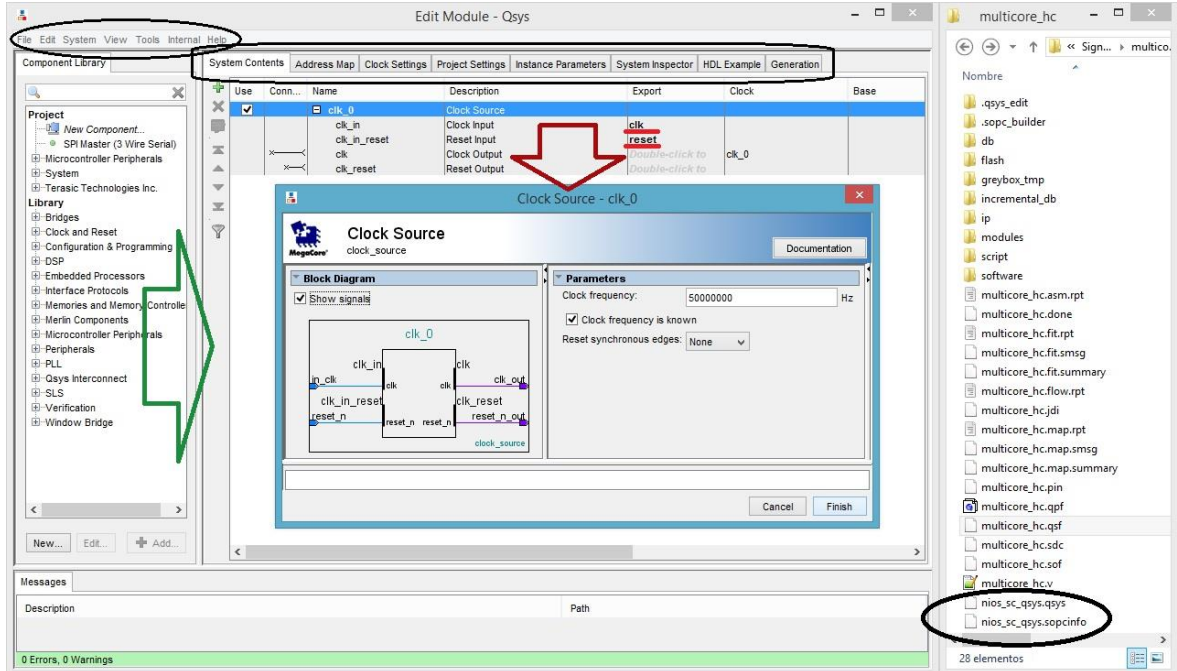


Figura 18. Adecuación del entorno Qsys, detalles base de la herramienta

El entorno Qsys al igual que el entorno SOPC Builder y muchos otros entornos de desarrollo (Hardware o Software), se centran en una interfaz gráfica con opcionalidades y menús de todo tipo, para lograr abstraer la información allí plasmada por el usuario desarrollador a términos locales de compilación o traducción a lenguaje de máquinas. Qsys no es la excepción a este estándar comúnmente adoptado a través de menús atractivos e intuitivos con el fin de ser más comerciales, justo como se pretende hacer en la relación usuario-máquina para el monitor de desarrollo.

Como se aprecia en las carpetas y archivos de la Figura 18 (base hardware del proyecto sobre el entorno Quartus II de nombre "multicore_hc"), se resaltan dos archivos importantes para el desarrollo que se efectúa sobre el entorno Qsys, tanto el archivo descriptor de implementación en dicha interfaz (extensión *.qsys), como el archivo funcional de descripción física del chip (extensión *.sopcinfo), el cual utiliza el entorno de desarrollo software (Nios II Eclipse) para adquirir información del hardware que podrá programar a futuro; cabe aclarar que el archivo "*.sopcinfo" será creado únicamente al ejecutarse la compilación sobre el entorno Quartus II como una abstracción física del módulo respectivo previa descripción del archivo "*.qsys".

Qsys es básicamente un entorno que proporciona una pestaña de "System Contents" en donde se pueden enlazar múltiples módulos hardware en la creación de sistemas embebidos en una FPGA; dichos módulos pueden ser obtenidos a través del panel izquierdo conocido como "Component Library", de igual manera el usuario puede crear sus propios componentes hardware en base a estándares de comunicación previstos por

la compañía Altera, a través de una serie de menús que guiaran al usuario en la creación de dicho componente al clicar sobre la opción “new” de dicho panel, proceso que puede requerir un estudio a profundidad en base a manuales de la compañía Altera en la familiarización de estándares de comunicación e integración de sistemas de forma serializada.

Como se observa en la Figura 18, existe un módulo base de reloj (Clock Source) y su respectivo menú de configuración proporcionado por el entorno; en este se puede ver la estructura física del módulo (Block Diagram), así como su opcionalidad de frecuencia y forma de reinicio; generalmente este es un módulo puente entre el exterior del chip hacia el interior del mismo para poder facilitar el suministro de reloj a todo el sistema que está siendo descrito si así se desea, el valor de la frecuencia se da como un extra para que el compilador pueda acomodar los recursos adecuados o si existe un módulo de fase PLL quien necesite de esta información, al igual que los archivos descriptores de este sistema. También se puede observar como cada conexión final del módulo ejemplificado a través de “Block Diagram”, puede ser perfectamente exportada fuera de todo el sistema que se está describiendo si se desea bajo la acción de un par de clics sobre la columna “Export” de dicho modulo, el cual siempre estará empujado sobre la pestaña de “System Contents” como un LEGO más sobre la estructura del nuevo chip; este módulo de reloj no solo se puede exportar y configurar, sino que debe atender ciertos requerimientos de diseño, ubicación en el sistema, conexión y comunicación, entre otros conceptos a abarcar en el transcurso de los avances.

Ya para finalizar con la organización del espacio de trabajo, se pueden observar unas pestañas superiores que también resultan importantes para describir y ejemplificar el sistema, la mayoría son de lectura y solo describen en forma más organizada ciertas características en base a la descripción de la pestaña “System Contents”, sin embargo otras como “Address Map”, “Clock Settings” y “Project Settings” servirán para cambiar características muy generales del mismo; otras como “HDL Example” y “Generación” se hacen relativamente opcionales y solo tienen sentido al final del proceso de diseño. Otras opcionalidades se ejemplifican como la mayoría de entornos a través de menús superiores de edición, así como un panel inferior de depuración para mensajes con impacto sobre los cambios humanos que se vayan realizando.

2.1.1.2.1.2 Armado del sistema.

Ya teniendo los conocimientos previos en el manejo del entorno, resta empezar a armar los “LEGO” sobre la idea de diseño del chip de procesamiento software, integrando los diferentes módulos de video y adquisición de señales entre ambos mundos.

El sistema que se ejemplificará a continuación, se hace bastante extenso, sin embargo logra conexiones relativamente sencillas e intuitivas para el usuario sobre una interfaz gráfica llamada Qsys, interacción que se ira absorbiendo poco a poco bajo la abstracción LEGO que se viene adoptando.

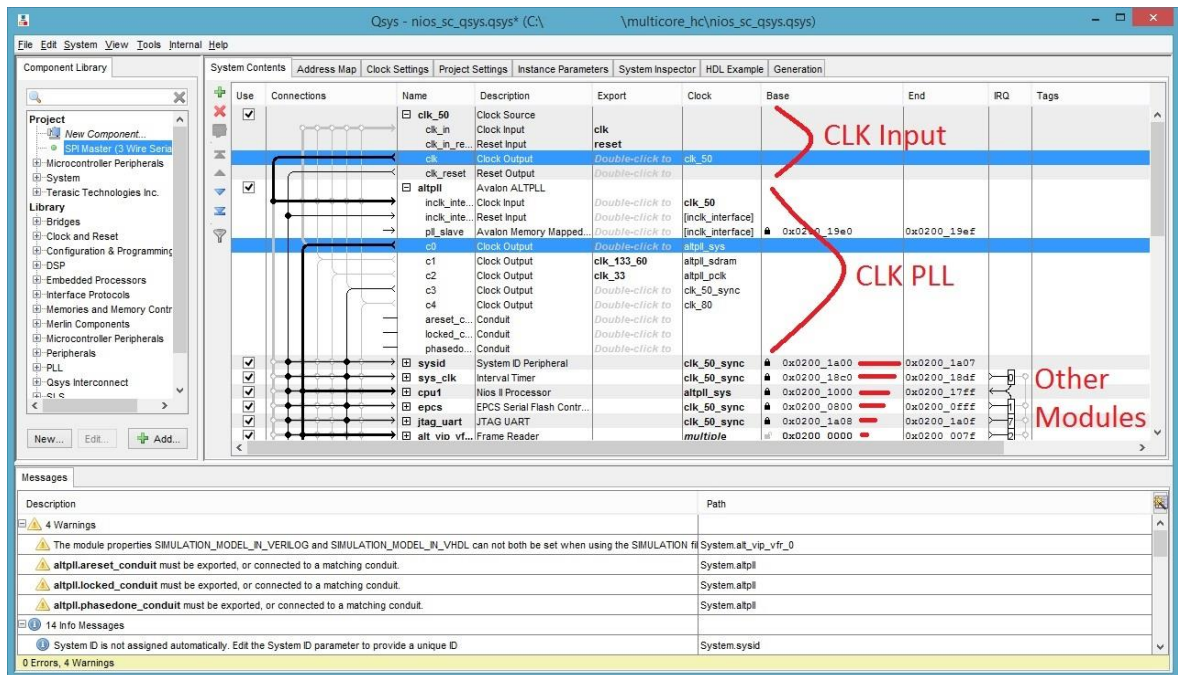


Figura 19. Descripción gráfica “Qsys” de un chip de desarrollo sobre FPGA.

Primero que todo se puede ver en la Figura 19, como un par de módulos (clk_50 y altpll) se encuentran expuestos en su conexión, cada uno con una función específica, el primer módulo ya fue ejemplificado anteriormente como un puente del hardware a la lógica externa al chip que se está diseñando, este módulo de Clock Source bajo el nombre de “clk_50”, posee una señal de reloj proveniente de la tarjeta de desarrollo (De0-nano) con la ayuda de su interfaz de conexión “Clock Input” y una rápida exportación de la misma, la cual será expuesta a unos pequeños arreglos dentro del módulo y finalmente será puesta a disposición del usuario a través de la interfaz de conexión “Clock Output”; esta dinámica de funcionamiento entradas-salidas, se reflejara por todo el sistema.

Al igual que este último módulo toma su reloj del exterior, el segundo módulo de Avalon ALTPLL bajo el nombre de “altpll”, toma su reloj (Clock input) de la interfaz de conexión (Clock Output) ofrecida por el primer módulo sin que salga de la descripción Qsys, esto se logra con un sencillo clic en la unión de ambas interfaces (Clock Output a Clock Input) sobre la columna “Connections” de la pestaña “System Contents” tal y como se ejemplifica en la Figura 19, este último módulo se encargará de generar los diferentes relojes a disposición de todo el sistema (c0, c3), al tiempo que exporta algunos necesarios para la lógica externa del chip (c1, c2,); el entorno tiene la posibilidad de detectar los relojes disponibles por su tipo de interfaz de conexión y serán ubicados rápidamente sobre la pestaña “Clock Settings”, en la cual se podrán ver características y cambiar nombres a conveniencia del desarrollador; cabe resaltar la importancia de los relojes sobre sistemas de comunicación entre las diversas interfaces y módulos representados a lo largo del entorno, módulos que igualmente toman su señal (Clock Input) gracias a la función de “altpll”, módulos que por el momento se encuentran minimizados (No explícitos) y solo se refleja una línea grisácea con su respectivo nombre y otras características sobre el

entorno Qsys ejemplificado en la Figura 19.

Ha de aclararse que no solo hay módulos de reloj en la librería de componentes, existen módulos hardware de todo tipo y función para una determinada labor y uno de los más importantes es referido al procesador que correrá el sistema software del equipo médico, procesador de propiedad intelectual de la compañía Altera conocido por la comunidad en general bajo el nombre “Nios II”, aunque sea un procesador descrito en base a lógica de FPGA y no pueda ser exceder su velocidad de funcionamiento a más de unos 160 megahercios referente al modelo de la FPGA utilizada (Algunas hasta 300 MHz), es un procesador muy capaz y compatible que se ayuda del paralelismo a su disposición.

2.1.1.2.1.3 Interconexión explícita de los sistemas maestro-esclavo.

Esta etapa define más claramente las órdenes y jerarquías de los módulos entre sí, también introduce al estudiante en los diferentes estándares serializados de comunicación y la relación de la programación secuencial software respecto al sistema embebido; lo anterior a través del uso de hardware especializado (Memorias externas y macro bloques) que provee tanto la tarjeta de desarrollo (De0-nano) como la FPGA misma.

El procesador Nios II es uno de los módulos más complejos y delicados, dado a su función y características de comunicación e interconexión, sin embargo el entorno Qsys hace que todo parezca más fácil de lo que realmente es, solo basta generar un par de conexiones y todo podrá funcionar correctamente.

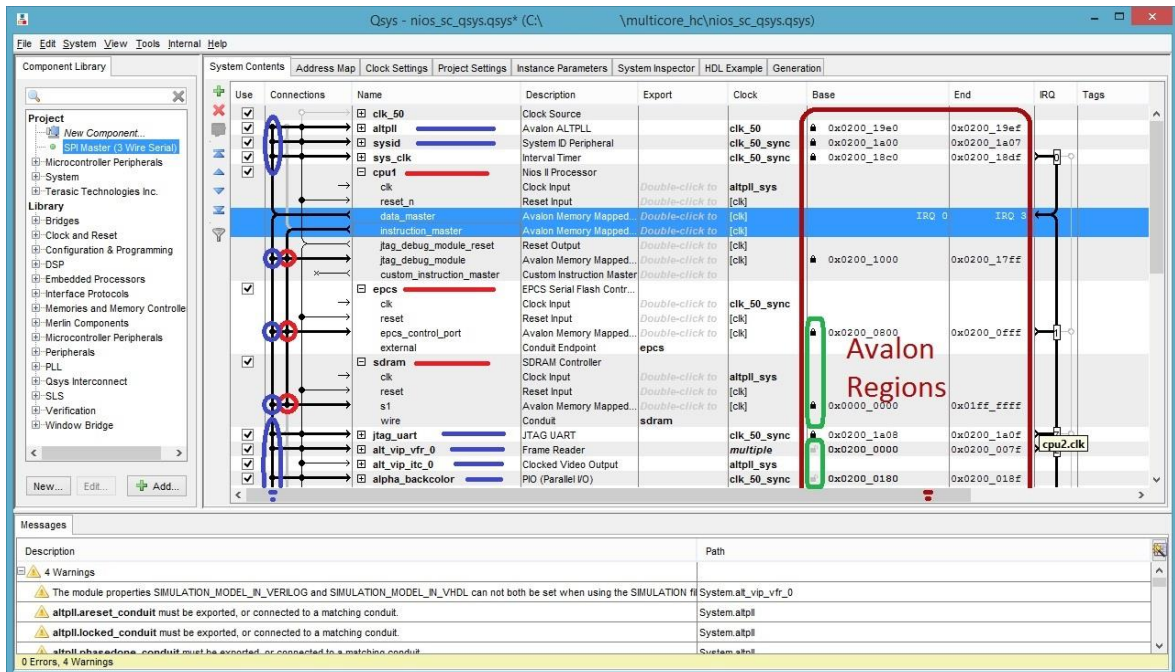


Figura 20. Estructura de conexión sobre el procesador Nios II.

Como se puede observar en la Figura 20 y aún sin saber cómo sucede la interacción entre un procesador y sus respectivas memorias (tanto volátil de uso veloz “RAM/SDRAM” y no volátil de carga software “Flash/EPCS”), se sabe que el procesador cuenta con dos hilos de comunicación maestro de tipo “Avalon” (data e instruction master) que enlazaran a todo el sistema como uno solo.

Respecto al hilo de instrucción maestro, se puede decir que está dedicado únicamente a correr labores de ejecución de la programación software y solo enlaza los dispositivos que intervengan en esta dinámica esto se da por la misma topología de estos sistemas, en donde un procesador no posee memoria flash ni memoria RAM propias en su estructura y se da la necesidad de especificar la conexión e internamente la descripción los espacios de memoria a utilizar y así no interferir con otros procesos que puedan ser aprovechados por otros procesos de forma paralela. Respecto al hilo de datos maestro se puede decir que es el que proporciona real interacción con el sistema embebido en general y enlazará la diversidad de datos que el sistema final tenga disponible para que sean procesados por el Nios II, es por esto que dicho maestro debe ser conectado a cada módulo con el que se quiera poseer una interacción casual y no olvidar definir un número de interrupción sobre la columna “IRQ”, para aquellos módulos que la usen, de otro modo habrá problemas en compilación póstuma de software dado al olvido de las mismas.

Otro concepto referente a la ubicación debe ser abordado, ya que como se explicó anteriormente, pueden existir múltiples conexiones de los maestros previamente descritos, sin embargo ¿Cómo sabe el procesador a qué módulo escribir?, es allí en donde cada módulo alberga una dirección “Avalon” relativa, la cual describirá su posición y rango de escritura y lectura disponibles para que el procesador pueda manejar mientras corre su respectivo software. Ha de aclararse que cada maestro Avalon concede comunicación con sus esclavos interconectados y no se relaciona con ningún otro maestro, esta es la razón de que cada maestro existente en esta descripción (por ahora gráfica), este representando una serie de direcciones relativas a cada hilo maestro y no absolutas a todo el sistema, sin embargo cada módulo esclavo solo puede poseer una determinada región avalon, la cual debe ser respetada cuando más de un maestro es conectado al mismo esclavo, es en este punto en donde se hace conveniente fijar las bases de algunos módulos desde el comienzo del proceso de descripción en valores que no intervengan con las direcciones relativas de otros maestros, como sucedería en el caso de existir dos procesadores Nios II (Maestros) en el sistema, que quieran acceder a un mismo esclavo; esta fijación de direcciones se logra a través de la opcionalidad que proporcionan los candados de la columna “Base” en la pestaña “System Contents” ejemplificados sobre la Figura 20.

Otra manera de modificar y corregir las direcciones avalon de forma más ordenada y rápidamente editable, es a través de la función que proporciona la pestaña “Address Map”, la cual se ejemplifica sobre la Figura 21 y en la que se pueden entender más fácilmente los conceptos previamente explicados.

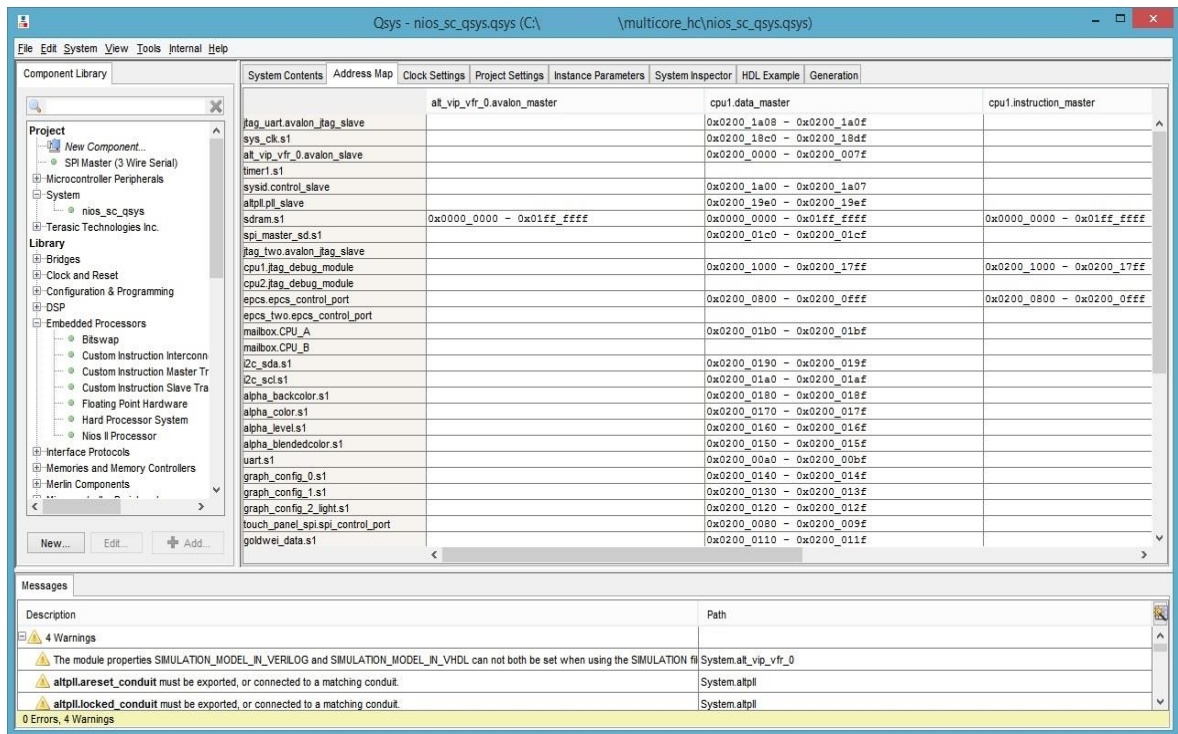


Figura 21. Mapa de direcciones relativas Avalon.

2.1.1.2.1.4 Revisiones, instanciación y compilación del chip final.

Esta etapa abarca varias revisiones y compilaciones fallidas del chip desarrollado a través de la plataforma Qsys, como un proceso de ensayo y error en cuanto a algunas falencias de conexión, generación incompleta del sistema y una serie de evidencias que surgen por desconocimiento u o error que proporciona el factor humano.

Esta etapa surge dado a la necesidad de depurar pequeños errores y tener en cuenta ciertas opciones que facilitan la edición y generación final sobre el entorno Qsys.

Respecto a diseño de un chip sobre el entorno Qsys se refiere, una cantidad importante de conexiones, asignaciones de bases y otros conceptos previamente explicados que deban ejecutarse, se hace bastante útil conocer las opciones del menú “System” (Figura 22), en la generación de conexiones monótonas y resolución de cruces de direcciones avalon, entre otras.

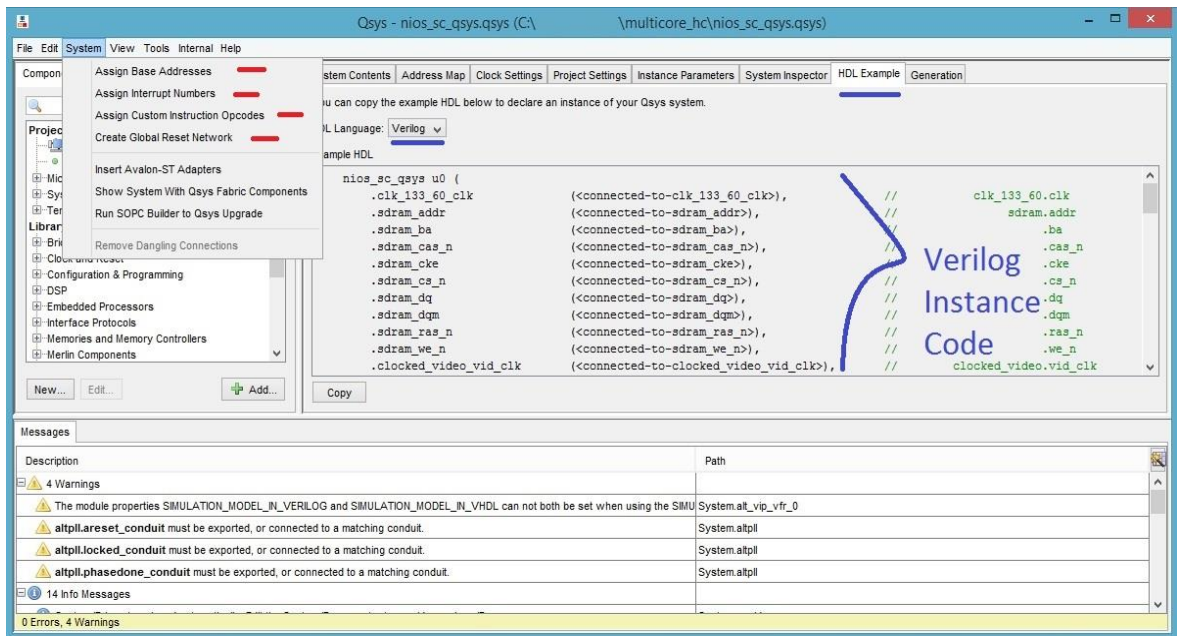


Figura 22. Opciones funcionales de edición e Instanciación final sobre el entorno Qsys

Como primera instancia se tiene la opción de asignación automática de direcciones base, la cual aplica y ordena las direcciones base de los módulos existentes hasta el momento en busca de resolver solapamientos de las mismas, sin embargo no actúa sobre módulos que posean direcciones fijadas con la opción de candado previamente descrita, para lo cual el desarrollador debe estar atento a que si persiste la falla de bases podría ser por una mala fijación de dichas direcciones.

Como segunda opción se tiene la elección de números de interrupción aleatorios, la cual se aplica a los campos vacíos de interrupción que para el usuario no sean relevantes y puedan ser fijados al azar, aunque se recomienda que el desarrollador sea consciente de estos números para evitar problemas de rendimiento en labores que no lo merezcan.

Aunque la opción de creación de “Opcodes” personalizados no está muy clara en funcionamiento y debiese leer un par de manuales para su entendimiento, no estoy interesado en crear códigos de operación para los procesadores de manera especializada sino que permanezca con sus códigos de fábrica junto a su soporte de librerías; sin embargo en cuanto a la última opción en la creación de redes de reseteo global, se podría decir que debiese ser la más usada en casos donde el usuario no necesita que un módulo A sea reiniciado por algún otro modulo B en específico (ya que ahorra inmensa cantidad de tiempo uniendo puntos de interconexión hacia interfaces de tipo “Reset Input”); solo basta clicar esta opción automática para que se junten todas las interfaces de tipo “Reset Input” con interfaces tipo “Reset Output” y eso será todo con respecto a las uniones de reseteo. Cabe aclarar que no hay manera de evadir las conexiones de reloj, maestro-esclavo, exportaciones fuera del chip y configuración de cada módulo por separado.

Una vez se han realizado todas las verificaciones del caso y el diseño del chip esta gráficamente listo (descrito) sin errores en el panel de mensajes, solo resta la etapa final de compilación y generación sobre la base hardware del equipo, que para este caso se hará de tipo Verilog HDL como se aprecia en la Figura 22, en donde solo baste copiar y pegar esta instanciación en alguna parte del código implementado sobre el entorno Quartus, agregar el archivo de extensión `“.qsys”` al proyecto hardware para que así el entorno reconozca esta instanciación, asignar algunos cables, juntar correctamente un par de conexiones sobre el lenguaje Verilog HDL y el proyecto hardware estará listo para compilar y funcionar a través de una FPGA.

En este punto cabe aclarar que aunque el chip o módulo descrito previamente a través del entorno Qsys este compuesto de muchos otros módulos y descripciones hardware avistadas sobre la pestaña `“System Contents”`, además de otros sub-módulos que no se logran avistar, el archivo de extensión `“.qsys”` bastará para que el proyecto pueda analizar y generar todos estos archivos de forma temporal en las carpetas `“db”` y `“incremental_db”` (Figura 18) sobre el proceso de compilación, ya que los sistemas Qsys no necesitan generación extra de ningún tipo como si sucede en la plataforma SOPC; esta generación se puede realizar de manera opcional en la plataforma Qsys si se desea la equivalente descripción de hardware en formato Verilog HDL u otros para fines educativos. Por ultimo resta esperar la compilación del archivo de extensión `“.sopcinfo”` en la generación póstuma del sistema de archivos software que programaran los procesadores que hubiesen en la descripción Qsys.

Dado a lo anterior, es bueno destacar que el aspecto de generación y compilación que poseen los chips descritos desde el entorno Qsys es muy superior a la descripción del entorno SOPC Builder, ya que nunca se hace necesario acceder a la pestaña `“Generation”` del entorno para crear módulos Verilog HDL de forma lenta y estorbosa; de cualquier manera si el usuario desea efectuar esta generación de archivos, esta se hará sobre una única carpeta del mismo nombre adoptado para el chip diseñado en Qsys sin mayor desorden visual, más nunca va a encontrar una cadena tan desordenada de archivos sobre la carpeta principal del proyecto como si lo hace el entorno SOPC Builder, la cual se ejemplifica en la Figura 23.

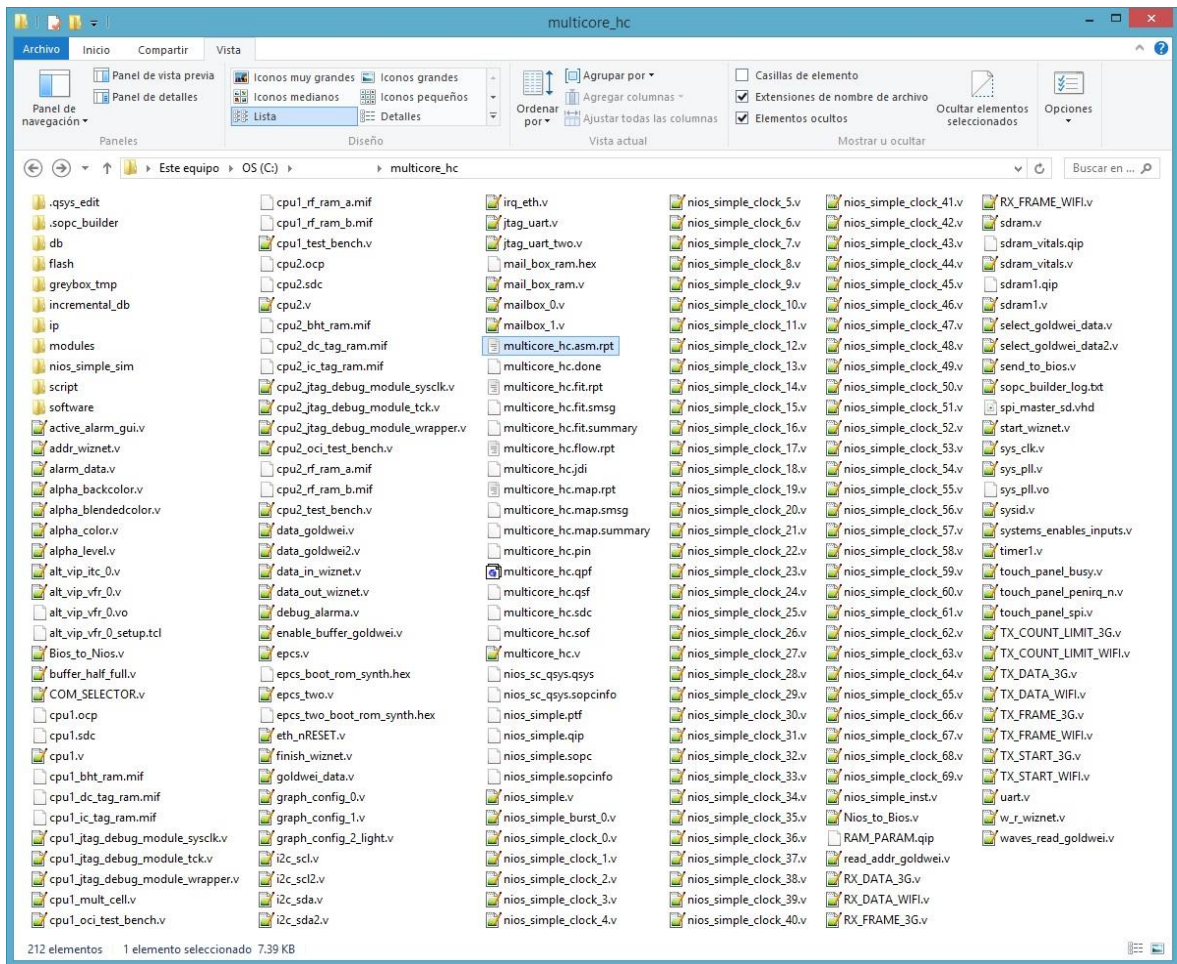


Figura 23. Generación forzosa de archivos por parte del entorno SOPC Builder

2.1.2 Introducción a entornos de desarrollo de software.

Ya se ha implementado todo lo referente a lógica hardware y chips embebidos a través de los diseños dispuestos para este proyecto, sea entonces el momento de darle paso a la funcionalidad de ejecución software sobre la base de programación de los procesadores dispuestos (previo desarrollo sobre la plataforma Qsys), pero antes se debe adquirir cierto nivel de conocimiento respecto a los entornos software que se tienen a disposición.

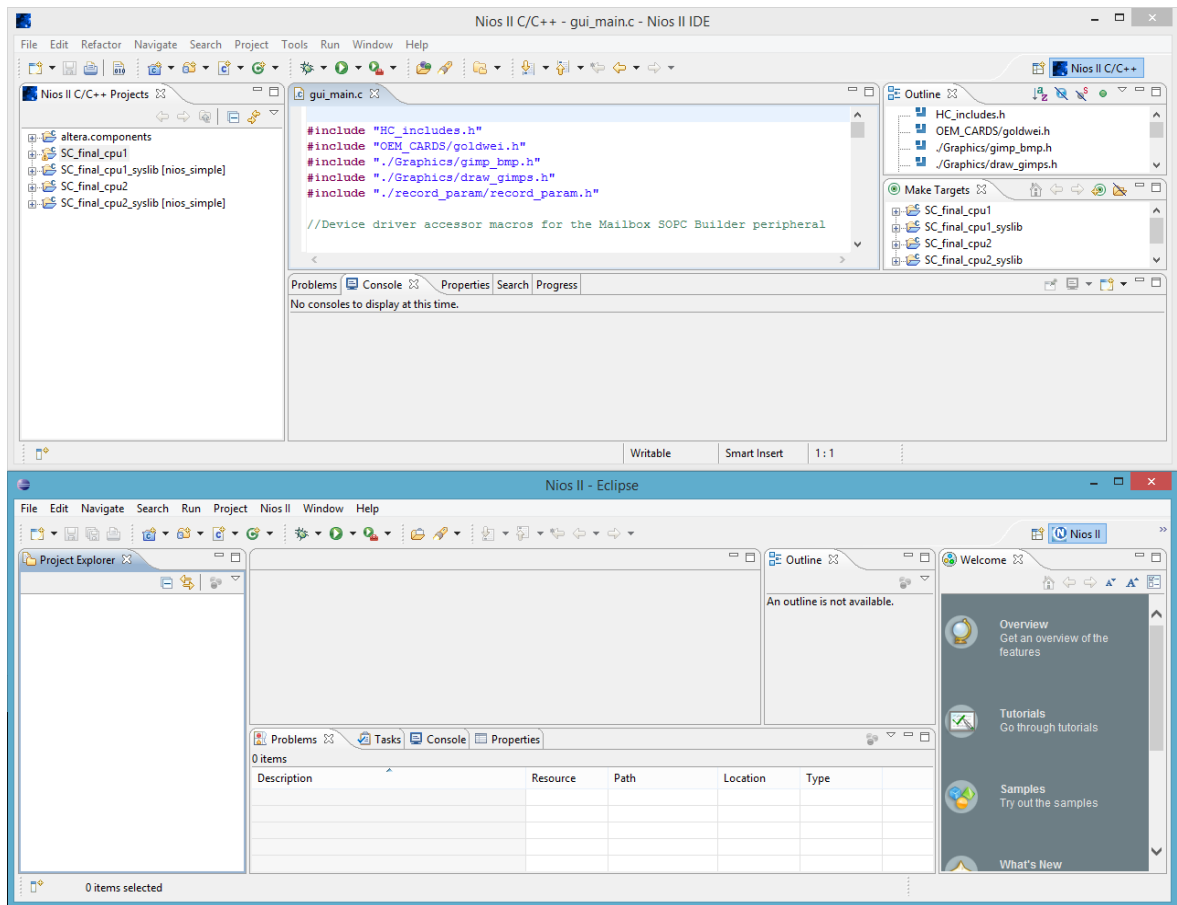


Figura 24. Entornos “Java” de desarrollo software, sobre procesadores (Nios II) basados en tecnología de la corporación Altera

En la Figura 24 se observan los dos entornos que se usan por el equipo de trabajo electrónico en el área de diseño y desarrollo; el primero (Nios II IDE) está referido a un entorno que se usa para las múltiples compilaciones de los sistemas generados a través del entorno SOPC Builder y el segundo (Nios II Eclipse) está referido mayormente al desarrollo sobre la abstracción que proporciona Qsys, esto se da por la misma forma en que cada entorno de la Figura 24 proporciona sus propias opciones en la creación de proyectos software para procesadores embebidos.

El entorno “Nios II IDE”, siempre solicitará el archivo de extensión “*.ptf” para abstraer los datos del chip que se describe en el mismo y así generar las librerías software del sistema con las cuales compilar la respectiva programación, archivo que por su tipo de extensión, se asocia más comúnmente con los ficheros obligatoriamente generados por el entorno “SOPC Builder”; por otro lado el entorno “Nios II Eclipse”, solicita siempre el archivo de extensión “*.sopcinfo” para describir el sistema hardware base y del cual generar las librerías software para la compilación de tal programación en procesadores; archivo que se ve más relacionado con los ficheros generados por el entorno Qsys, aunque la extensión “*.sopcinfo”, es un tipo de archivo de descripción hardware que también es

generado de la abstracción sobre la plataforma SOPC Builder y por ende el desarrollo sobre esta plataforma también puede ser adoptado por el entorno “Nios II Eclipse” para la compilación software en base a esta plataforma hardware.

2.1.3 Edición software alejada de los entornos dispuestos.

Al igual que el entorno de desarrollo hardware “SOPC Builder” posee un entorno de desarrollo software como lo es Nios II IDE y el entorno de diseño hardware “Qsys” posee el entorno de desarrollo software “Nios II Eclipse”, hay que definir cuál es el que proporciona mayor flexibilidad de edición respecto a búsqueda, reemplazos y complejidad de la misma y aunque la respuesta a este cuestionamiento siempre será “Nios II Eclipse”, no tiene relevancia como editor en este momento, ya que los archivos fuente software de este proyecto tienen una mayor calidad de edición a través de una aplicación más especializada llamada “Qt Creator”; ahora respecto a compilación se refiere, cabe decir que ambos entornos son muy parecidos y veloces, excepto cuando existe algún cambio de tipo hardware, en donde al existir una regeneración del sistema, la plataforma de compilación software “Nios II IDE” da pie a varios minutos extra de compilación, en comparación al entorno “Nios II Eclipse”, el cual es mucho más rápido y funcional, sin embargo posee falencias en el reconocimiento automático de modificaciones externas y es por ello que el desarrollador no debe olvidar regenerar los índices a través de un par de clics dentro del entorno, lo cual pueden ser un poco molesto o engorroso para algunos.

2.2 ELIGIENDO EL SISTEMA OPERATIVO DEL PROYECTO SOFTWARE

Realmente no fue un completo inconveniente definir el sistema operativo que iba a funcionar sobre esta plataforma embebida, ya que en la creación del proyecto, ambos entornos poseen la capacidad de adjuntar las respectivas librerías software de un único sistema operativo conocido como “uC/OS II” de la compañía Micrium, el cual proporcionará ciertas funcionalidades y nuevas estrategias de acción para el proyecto en general.

Antes de empezar a usar de dicho software, había que definir si dichas librerías conocidas como el sistema operativo “uC/OS II”, eran la mejor herramienta de trabajo para este tipo de proyectos, sin embargo, los resultados de la investigación no tomaron ni mucho tiempo, ni muchas páginas de investigación, ya que desde el comienzo se obtuvieron varios comentarios muy buenos por parte de la comunidad en general y para quienes aún no lo saben, este software está aprobado por la agencia americana de regulaciones conocida como la “FDA” para su utilización fiable sobre equipos médicos; estudios y múltiples pruebas tuvieron lugar en dicha organización con el fin de garantizar al usuario final (desarrolladores en la FCV), ese nivel de tranquilidad que se necesita para iniciar el proyecto con buenas bases y soporte en el ámbito software.

2.2.1 Adquiriendo derechos sobre las librerías del sistema operativo disponible.

Es un hecho que uC/OS II era la mejor opción para implementar sistemas operativos sobre sistemas embebidos como las FPGA cumpliendo con todas las normas de aprobación, sin embargo es también un hecho que no es un software gratis, ya que son unas librerías con desarrollo comercial que debían ser adquiridas a través de la compañía, así que se hicieron las respectivas gestiones financieras, se contactó a Micrium y se licenció el software para su libre uso; gestión que tendrá peso en el momento de comercializar el monitor, así como generar fiabilidad de uso para aquellos monitores que vayan a terminar funcionando en el hospital de la FCV.

Ya con las herramientas y el sistema software desarrollado a través del entorno Nios II IDE listos para ser trasladados al nuevo entorno Nios II Eclipse, se procede a abordar las funcionalidades de sistema operativo con fluidez, aprender a cambiar ciertas opcionalidades desde el mismo entorno y afirmar mucho más el conocimiento del mundo software, evidenciando capacidad de manejo en la resolución de inconvenientes que se vislumbran con cada nueva implementación, optimización y detalle necesario para ejecutar una interfaz gráfica de usuario completamente funcional y mejorada.

2.3 SOLUCIÓN DE ERRORES DE DESCRIPCIÓN HARDWARE.

Ya antes se discutieron la serie de entornos y herramientas a disposición, es momento de abarcar algunos desarrollos del mundo hardware que venían siendo evidenciados en el proceso de estudio y manejo de los sistemas, quienes afectan a todo el equipo de trabajo con nuevos desarrollos innecesarios, desaciertos de tipo físico funcional de la descripción sistemática, planteada previamente como base hardware del MSV (Sección 2.1.1.1.1).

Los problemas de funcionamiento interno evidencian fallas y silencios inesperados en el sonido, estabilidad y corrimiento del video, modos de graficación de las ondas vitales del paciente, así como problemas de comunicación hardware entre procesadores software, flujo síncrono de datos al sistema software desde el módulo de adquisición de datos que se comunica con la tarjeta de signos vitales, optimizaciones de la cantidad de uso de lógica programable, entre otros arreglos que tomaron bastante tiempo a partir de una metodología de pruebas, implementaciones, rediseños e implementaciones finales; esto teniendo en cuenta que cada compilación demora un tiempo considerable dado la cantidad de hardware existente y que a su vez el entorno Quartus II debe procesar desde cero, a partir de la ocurrencia cambios en las diversas descripciones hardware por mínimo que estos sean.

Muchos de los desarrollos implementados no pueden ser especificados en este documento dado a políticas de confidencialidad de la organización; sin embargo, se dará una ejemplificación del tipo de problema y como fue solucionado para cada caso, esto con el fin de dar muestra de la veracidad de los desarrollos y evidenciar problemas de implementación que pudiesen ayudar a los interesados que enfrenten entornos de desarrollo embebido sobre FPGAs.

- A. El primer error se evidenció como un aparente estado software de falla en los sonidos de alarma del sistema, en donde en algunas ocasiones en las que sucedía un cambio de tono del “Beep” cardíaco, el sonido de repente se cancelaba por completo durante aproximadamente un minuto.

Después de realizar un par de pruebas de sonido y ver como el sistema software cumplía correctamente su función, solo restaba revisar el modulo hardware encargado de reproducir los sonidos del sistema, en donde se logró detectar una falla muy mínima reflejada sobre el modulo divisor de reloj encargado de generar la frecuencia tonal, esto a partir de una onda cuadrada que era generada a través de un contador y un cambio de nivel basado en una comparación de un conteo y una referencia.

La secuencia base de comparación, proporcionaba el cambio lógico de nivel de dicho reloj a través de la relación entre la entrada del módulo (Valor de referencia) y el contador ascendente implícito en el módulo, lo cual es correcto para generar dicho cambio de onda cuadrada que dará vida al módulo hardware de sonido; sin embargo el tipo de comparación se ejecutaba a través de una operación de igualación ($==$), lo cual es un error en este tipo de casos, ya que el módulo va a estar cambiando constantemente su valor de referencia, alterando así el valor de comparación final, el cual podrá saltar su responsabilidad de comparación, impidiendo la salida de la forma de onda cuadrada de sonido, junto con la máquina de estados que este módulo supone.

El problema surge mientras dicho contador posea un valor relativamente alto y justo después, el sistema software decide cambiar la tonalidad del sonido enviando un valor de referencia menor que se comparará con dicho contador; la igualación del valor alto del conteo con el nuevo valor de referencia menor nunca se llevara a cabo y es entonces donde el contador deberá hacer overflow o sobrepaso de su máximo valor posible para volver a iniciar desde cero y así alcanzar dicha comparación errónea de tipo igualación ($==$) nuevamente, razón por la cual el modulo tomaba cerca de un minuto (contador de 32 bits) en volver a generar siquiera un flanco de onda cuadrada nuevamente, afectando por completo la funcionalidad de este sistema hardware y el silencio percibido por el usuario.

La solución se basó en comparar estos dos valores a partir de la forma mayor igual ($>=$), ya que si sucede una disminución sorpresiva en el valor de referencia respecto a un valor alto de contador debido a diferencias de reloj y cambios que devienen del sistema software, la igualación de “mayoría” podrá reiniciar el modulo a cero inmediatamente, siendo imperceptible en tiempo para el usuario, ya que se trata de un reloj de onda cuadrada referido a un valor mínimo de 8kHz de frecuencia de funcionamiento.

- B. Otro error se pudo evidenciar como un aparente estado de falla software, en el dibujado de los números que indican los valores estáticos de los signos vitales de los pacientes, en donde en algunas ocasiones muy esporádicas surgía un parpadeo incomodo, que imprimía un toque ciertamente desconcertante y poco fiable para un equipo médico.

En definitiva fue un error que hizo dudar bastante en un inicio, pero al revisar el correcto funcionamiento de la adquisición de datos por el sistema software, así como verificar la

fiabilidad y estabilidad en la recepción de datos del sistema hardware a través de máquinas de estado, se puso en tela de juicio el tema de la sincronización de datos y el tiempo de propagación de los mismos, en donde juega un papel importante evitar la recepción de un dato, con la misma señal de reloj que está encargada de modificarlo, ya que siempre se debe garantizar un tiempo mínimo para que los sistemas hardware propaguen su señal a través de compuertas, transistores y conexiones físicas.

En efecto el error surgía a través de la sincronización entre el módulo de adquisición software para con el módulo de adquisición hardware de la tarjeta de signos vitales, el cual se encargaba de recibir los datos a través del accionar de una señal de reloj que a su vez se encargaba de generar la sincronización de dicho módulo de adquisición hardware; esta falla de sincronismo permitía que los nuevos datos que llegaban a la FPGA pudieran ser modificados al tiempo que eran leídos por el sistema software como tal, cosa que se considera tiempo nulo en la propagación de los datos y que en definitiva producía el parpadeo evidenciado de manera repetitiva en la interfaz gráfica, la cual recibía un dato erróneo producto del pésimo sincronismo y lo mantenía mientras la tarea software de adquisición de datos volvía a leer un nuevo dato del módulo cada cierto tiempo y fortuitamente ya era correcto, alternando entre un valores bien propagados y otros que no.

La solución a este inconveniente, fue analizado con detenimiento para hacer notar que el reloj encargado de proporcionar el sincronismo a los datos provenientes de la tarjeta de adquisición, era completamente innecesario y único producto de los cambios que han tenido lugar con el paso de los desarrollos de optimización, motivo por el cual fue eliminado; ahora bien, sabiendo del sincronismo faltante sobre este módulo de adquisición vital con respecto al módulo de adquisición del sistema software, se decidió implementar que ambos módulos funcionasen a través de la misma señal de reloj pero en flancos diferentes de onda, garantizando un tiempo de medio ciclo de reloj en cuanto a la propagación de las señales que tanto se recalcaron anteriormente, tiempo que debe ser tenido en cuenta siempre y más en sistema con gran peso hardware como se refleja en este proyecto, el cual tendrá múltiples y pequeños retrasos acumulados, producto de los ruteos y vías internamente adaptados en cada nueva reprogramación global de descripción hardware sobre la FPGA.

- C. Desajuste de las señales de video en el eje vertical, en dónde las gráficas de hardware de alarmas de video eran correctas, pero eran dibujadas de forma alternada poco convencional por el sistema de video, en donde parte superior de dichas alarmas, eran mostradas en la zona inferior del video, mientras que el restante inferior se encontraba dibujado en la zona superior de la pantalla, tal y como sería una imagen empotrada en un rodillo que cambia su posición.

En definitiva esta era una señal clara de desajuste en el módulo de sincronización de video, el cual suponía un mal reinicio a cero del contador que lleva la razón de los pixeles sobre el eje vertical; este sincronizador funciona como una guía del sistema de graficación de alarma vital hardware y estaba obteniendo una mala información de parte de su guía.

Fue sencillo revisar el modulo y ver que aunque poseía sus respectivos valores de inicialización, ha de saberse que el sistema de video será detenido e iniciado al antojo del

software mientras el módulo de sincronización guía continua en ejecución, por ende no bastaba una inicialización local de dicho contador en el módulo de sincronización, sino un sistema de reinicio basado en las mismas señales de video, del cual se pudieran llevar a cero los contadores de dicho módulo, en el momento en que las señales mismas indicaran la “no zona activa de video”.

La solución se centró en una sencilla implementación de reseteo asíncrono que utiliza las señales de habilitación de video como base de reinicio y eso sería todo para la generación de un correcto conteo vertical de pixeles, lo cual resolvería el problema de desajuste vertical.

- D. El módulo de graficación de las ondas por medios hardware en general, empezó a presentar múltiples problemas de cambio de posición y limitación en cuanto a opcionalidad de configuración dado a los nuevos requerimientos que fueron surgiendo y que se presentan con normalidad en cualquier núcleo de diseño.

Se revisó el módulo hardware en cuestión y se evidenció como está completamente asfixiado por los múltiples desarrollos que han tenido lugar a mejoras y corrección de fallas sobre el mismo, a esto se le suma una incomprensión total y carencia de sentido en varios de los arreglos allí implementados, que se da por la misma forma parchada que refleja el desarrollo sobre este módulo.

En definitiva la solución que mejor se acomodó a la percepción del grupo de trabajo, se centró en el rediseño definitivo del módulo, de esta manera se extrajo lo máximo que se pudo comprender en cuanto a la información confusa del módulo existente y los requerimientos que se pretendían lograr con la nueva implementación; el resultado del nuevo diseño es completamente observable a través de la estabilidad gráfica que visualmente proporcionan dichas ondas sobre la pantalla del monitor local, el nuevo nivel de opcionalidad para los requerimientos que se dieron en su momento e inclusive el rediseño optimizado en cuanto a gasto de elementos lógicos (LEs) de la tarjeta de desarrollo (De0-nano) y con ello, una mayor estabilidad en el sistema de descripción hardware.

- E. Optimización de los módulos hardware sobre todo el sistema.

Esta es una situación referida a toda la modularidad del proyecto de descripción hardware, en donde se realiza una labor de análisis y recorte de recursos innecesarios que se pudiesen ver reflejados sobre los diferentes módulos hardware y más específicamente en los módulos de gran volumen junto a algunos repetitivos como pudo reflejarse en el rediseño que tuvo lugar en el punto anterior (Repetitivo de ondas vitales).

Como se menciona previamente, esta etapa abarca cierto nivel de desarrollo software que tiene sus estudios desde el inicio de esta práctica, sin embargo queda aún un pequeño asunto respecto al desarrollo hardware del que dispone la plataforma Qsys, este asunto se resuelve en la necesidad de generar un módulo de intercomunicación interna entre los procesadores embebidos existentes (mailbox), el cual no se encuentra en el panel de

elementos de dicha plataforma, es decir, no basta arrastrarlo al entorno y juntar un par de cables por ahora, ya que en teoría este módulo no existe en la plataforma Qsys y crea así la necesidad de diseñarlo desde cero junto a los estándares de comunicación Avalon que ofrece la compañía Altera como alternativa principal, proceso que requirió un tiempo para la investigación e implementación de dichos estándares sobre el proyecto en un nuevo módulo hardware-Qsys y con ello una mejor comprensión de la dinámica entre los mundos hardware software.

Finalmente un pequeño manual de creación de estos módulos del panel de componentes para el fácil manejo y desarrollo futuro sobre FPGA en entornos de sistemas integrados, el cual fue entregado en parte a los desarrollos de la FCV, esto para los interesados en adquirir el conocimiento e inclusive ir un poco más allá si se hace oportuno.

2.4 COMPRENSIÓN Y CREACIÓN DE COMPONENTES SOBRE EL ENTORNO QSYS

Para retomar lo que envuelve la plataforma Qsys, se tiene básicamente un entorno de desarrollo de hardware a través de descripción gráfica más avanzada, rápida y poco precisa, en comparación al desarrollo literal de los diferentes códigos de descripción, en la que se logran mezclar e interconectar todo tipo de modularidad y lógica embebida a mayor escala sobre la premisa de estándares de comunicación que logran la interactividad entre los mismos.

2.4.1 Presentación del componente.

El módulo que se plantea como un intercomunicador entre procesadores (mailbox), lucirá así:

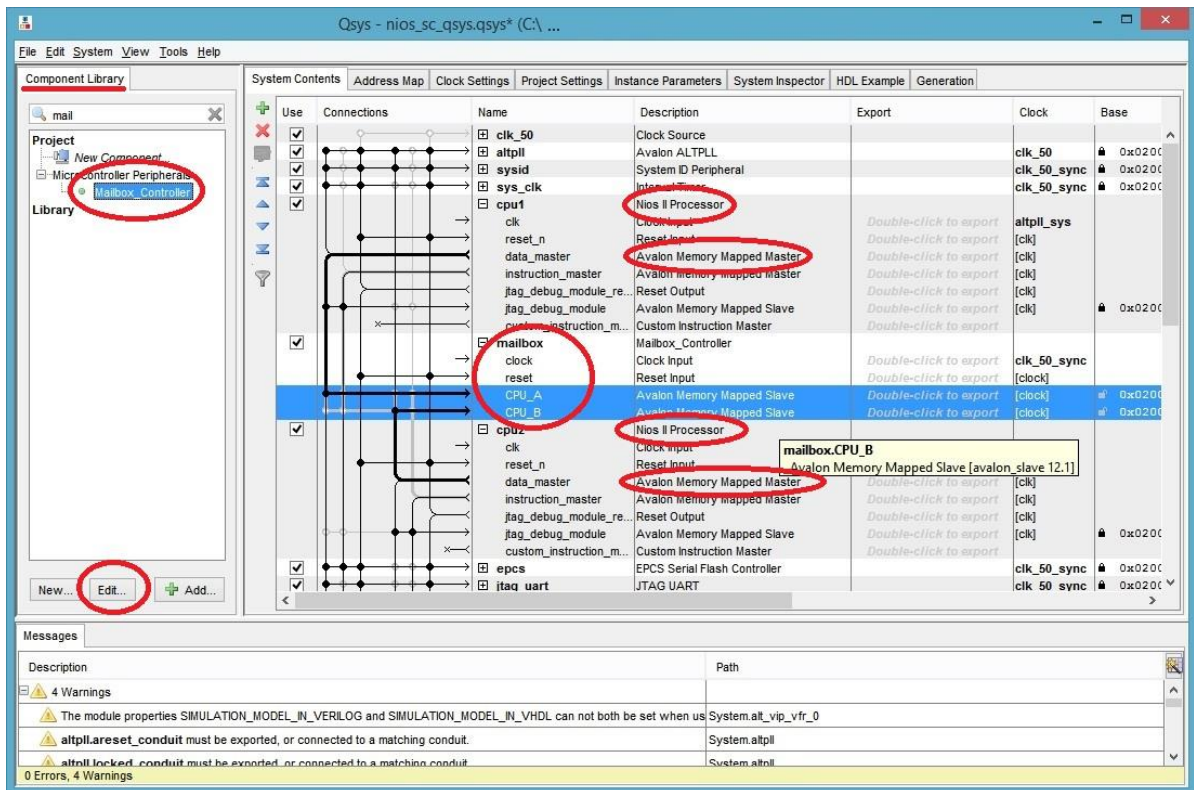


Figura 25. Módulo “mailbox” e interconexión de procesadores sobre el entorno Qsys.

Como se logra apreciar en la Figura 25, el concepto del módulo (mailbox) pretende abarcar un estándar de comunicación entre procesadores (Avalon Memory Mapped), a través del cual se establecen una serie de espacios de memoria (lectura y escritura), en los cuales se pueden dejar mensajes o “mails” en la “bandeja de entrada” del otro procesador, proceso que permitirá establecer cierta flexibilidad entre múltiples procesadores completamente diferentes en sus características, esto siempre y cuando se sigan las reglas del dicho módulo, las cuales se explicarán más adelante y se verán reflejadas sobre la lógica software o librería-controlador del módulo.

Ya se tiene claro el objetivo del módulo, sin embargo se debe conocer más exactamente el tipo de comunicación que representa un estándar avalon memory mapped (AMM), ya que es deber del ingeniero electrónico, establecer las pautas y hacer los cambios pertinentes llegada una falla o problema de funcionamiento.

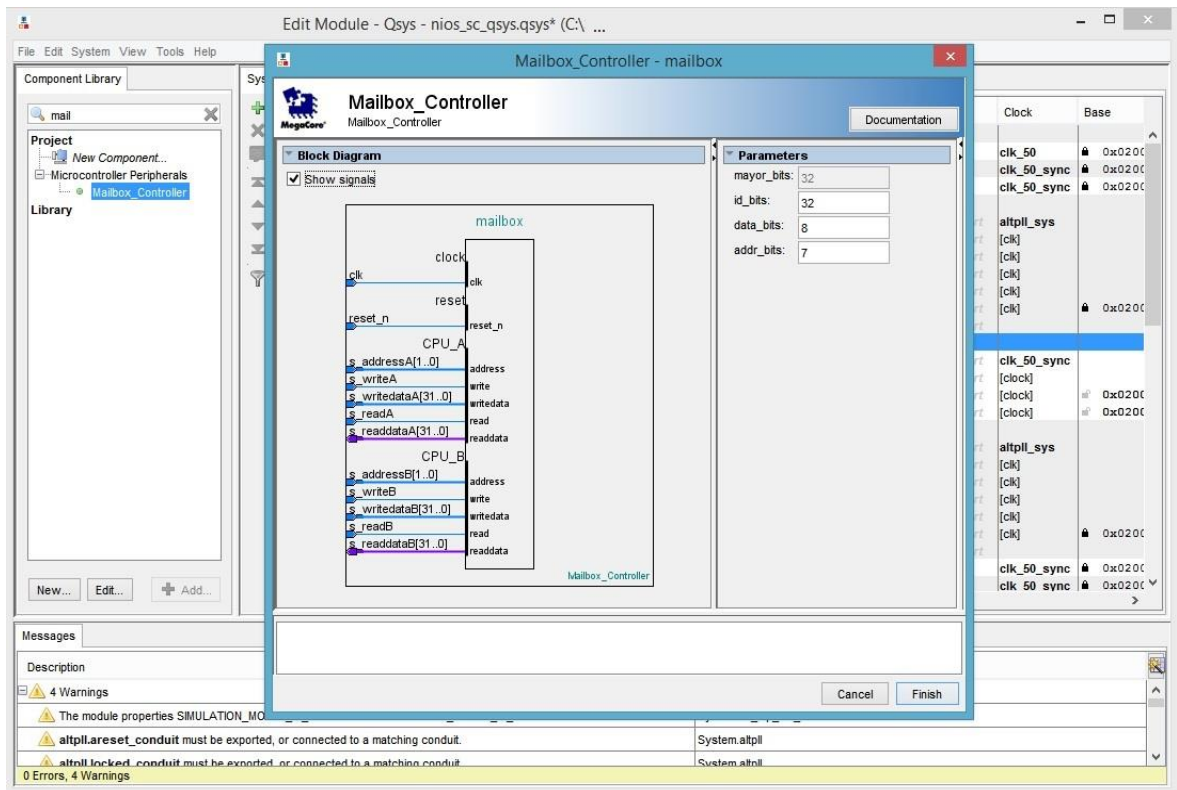


Figura 26. Estructura de comunicación “mailbox” sobre el entorno Qsys.

La Figura 26 representa como cada nuevo sistema en el entorno Qsys debe poseer su respectivo valor de reloj y reseteo, con los cuales se supone la sincronización de la comunicación para con los dispositivos debido al funcionamiento maestro-esclavo de este y muchos de los estándares industriales de comunicación existentes en el planeta (La compañía Altera en su página web promociona sus dispositivos y estándares como muy compatibles con otros sistemas comerciales). Por otro lado se sostiene la premisa de este estándar avalon memory mapped slave (AMM-Slave) como dos interfaces de comunicación hardware llamadas “CPU_A” y “CPU_B” según la Figura 26, interfaces esclavas dedicadas a trabajar con los valores de reseteo y reloj y así lograr una comunicación estable entre con sus respectivos maestros “AMM-Master” existentes, que para este caso serán las interconexiones con los procesadores del sistema que se viene describiendo (Figura 25).

2.4.2 Creación y edición del componente.

Para la creación de estos módulos, el entorno presenta unas opciones de “nuevo” y “editar” en la parte inferior de su panel izquierdo llamado “Librería de componentes” (Figura 25), en las cuales se establecen las pautas, parámetros e interfaces que manejará el módulo en cuestión; al entrar en modo “editar” del módulo mailbox se puede apreciar lo siguiente.

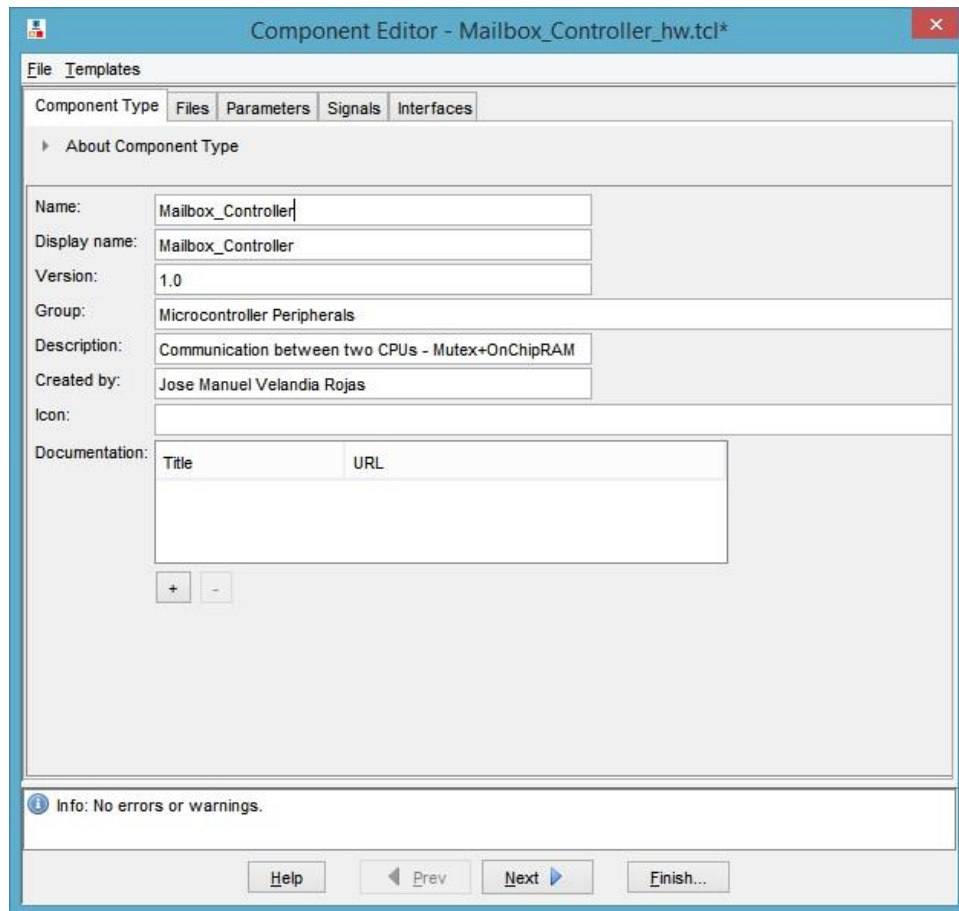


Figura 27. Pestaña de descripción básica del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.

No hay mucho que decir es simplemente un espacio para adjuntar valores de autoría e información del componente o módulo de mailbox, el cual se guardara en formato de “script” cuya extensión será (*.tcl) como descriptor simplificado de todo el componente.

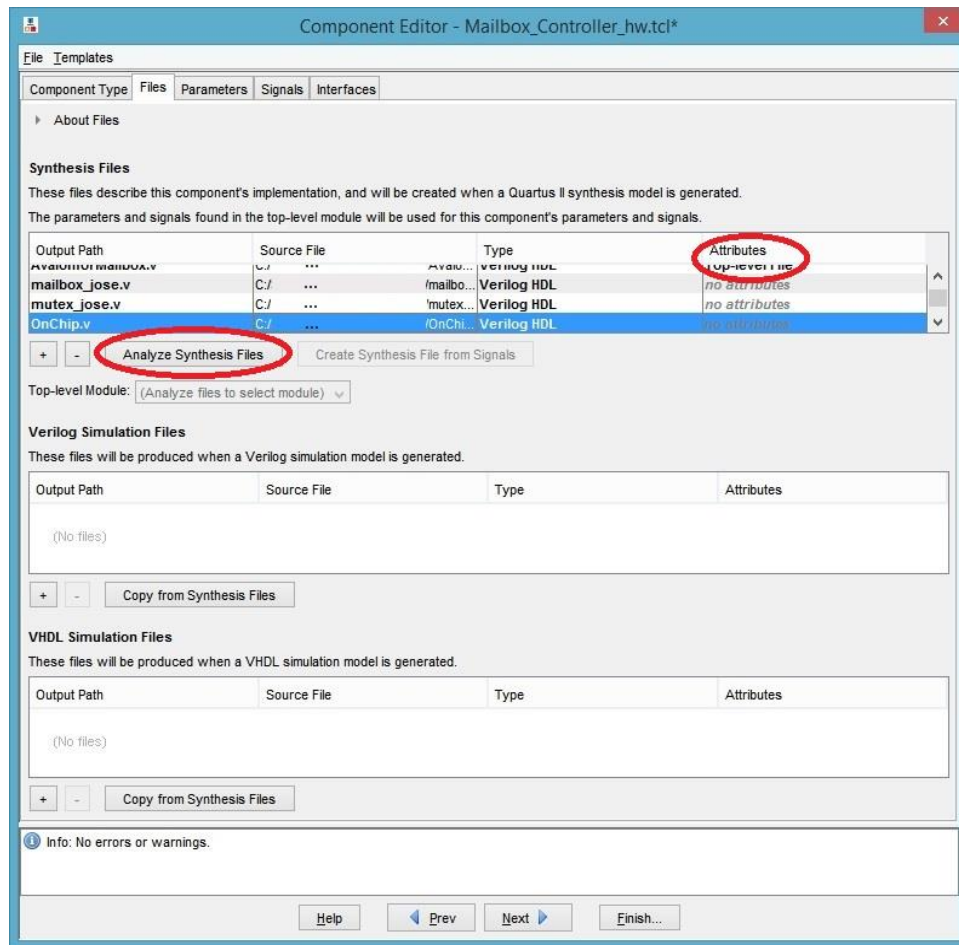


Figura 28. Pestaña de archivos de descripción del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.

Esta es probablemente la pestaña más importante, ya que finalmente enlaza los archivos de descripción hardware (Verilog), que pautan la funcionalidad física del componente, la cual es la esencia de cualquier descripción a mayor escala.

Para comprender un poco mejor el concepto del mailbox, este está compuesto por una memoria Verilog de registros “Onchip RAM”, actuando como bandeja de mensajes que dejan los procesadores, un módulo de exclusión mutua o “mutex” Verilog, el cual se encargará de limitar el acceso a un procesador a la vez sobre dicha bandeja, evitando errores en las señales y corrupción de datos, un módulo Verilog mailbox que enlaza los recursos de Onchip y Mutex, siendo apropiado por un módulo Verilog final que se encargará de adaptar la funcionalidad mailbox, con la comunicación (Avalon), que plantea el entorno para con los procesadores.

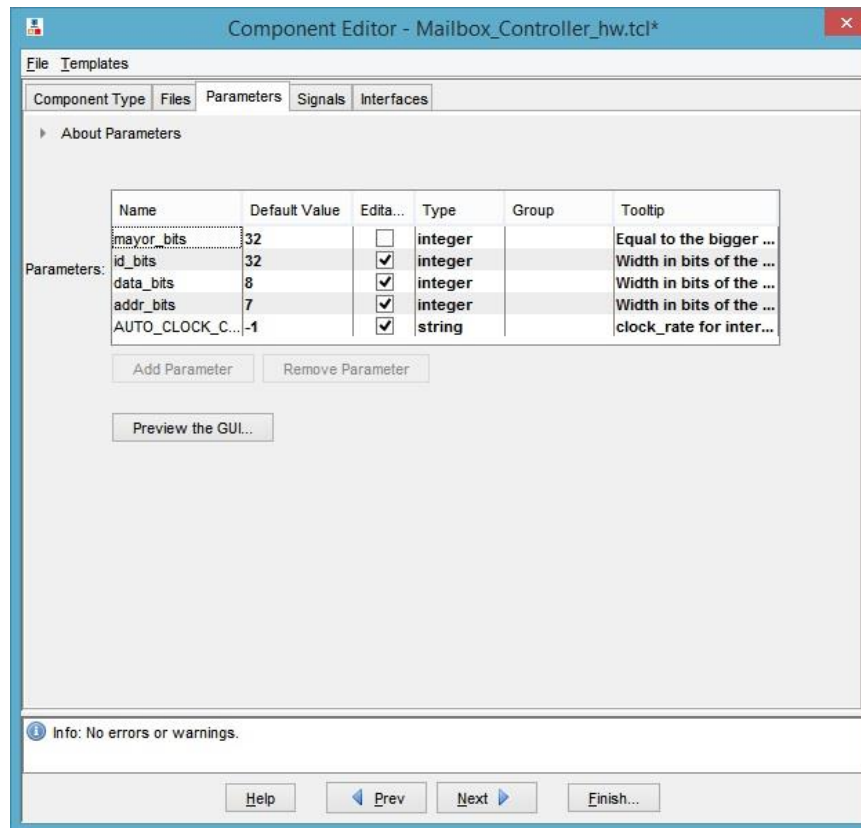


Figura 29. Pestaña de parámetros del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.

Esta pestaña permite definir algunas características y descripciones menores de los parámetros existentes en el componente gracias a los mismos existentes en los archivos de descripción hardware de la pestaña anterior (Figura 28). Para que este reconocimiento, así como el reconocimiento de las siguientes pestañas se lleve a cabo, se debe definir el archivo de máxima jerarquía (Top Level File) en la columna de atributos de los archivos de la pestaña anterior, sobre el archivo responsable de la comunicación avalon y por ultimo clicar en el botón de análisis para que la herramienta se pueda cerciorar que la descripción escrita en los archivos está correctamente aplicada o editada para ser funcional con una descripción hardware (Figura 28).

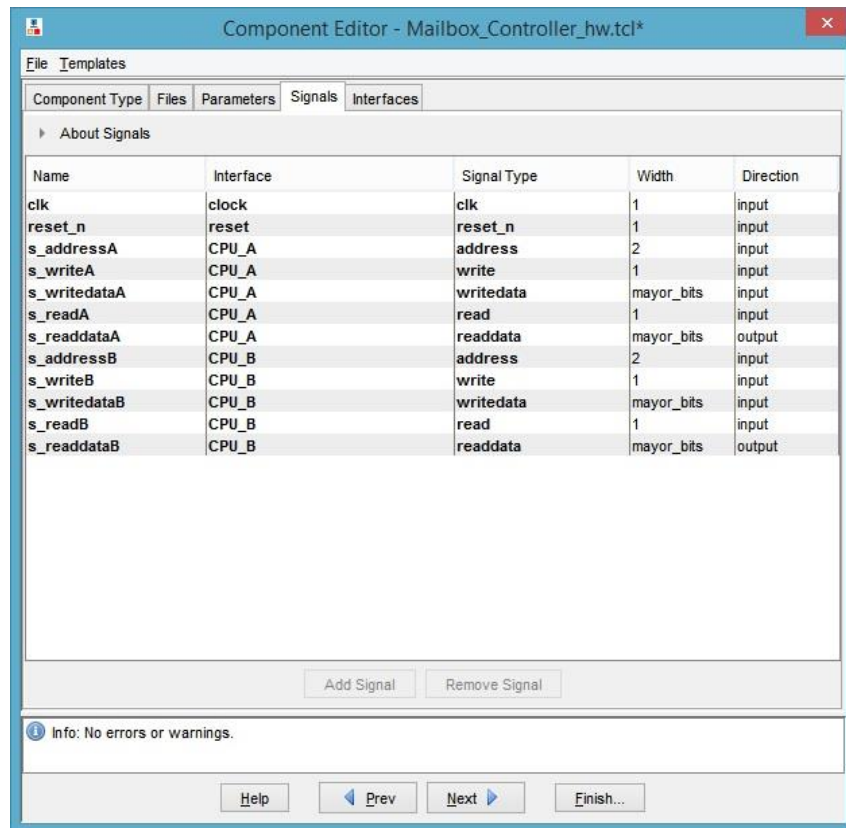


Figura 30. Pestaña de señales hardware del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.

Una vez el archivo de mayor jerarquía es analizado (compilación corta), esta pestaña extrae sus respectivos nombres de entradas y salidas, para que el usuario pueda hacer los cambios respecto al tipo de interfaz a la que pertenecen las señales, así como el tipo de señal dentro de dicha interfaz avalon, restando cambios de nombre si se desea, aunque no es recomendable hacer esto último ya que pueden haber confusiones futuras, lo recomendado sería cambiar el archivo de descripción y repetir el proceso de análisis (Figura 28).

Aunque por el momento en la columna de interfaces de las señales ejemplificadas en la Figura 30 se observa “CPU_A” y “CPU_B”, estas son realmente de tipo “AMM-Slave”, solo que al incursionar en la pestaña de “Interfaces” de la Figura 31 se pueden cambiar nombres de la interfaz para mayor guía en la edición del componente y aun así no dejen de ser tipo “AMM-Slave” (Figura 25).

Por otro lado cada señal continúa con su tipo de señal asignado en la columna “Signal Type” dentro de la interfaz AMM-Slave a la que pertenecen (ej. La señal “s_readA” sigue siendo del tipo “read” dentro de la primera interfaz “AMM-Slave” (llamada CPU_A) y no cambiará ese identificador “read” mientras siga siendo del mismo tipo de señal).

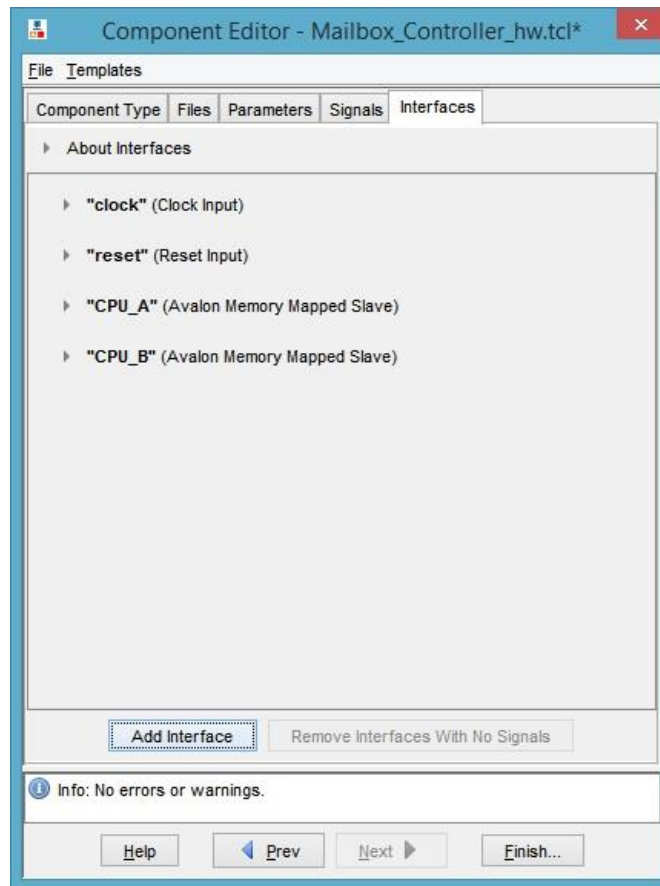


Figura 31. Pestaña de Interfaces del módulo “mailbox” sobre la herramienta de edición del entorno Qsys.

Como última instancia se tiene la pestaña de interfaces, la cual resume la creación del componente y orienta la lógica de comunicación del estándar avalon, en base a las especificaciones realizadas en la pestaña de señales (Signals), esto con el fin de no tener ningún error en la barra de información inferior y lograr un componente completamente compatible y funcional para con el entorno, ya que hay unos mínimos enlaces de señales de reloj y reseteo necesarias para que una interfaz de tipo AMM-Slave sea correcta y se resuelve en esta pestaña.

2.4.3 Temporización del estándar de comunicación “Avalon Memory Mapped”.

Ahora bien para entender la dinámica de comunicación, bastará con desplegar alguna de las interfaces avalon (CPU_A o CPU_B) de la Figura 31 y echar un vistazo a las formas de onda, tal y como se ejemplifica a continuación, cabe aclarar que las formas de onda no vienen provistas del análisis del archivo de mayor jerarquía de la Figura 28, sino que siempre están dadas por estas pestañas, para así garantizar que el “timing” o temporización para este tipo de interfaz sea correcta, ya que es un estándar y por ende no debe cambiar.

Aunque es una interfaz tipo esclavo la que se pretende mostrar, es muy probable que el lector pueda entender la interfaz maestro al mismo tiempo, ya que si se entienden las formas de onda con las que el esclavo recibe, se puede intuir lo que un maestro tuvo que realizar para que exista dicha comunicación y por ende un hardware que use tal lógica debe ser desarrollado desde la perspectiva maestro con pocas posibilidades para el error.

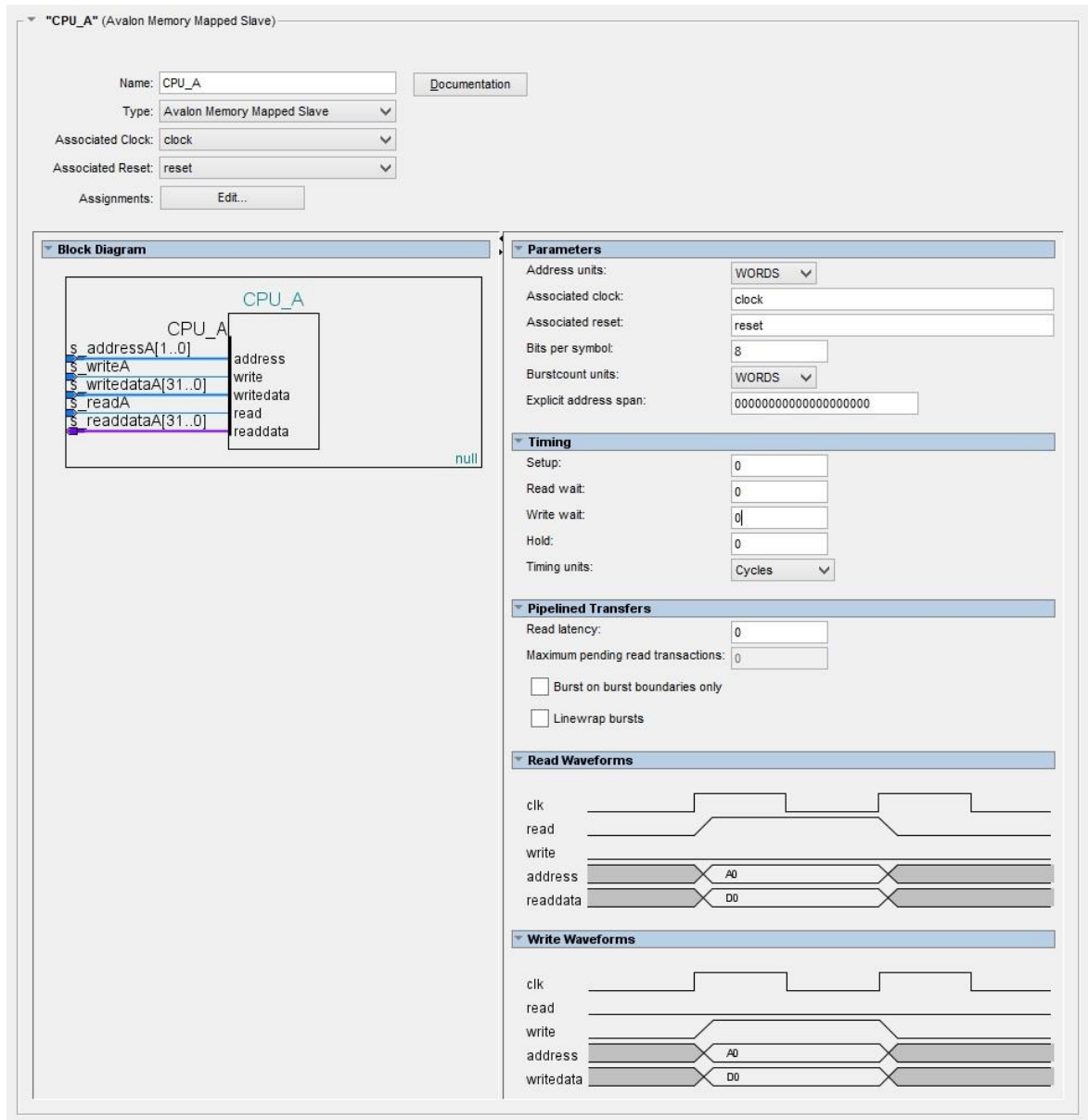


Figura 32. Dinámica modificada de comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.

Como se aprecia en la Figura 32, la interfaz posee sus respectivas características de forma, orden y temporizaciones, asocia siempre un reloj y un reseteo y proporciona las

diferentes formas de onda guía respecto a las señales previamente creadas y enlazadas, así como algunas otras opciones de preferencia para el desarrollador, sin embargo para una configuración sencilla sin esperas de lectura o escritura ni de latencia en la lectura, se tiene que para “Read Waveforms” (De esclavo a maestro):

- a. Cada flanco positivo de la señal de reloj asociada y en orden de atender una orden de lectura del dispositivo maestro, la señal de un bit de tipo “read” (s_readA en este caso) tomará un nivel alto indicando que el procesador desea leer un dato.
- b. La señal de un bit de tipo “write” (s_writeA en este caso) deberá permanecer en bajo.
- c. Al tiempo que la señal de escritura está lista, la señal de dirección a la cual se desea escribir el dato en el componente, también debe estar disponible en la señal de entrada tipo “address”. Este proceso se puede asemejar al funcionamiento de lectura de una memoria eligiendo un sector de memoria del cual leer, en donde una dirección proporciona una respuesta diferente de la región que se desea leer sobre una misma y única salida hacia el procesador en este caso.
- d. Una vez la señal de tipo “read” y “address” se encuentran listas, el módulo sabrá que dato entregar hacia el procesador, el cual deberá estar disponible para ser leído en la señal de salida tipo “readdata” inmediatamente debido a la configuración designada para la Figura 32. Esta salida que el procesador pretende leer a través de la señal de tipo “readdata”, está limitada en tamaño al ancho máximo de una palabra o “Word” de 32 bits que resulta de la arquitectura propia del dispositivo maestro o procesador para este caso y por la cual se pretende una comunicación serializada.
- e. Para el siguiente flanco positivo de reloj, la señal de tipo “read” y “address” ya no estará más disponibles y acabará el proceso de lectura de la señal “readdata”.

Una vez ha terminado el proceso (el cual puede verse en las formas de onda proporcionadas por el entorno para una interfaz de tipo “AMM-Slave” en la Figura 32), solo resta decir que se terminó el proceso de serialización y el procesador ha recibido el dato que quería obtener del mundo hardware, a través de la misma señal de 32 bits de tipo “readdata” para este y muchos otros componentes de la descripción, por supuesto haciendo uso de su respectivo sistema de entrada-salida IO (input-output) propios, tal y como sucedería en cualquier otro procesador o inclusive microcontrolador al adquirir un dato del mundo exterior hacia su lógica interna.

El destino de este dato leído pasara a manos de la lógica software prevista, descrita y finalmente programada en dicho procesador, ya sea almacenar el dato a través de una variable en la memoria RAM que corre el sistema software, realizar una operación matemática con el mismo o sencillamente despreciar el dato y seguir adelante, son factores software que se trataran más adelante y que ya no dependen del entorno Qsys, siendo los factores binarios y de proceso hardware los que interesan en esta parte de la

dinámica del sistema, factores como el proceso de escritura “Write Waveforms” del “AMM-Slave” el cual se describe a continuación:

- a. Cada flanco positivo de la señal de reloj asociada y en orden de atender una orden de escritura del dispositivo maestro, la señal de un bit de tipo “write” (s_writeA en este caso) tomará un nivel alto indicando que el procesador desea escribir un dato.
- b. La señal de un bit de tipo “read” (s_readA en este caso) deberá permanecer en bajo.
- c. Una vez la señal de tipo “write” se encuentra en nivel lógico alto, el dato que el procesador desea escribir ya estará disponible para la escritura al mundo hardware en la señal de tipo “writedata”.
- d. Al tiempo que el dato y la señal de escritura están listos, la señal de dirección a la cual se desea escribir el dato en el componente, también estará disponible en la señal de entrada tipo “address” una vez más, de forma similar a como ocurría en el proceso de lectura, solo que esta vez se desea escribir un dato (writedata) a partir de una dirección (address) hacia varias regiones del mundo hardware.
- e. Observando que la señal de escritura tipo “write” caerá a nivel lógico bajo al siguiente flanco positivo del reloj junto con el dato y la dirección a la cual el procesador desea escribir en el módulo, es claro para el desarrollador que el momento indicado para adquirir dicho dato proveniente del procesador, está dado por el flanco negativo del reloj principal asociado, el cual ocurre mientras la señal de tipo “write” se encuentre en nivel lógico alto como indicador, así se evitan problemas de propagación e inclusive ruido de establecimiento, tanto del dato como de la dirección.

De la configuración anterior se puede inferir una rápida respuesta de escritura por parte del procesador hacia el mundo hardware y viceversa, sin embargo el proceso de lectura no debiera ser tan inmediato, ya que se incurriría en el error de adquirir una dirección poco estable en cuanto a propagación y establecimiento una vez la señal de lectura “read” hace su flanco positivo, lo cual resultaría en la lectura de un dato “readdata” erróneo, el cual sería adquirido por el procesador, es por ello que la configuración predeterminada que proporciona el entorno recién se crea dicho componente con interfaz AMM-Slave es la siguiente.

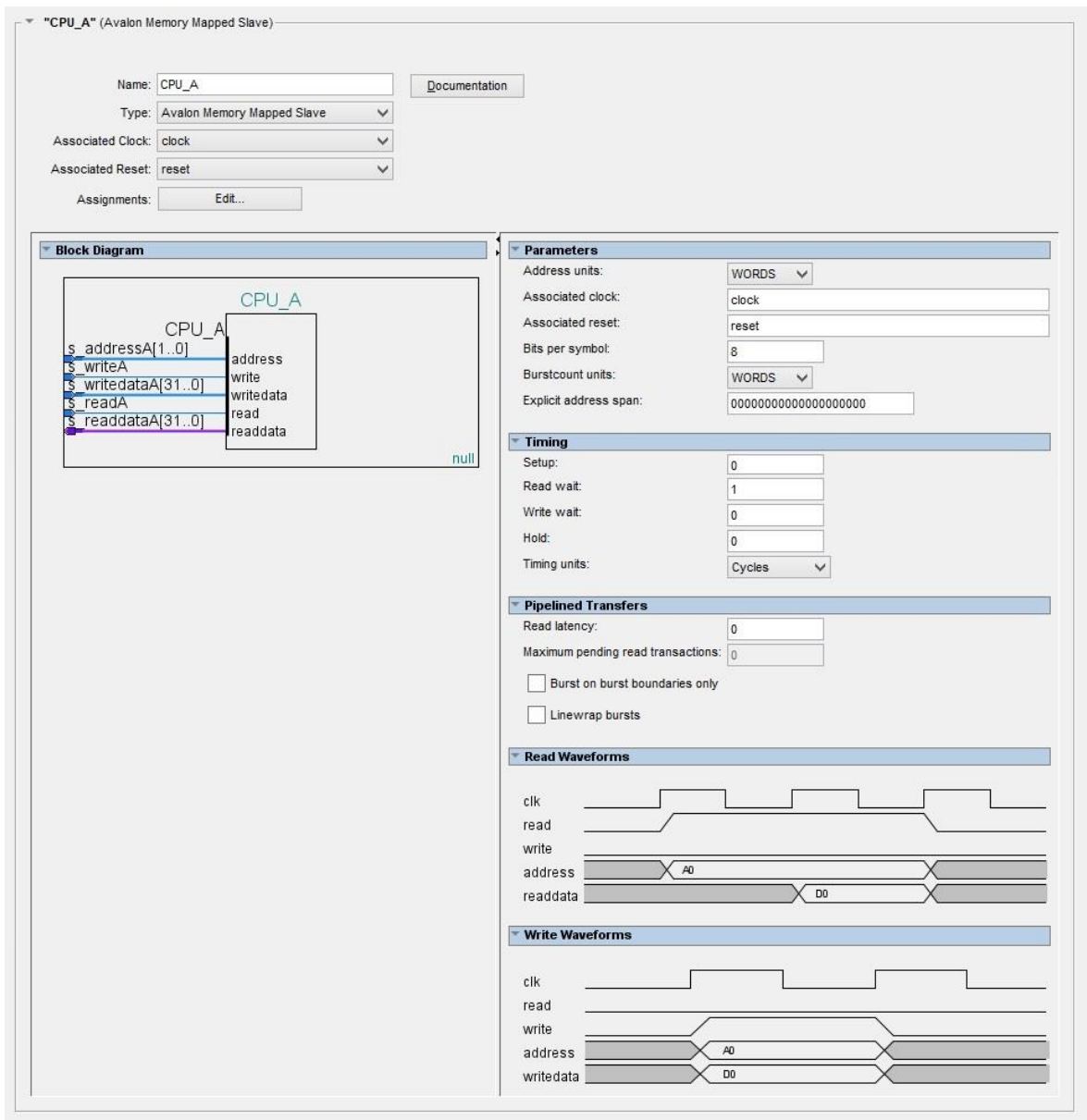


Figura 33. Dinámica predeterminada de comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.

En la Figura 33 se puede observar que existe un ciclo de espera en cuanto a la lectura, lo cual proporciona que el proceso de lectura le tome dos ciclos de reloj en ser completado, sin embargo se sacrifican para tener una lectura mucho más confiable y estable, en donde se puede aprovechar el primer flanco negativo de la señal de reloj que se genera después que la señal de “read” esté en alto, para leer y disponer del dato en la señal “readdata” que será leído por el procesador, esta forma de onda gana medio ciclo de reloj para obtener una señal de dirección “address” más estable y otro medio ciclo para dejar una señal “readdata” más estable para ser leída por el procesador.

En cuanto al proceso de escritura no se tiene este problema, ya que una vez las señales se encuentran listas pero inestables con el primer flanco positivo de reloj y la señal “write” se encuentra sobre nivel lógico alto, se dispone de medio ciclo de reloj para que dichas señales de dirección y dato se establezcan y puedan ser leídas sin ningún problema con el flanco negativo que prosigue en la forma de onda.

Con el fin de dar una pauta más instructiva respecto al funcionamiento del timing, latencias en lectura, escritura y demás, se realizan variadas configuraciones del componente en su interfaz AMM-Slave y se pone a disposición del lector para que saque sus conclusiones a partir de una pequeña explicación.

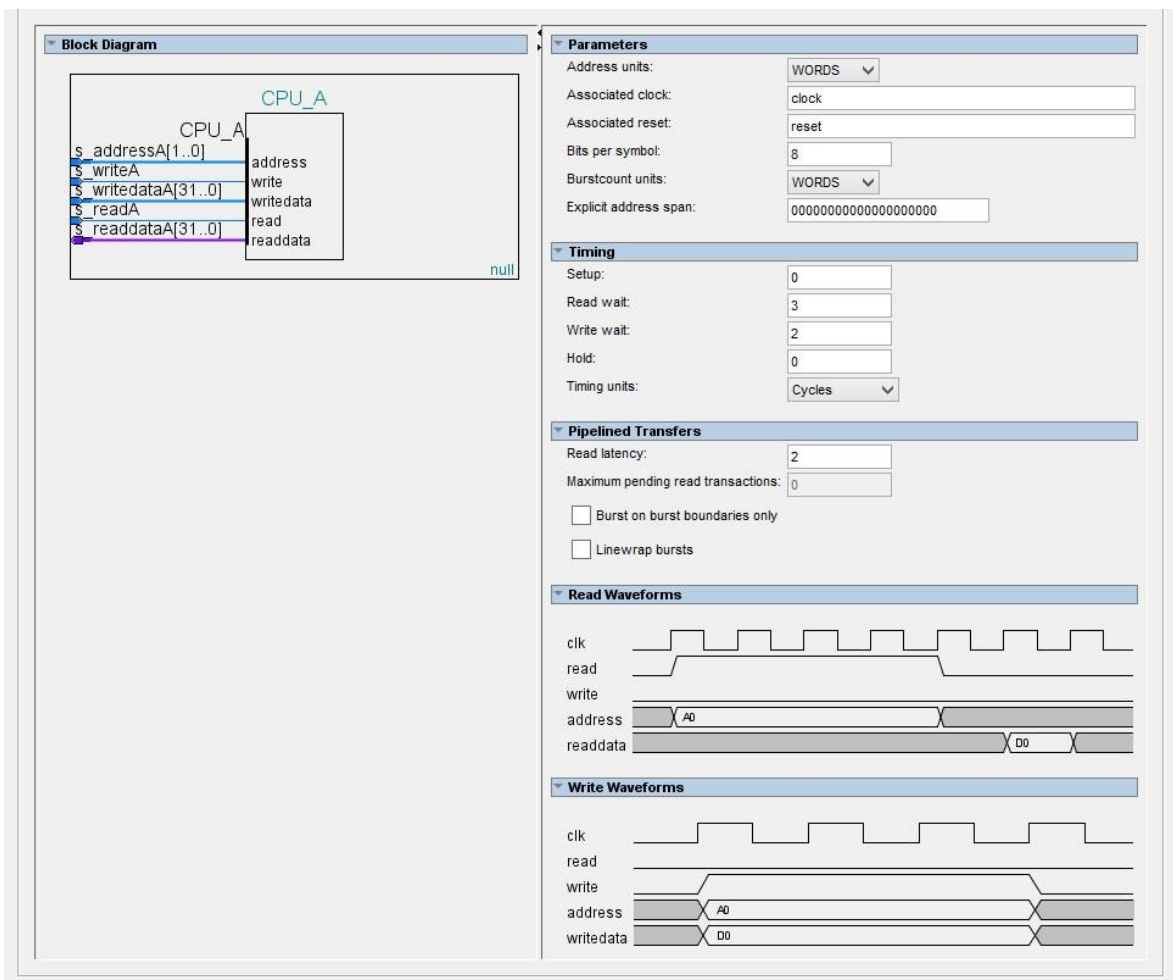


Figura 34. Timing 1 - Configuración aleatoria de la comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.

En la Figura 34 se observa como cada ciclo de espera o “wait” en cuanto a lectura o escritura, le suman un ciclo más de espera a las formas de onda de lectura (3 ciclos) y escritura (2 ciclos) respectivamente en el proceso de sostenibilidad de la dirección

“address”, mientras las señales indicadora “read” o “write” permanecen en alto. Por otro lado los dos ciclos de latencia de lectura solo afectan a la forma de onda en lectura y genera que la señal “readdata” sea leída con un desfase de dos ciclos de reloj después de su posición original sin latencia en un caso que el dato tenga retraso al ser leído por el módulo de mailbox o cualquiera que se esté diseñando.

Como ya se dijo antes, esta y todas las temporizaciones de las figuras que se configuren en el componente, deben concordar con la lógica descrita para el componente en su archivo Verilog de máxima jerarquía (Top Level File de la pestaña “Files” Figura 28), el cual es el encargado de gestionar la comunicación AMM-Slave con el módulo propiamente funcional “mailbox” para el cual fue diseñado dicho componente como puente con el procesador usando este tipo de comunicación.

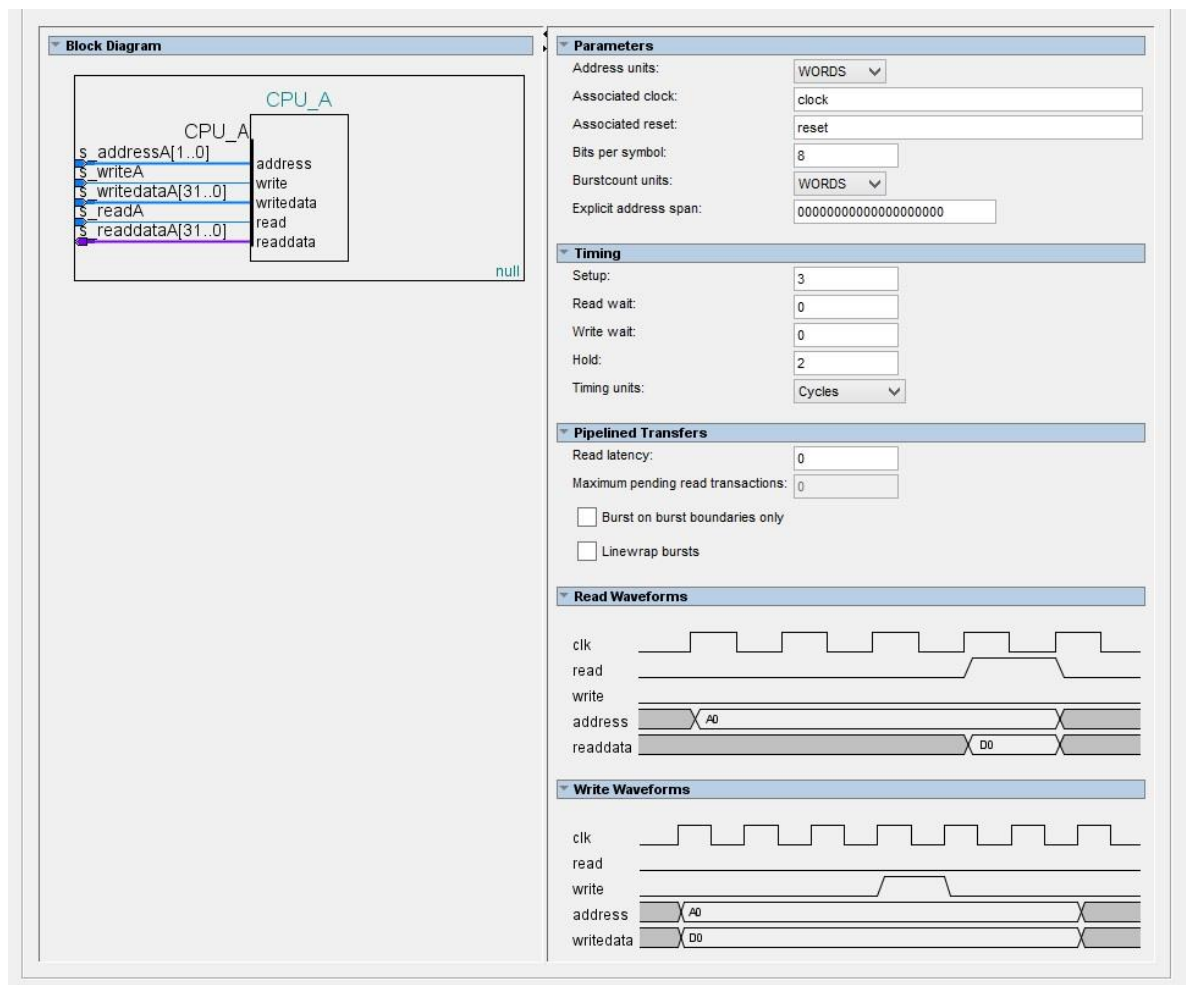


Figura 35. Timing 2 - Configuración aleatoria de la comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.

En este segundo ejemplo de temporización se vuelve a la configuración base pero poco recomendable de la Figura 32, solo que esta vez se agregan tres ciclos de “setup” y dos ciclos “hold” a la configuración, los cuales agregan un tiempo extra, tanto previo como posterior respectivamente, a la activación en nivel alto de las señales indicadoras de “read” o “write”, con la pequeña excepción de que los ciclos de “setup” no aplican a la señal de “read” ya que esta tiene su equivalente con los ciclos de latencia, la única diferencia es que mientras suceden los ciclos de latencia, la señal de dirección “address” ya no estará disponible, esto se podría dar por alguna razón propia de la arquitectura del procesador en donde no puede sostener su valor de dirección mientras su bandera indicadora de lectura “read” ya no se encuentre activa, sin embargo el desarrollador puede valerse de los ciclos de espera del ejemplo anterior para aumentar el tiempo en que la señal de dirección se sostiene activa si es lo que se desea.

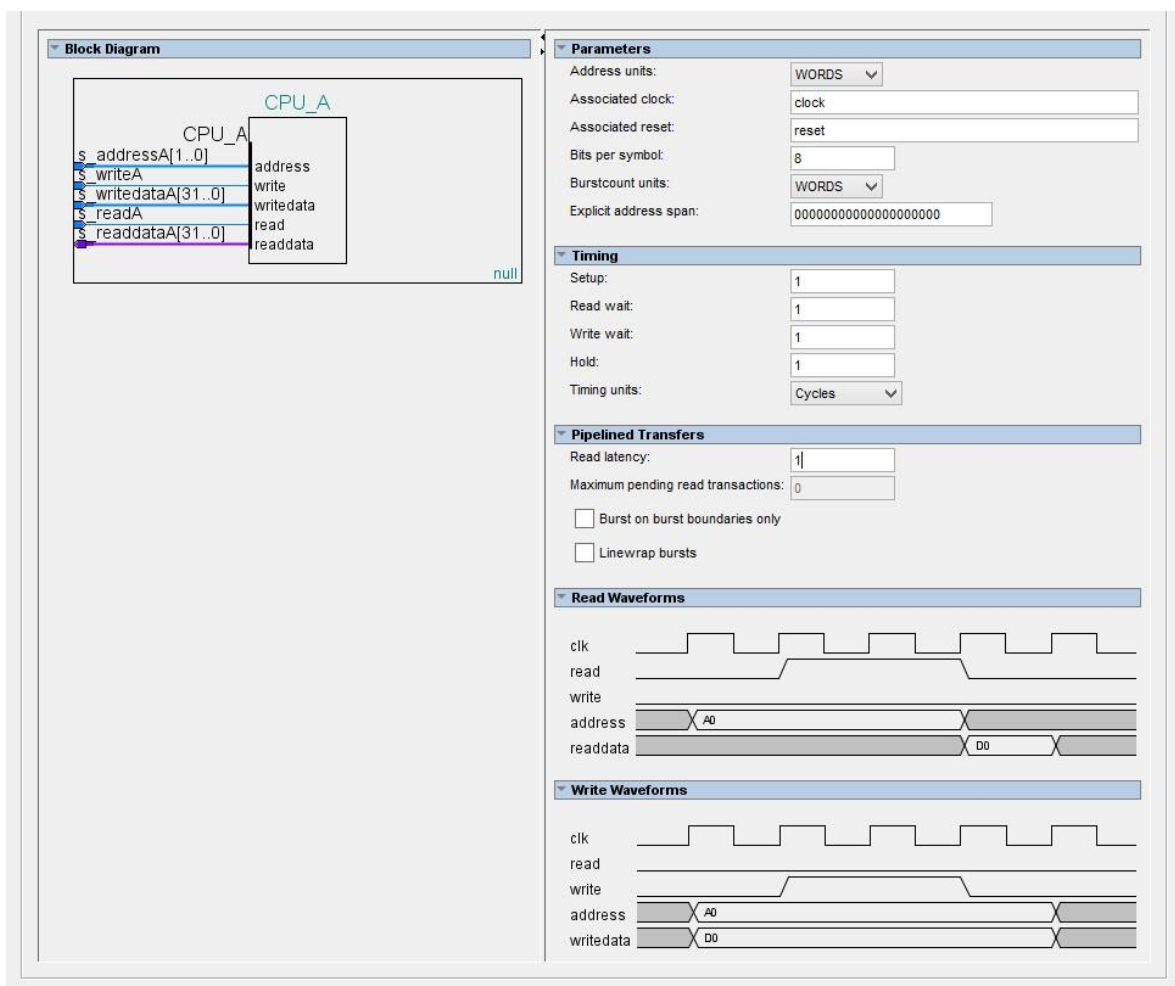


Figura 36. Timing 3 - Configuración aleatoria de la comunicación de una interfaz de tipo Avalon Memory Mapped - Slave.

Por ultimo un ejemplo final de configuración de la temporización para que cada quien saque sus conclusiones respecto a lo que se viene explicando y así entender más a fondo no solo el concepto de comunicación Avalon, sino varios conceptos de orden como latencia, espera, sostenimiento, estabilidad, entre otros y con ello una base muy importante en cuanto a entendimiento de los sistemas de comunicación que se utilizan en el día a día y no solo Avalon.

2.5 DESARROLLO SOFTWARE SOBRE EL ENTORNO NIOS II ECLIPSE

2.5.1 Presentación del entorno.

Ya se ha abarcado toda la teoría hardware posible, correcciones a diferentes módulos, creación de componentes y entendimiento de la estructura hardware y su conexión con el mundo software, lo cual concluye en el uso de un procesador a través de memorias y uso del entorno compilador disponible por parte de la compañía Altera, este entorno provee todas las características necesarias para lograr que la secuencialidad de las tareas descritas en forma software se vean reflejadas físicamente en el procesador y por ende en un equipo de monitorización vital completamente funcional.

El entorno como tal es una plataforma descrita en java que plantea de manera más gráfica las opcionalidades de compilación y edición de la consola de comandos, quien realmente posee el control del entorno de desarrollo de "Nios II", a continuación un vistazo rápido a la interfaz java de desarrollo sobre del procesador Nios II conocida como "Eclipse".

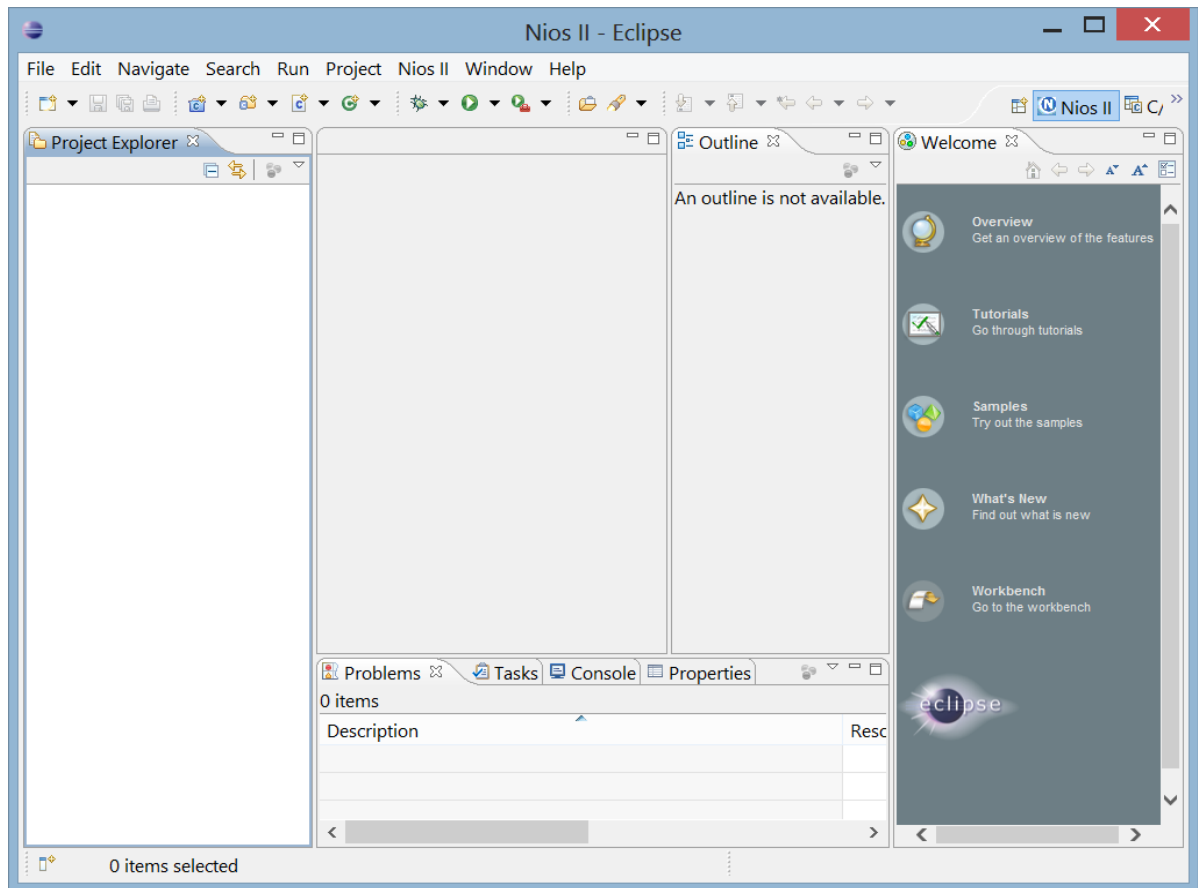


Figura 37. Entorno gráfico de desarrollo Nios II Eclipse.

Ya se puede entender un poco mejor la función gráfica del entorno en términos generales, la mayoría de configuraciones y cuestiones de edición en la creación de proyectos se podría trabajar solo a través de consola de fondo negro y esta interfaz no tendría sentido, sin embargo hay muchas otras características que es mejor mantenerlas sencillas y claras sobre una edición en entorno gráfico y por ende esta interfaz descrita a través de aplicación java se hace básica y crucial para evitar errores humanos.

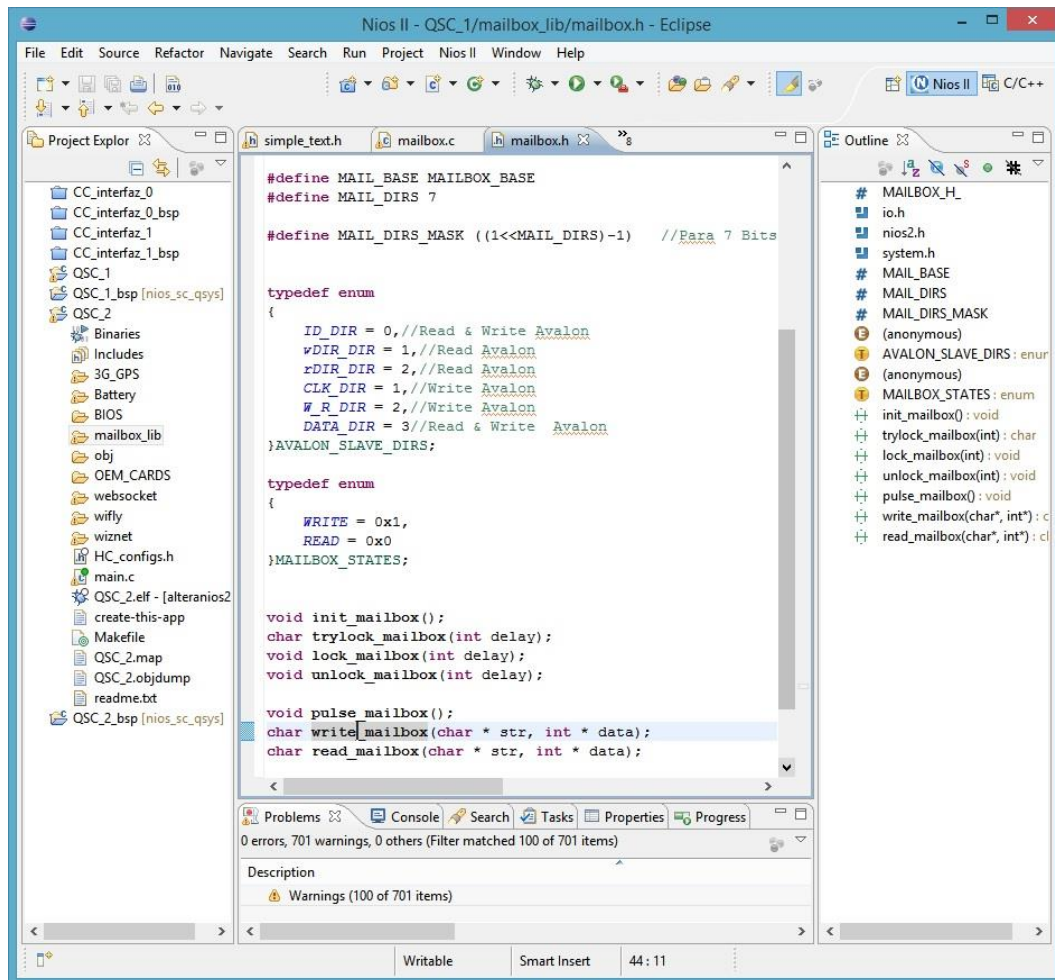


Figura 38. Entorno gráfico de desarrollo Nios II Eclipse – Edición de proyectos.

La Figura 38 emplea un ejemplo de edición de un proyecto software llamado “QSC_1” y otro llamado “QSC_2” sobre el entorno Eclipse; para este proyecto del monitor de signos vitales (MSV) se usan dos proyectos software por la misma razón de haber descrito dos procesadores en la lógica hardware previamente en el entorno Qsys, dos códigos software de dos procesadores diferentes que reparten sus tareas, se comunican a través del recurso “mailbox” y se explotan sus funcionalidades al máximo para lograr un sistema estable y rápido para con el usuario final del equipo médico.

En la Figura 38 se pueden observar otros dos proyectos para dos procesadores bajo los nombre de “CC_interfaz_0” y “CC_interfaz_1” pero no son relevantes para el desarrollo de esta práctica, ya que son un par de proyectos de prueba que cualquier desarrollador podría usar para irse guiando y realizando cierto nivel de investigación sobre los mismos sin necesidad de teorizar la situación e ir adquiriendo experiencia de manejo sobre el entorno; es importante repasar algo de teoría y asemejar los conceptos que se sostienen del entorno Nios II IDE sobre este nuevo entorno de desarrollo conocer sus diferencias y por ende ventajas y desventajas, así que es una práctica que recomiendo siempre en

compañía de los conocimientos de un tutor o persona experimentada en el tema que pueda resolver ciertas dudas que la teoría no puede lograr o por lo menos tomaría mucho tiempo en siquiera dar los primeros pasos de desarrollo sobre tan extensas bases.

De las habilidades de edición del entorno se pueden ver en su mayoría o por lo menos las que verdaderamente son usadas por el desarrollador en el menú desplegable que sostiene la selección de una sencilla palabra en el código a continuación.

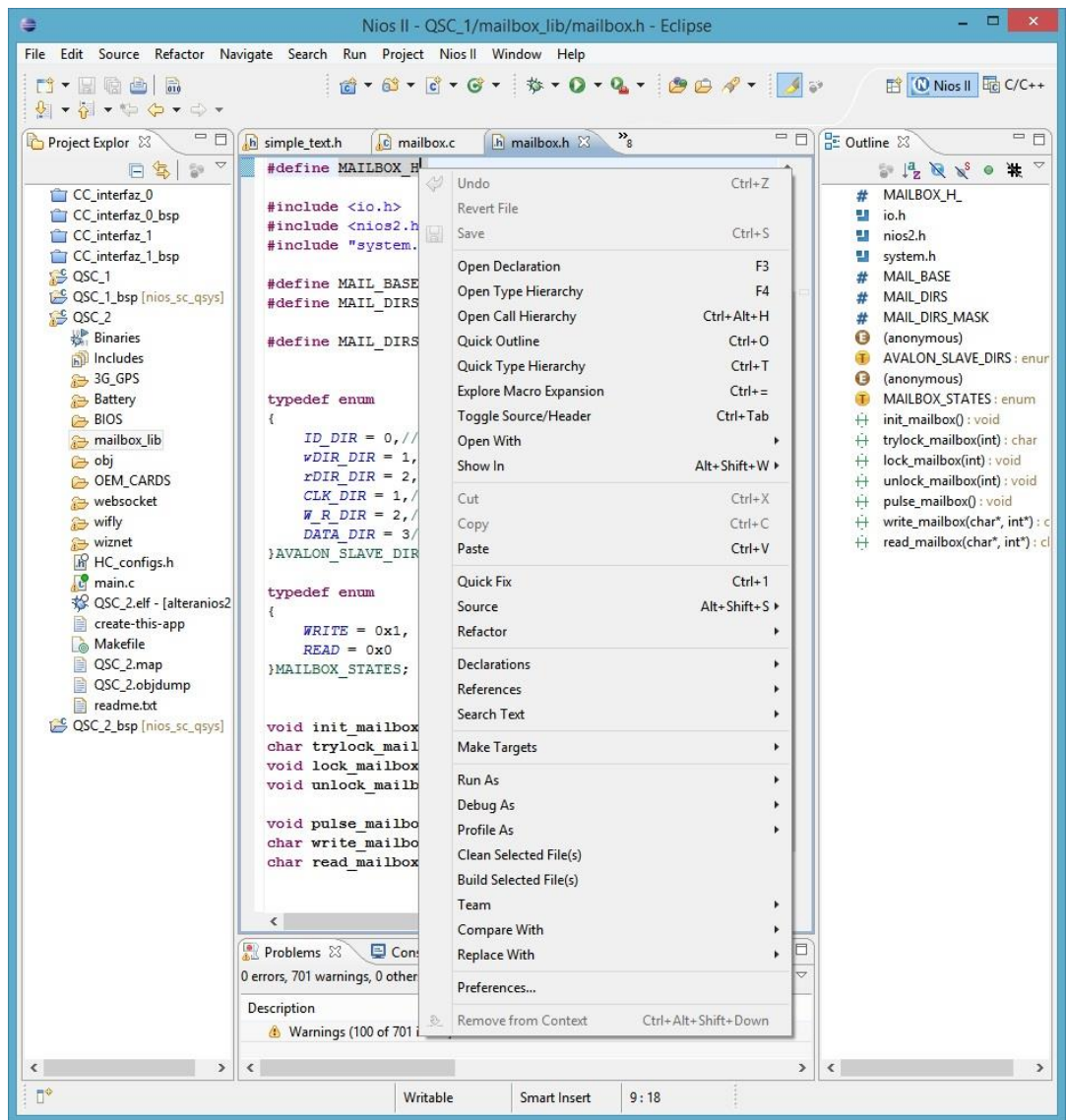


Figura 39. Entorno gráfico de desarrollo Nios II Eclipse – Opciones de búsqueda y edición.

Como se aprecia en la Figura 39 existen una serie de opcionalidades de búsqueda de declaraciones, referencias, texto etc., tanto en el proyecto actual como en todo el espacio de trabajo y los proyectos en el abiertos, movilidad entre el archivo fuente (*.c) y su

encabezado (*.h), vista rápida de las funciones existentes en la librería (Outline), función de refractor para el cambio global de los nombres de las funciones o macros en todas las declaraciones existentes y por último suele ser bastante útil saltar entre funciones y texto a través de teclas rápidas como “ctrl+clic”, “ctrl+shift+flechas”, “fin de línea”, “comienzo de línea”, entre otras las más usadas, así que es un gran editor de código software no solo en formato de C sino de C++ también, aunque por último tiene falencias de ser lento.

2.5.2 Creación y configuración de proyectos.

Habiendo comprendido las destrezas como editor, resta aprender lo concerniente a la creación y configuración de los proyectos funcionales que programaran ambos procesadores Nios II, lo cual no es muy difícil, ya que solo basta con entrar al menú “Archivo->Nuevo->Nios II Application and BSP from template”, asignar un nombre para el proyecto software, seleccionar una plantilla predeterminada para iniciar con una mínima base software y seleccionar el archivo resultante de la compilación hardware que define el desarrollo Qsys en extensión (*.sopcinfo), para así reconocer los procesadores existentes y luego finalizar el proceso de creación sin mayor complejidad.

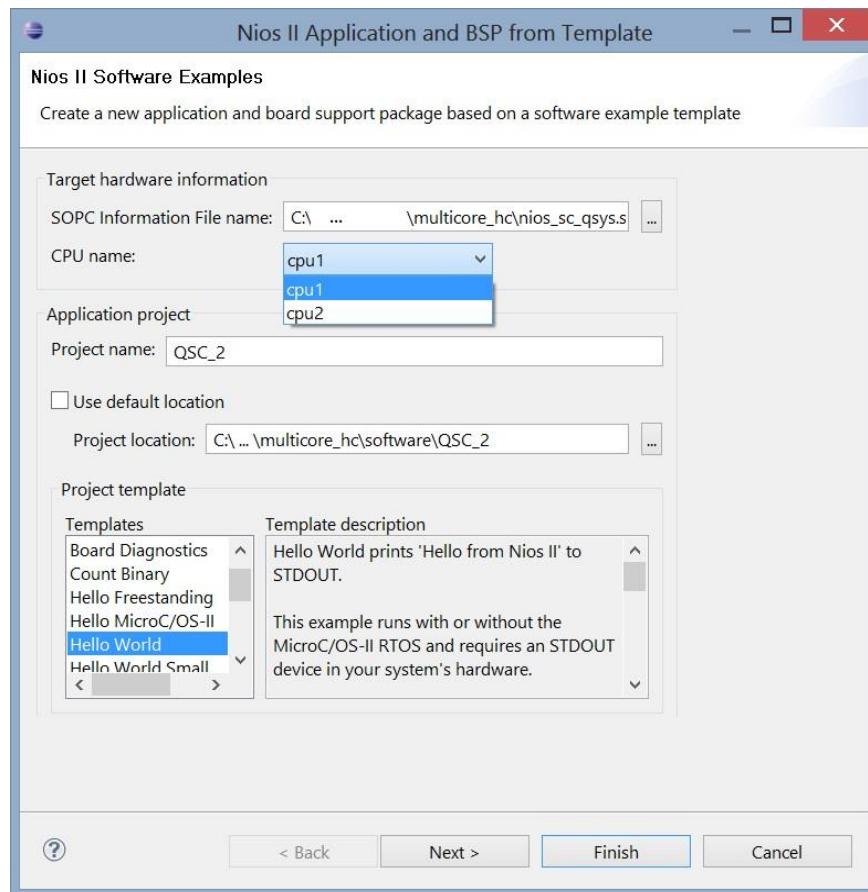


Figura 40. Entorno gráfico de desarrollo Nios II Eclipse – Creación de proyectos software en base a descripción hardware.

De modo predeterminado se crean dos carpetas siempre, la primera bajo el nombre asignado al proyecto la cual contendrá las librerías que el desarrollador quiera incluir para formar el sistema software del equipo médico. Por otro lado existe otra carpeta del sistema bajo el mismo nombre del proyecto pero con un agregado de nombre de “_bsp” (board support package), la cual contiene todas esas librerías del sistema, las cuales definen las funciones básicas de escritura y lectura para con los demás módulos descritos previamente en el entorno Qsys y que así pueda existir esa relación entre todas las interconexiones realizadas al procesador en su momento como se muestra a continuación.

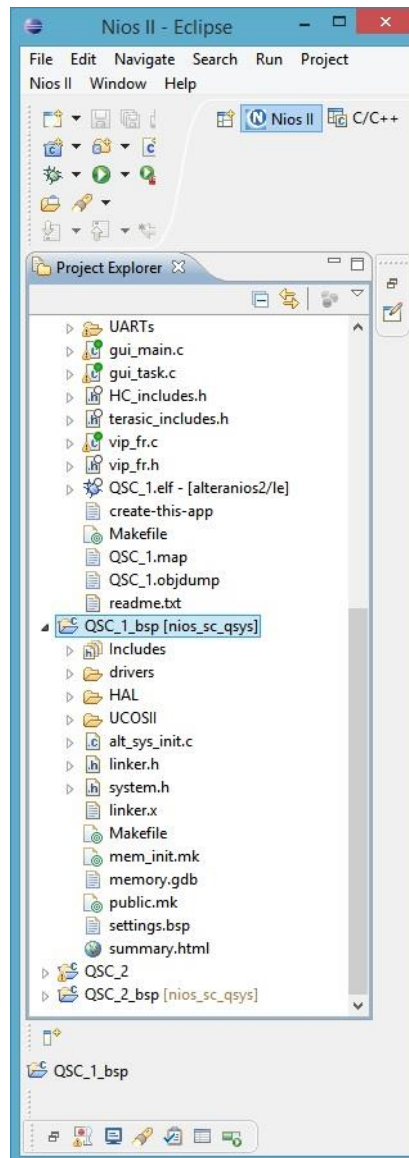


Figura 41. Entorno gráfico de desarrollo Nios II Eclipse – Proyecto software: carpeta de desarrollo y carpeta BSP básica de funcionalidades y definiciones propias del sistema que envuelve al procesador.

Como se puede observar en la Figura 41, existen una serie de librerías que opacan la aparición de la carpeta “QSC_1” debido a los desarrollos que se han venido haciendo con los aportes de varios ingenieros en el crecimiento de este proyecto y finalmente una carpeta de nombre “QSC_1_bsp”, la cual contiene una serie de librerías propias del sistema para lograr hacer uso de funciones de escritura y lectura hacia los módulos hardware por protocolo avalon previamente estudiando, entre otras definiciones de nombres y parámetros propias del sistema hardware ya compilado con nuevos cambios y correcciones. Todo lo anterior reduce al mundo hardware a una visión mucho más sencilla, en donde basta llamar una función desde el mundo software para escribir o leer un dato en los registros del mundo hardware usando protocolos avalon y todo lo concerniente a sincronismo y demás factores físicos en términos de comunicación. Por otro lado se puede observar como el segundo proyecto del segundo procesador también conserva sus respectivas carpetas “QSC_2” y “QSC_2_bsp” que tienen la misma estructura del primer proyecto de desarrollo software sobre el primer procesador.

Como última observación, hay ciertas características del mundo hardware que ya no pueden ser cambiadas a menos que exista una re-compilación del mismo a través de la funcionalidad reprogramable de las FPGAs, sin embargo existen ciertos factores aun configurables desde este entorno software que define las regiones de memoria para la ejecución del código, así como el uso de los diferentes componentes del entorno Qsys previamente descrito a disposición del mundo software para la ejecución de algunas tareas y servicios entre otras, esta configuración final se logra a través de este entorno Eclipse sobre la opcionalidad de edición de las características del “bsp” o “board support package” de manera más interactiva y menos manual, tal y como se muestra en la Figura 42.

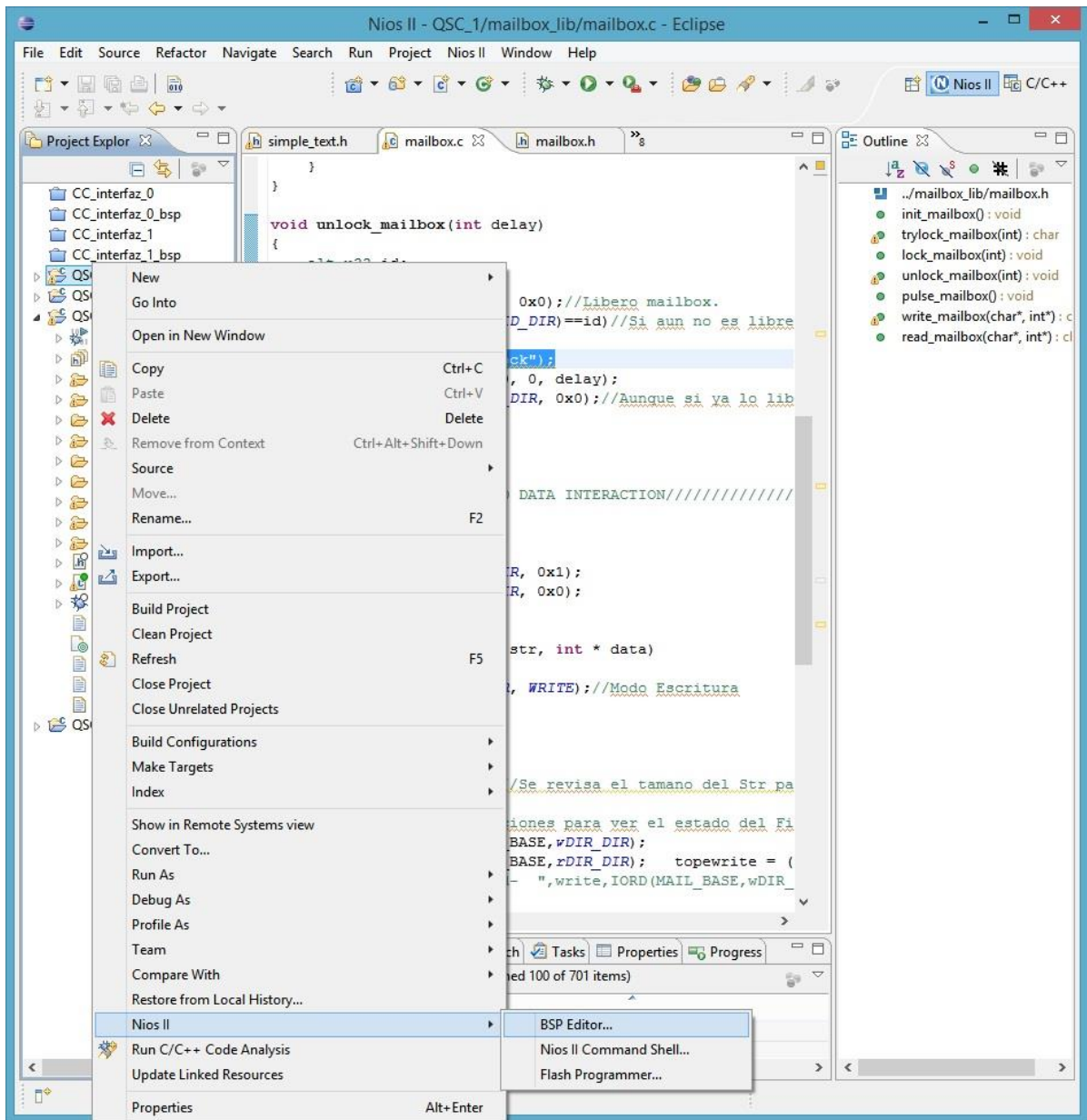


Figura 42. Entorno gráfico de desarrollo Nios II Eclipse – Edición final de las características de ejecución software a través de la herramienta de edición BSP.

La opción anterior desplegará un menú de edición para habilitar, deshabilitar y configurar variados parámetros de la ejecución software y que son claves para que el sistema funcione correctamente y no exista fallas de memoria o errores inesperados en la primera ejecución del sistema software en absoluto, este menú se compone de varias pestañas que se muestran y explican a continuación.

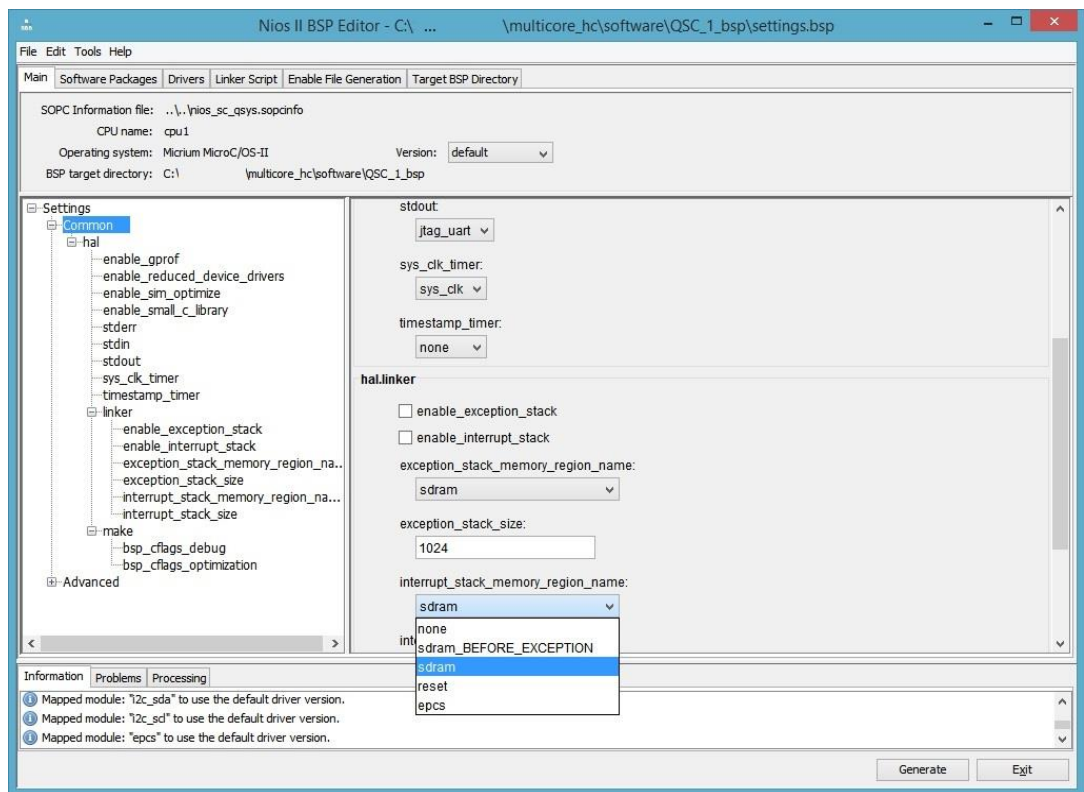


Figura 43. Entorno gráfico de desarrollo Nios II Eclipse – Edición BSP, pestaña principal de configuración básica.

Esta es probablemente la pestaña más importante, la cual resalta y configura los usos de las diferentes regiones de memoria para la ejecución de software, cabe aclarar que existen dos tipos de memoria: volátil (RAM) y no volátil (Flash – EPCS Altera), en las cuales el sistema podría ejecutarse y hay que definir muy bien cuales componen dichas regiones para evitar fallos de ejecución, también se pueden configurar los módulos dispuestos para hacer las funciones de “stdin”, “stdout” y “stderr” en este caso el módulo “jtag_uart” que enviará y recibirá comandos a través del USB que conecta la FPGA con el computador en función depuradora o “debugging”, los temporizadores del sistema para configurar tiempos y ejecución del sistema operativo a través de “timers” e interrupciones que en este caso se dejaron a cargo del módulo “sys_clk” descrito en el entorno Qsys, entre otras funcionalidades más avanzadas que se pueden configurar en el panel desplegable de la izquierda en la Figura 43 y otras más sencillas respecto a la activación de funcionalidades de las librerías del sistema operativo (uC/OS II), el cual se tratará más adelante; por ahora esta configuración se dejara predeterminada con las funcionalidades que ya posee de fábrica para cada nuevo proyecto software.

Otra pestaña considerada como segunda en importancia para el entendimiento de la dinámica software y el actuar del mismo a través de memorias y un procesador, es la pestaña “Linker Script”, ya que allí se pueden encontrar los diferentes valores que guían el desarrollo de la ejecución de tareas software así:

De la región de memoria en números hexadecimales “0x0” hasta “0x1F” de la memoria flash o EPCS física, se encuentra una región conocida por el procesador como “reset”, sin embargo estas regiones están camufladas bajo una “dirección relativa avalon” de referencia previamente estudiada que va desde la dirección “0x2000800” hasta “0x200081F” y con la cual el procesador se guía para incursionar sobre la cantidad de módulos que tiene a su disposición, cada módulo con sus regiones de referencia diferentes, de igual manera en la misma memoria flash no volátil (EPCS) desde la dirección “0x20” hasta “0x7FF” camuflada bajo sus equivalentes avalon “0x2000820” hasta “0x2000FFF” se encuentra la región conocida por el procesador como “epcs”.

Respecto a la memoria RAM, se tienen las direcciones desde la “0x0” hasta “0x7FFFFFFF” camufladas bajo las mismas direcciones relativas avalon, la región de memoria conocida como “sdram BEFORE EXCEPTION” y por ultimo las direcciones “0x800000” hasta “0x186A01F” camufladas bajo las mismas direcciones de formato avalon, la región conocida como “sdram”, la cual se usa para correr el sistema software en la pestaña de configuración principal. La razón de que las direcciones propias de la memoria RAM coincidieran con sus equivalentes avalon, se dio porque así se definió desde un comienzo en el entorno Qsys al asignar a esta memoria como la primera que en la lista de las direcciones relativas avalon, mientras que en el caso de la memoria EPCS se asignó más aleatoriamente.

Todo lo anterior se puede ver fácilmente ejemplificado en la Figura 44 que describe la información contenida en la pestaña Linker Script:

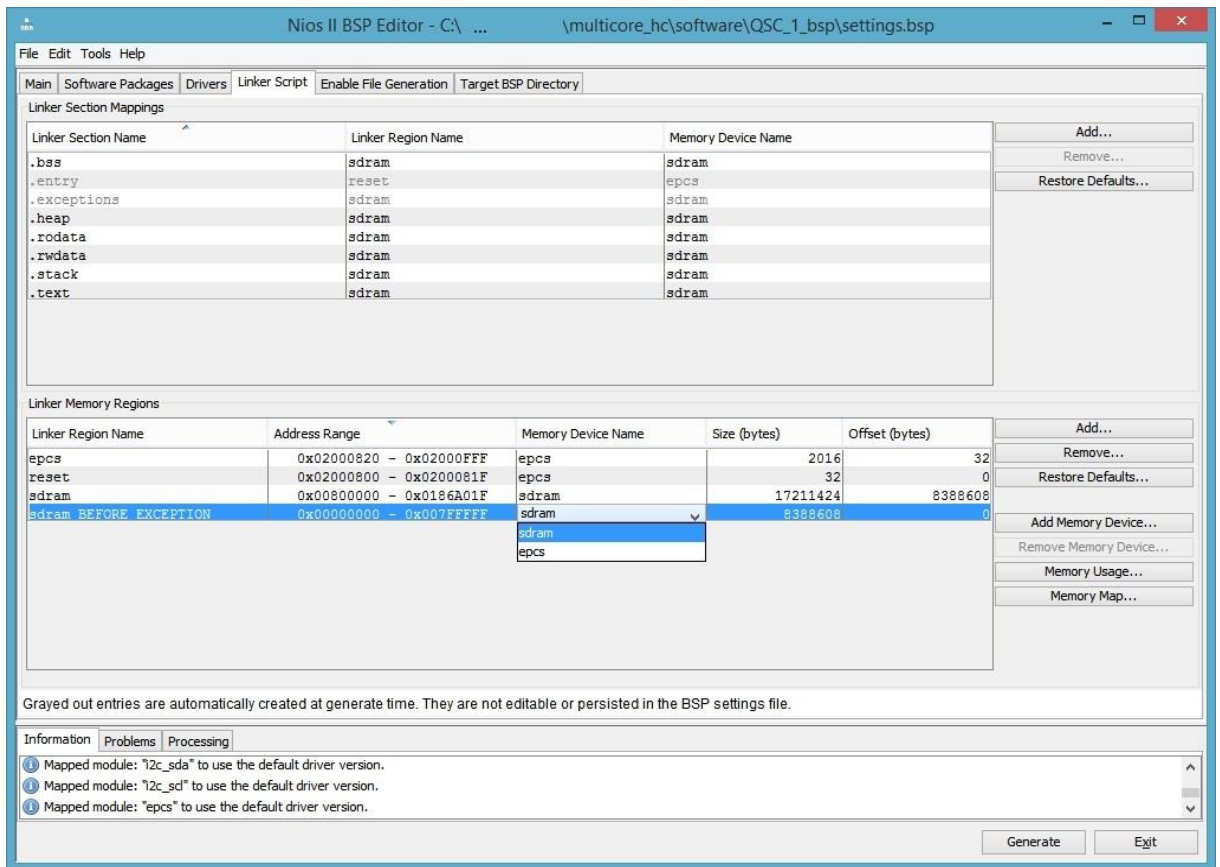


Figura 44. Entorno gráfico de desarrollo Nios II Eclipse – Edición BSP, pestaña de descripción de las diferentes regiones memoria del procesador.

2.5.2.1 Comprensión y uso de la distribución de memoria hardware prevista.

La importancia de entender este concepto de regiones de memoria, se ve más claramente ejemplificado en la Figura 45, en donde cada región tiene un fin y por ende un significado al momento de ejecutar el sistema software; para el entendimiento de la misma, ha de saberse que ambos procesadores utilizan la misma memoria volátil (RAM) y diferente memoria no volátil (ECPS y EPCS_two), además la memoria volátil, posee una región de memoria que no es utilizada por el sistema software (sdram BEFORE EXCEPTION) y por ende está a disposición del desarrollador, el cual la utilizará para fines destinados al video de pantalla que posee el equipo médico. Lo anterior distribuido así:

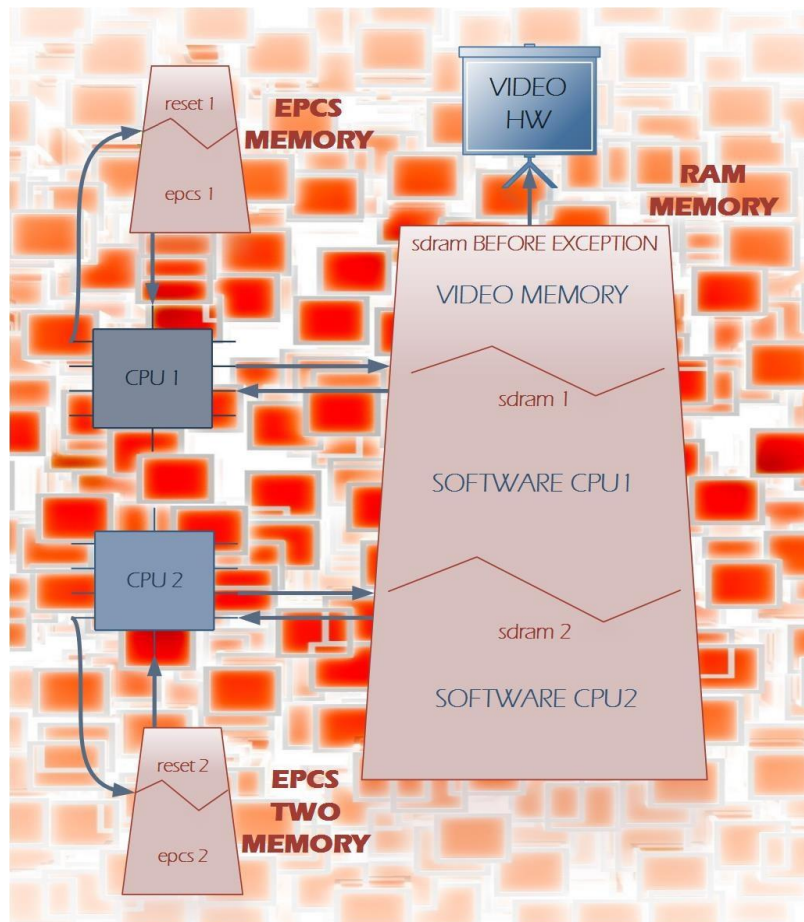


Figura 45. Repartición de las diferentes regiones de memoria existentes para el uso compartido entre los procesadores y el hardware de video dedicado.

2.5.3 Programación y verificación de la lógica software embebida en los proyectos.

Ya se ha configurado y entendido por completo la dinámica del sistema software a través del entorno "Eclipse", se tiene un entorno de trabajo disponible con un par de proyectos software para programar ambos procesadores con la lógica descrita en las carpetas "QSC_1" y "QSC_2", restando compilar y programar tal código software, esta es la razón de que exista la interfaz de desarrollo Eclipse y sus funciones de construcción, así como otras de forma y orden apreciables en el menú desplegable de la Figura 42; en esta misma ilustración se puede observar junto a la opción de "BSP editor", las herramientas "Nios II command shell" y "Flash programmer", las cuales permitirán descargar la compilación total de cada uno de los software y hasta hardware programable de la FPGA de manera no volátil a las memorias EPCS destinadas para ello, así como verificar que dicha programación sea ejecutada correctamente usando la opción de "debug" previamente definida como "stdout", "stdin" y "stderr" a través del módulo físico "jtag_uart", el cual funciona como un terminal de "prints" o impresiones a través del USB conectado desde la FPGA al computador, haciendo uso de protocolo "UART" para transmitir los

diferentes mensajes que sean necesarios, tema que se tratará dentro de muy poco en la descripción de la herramienta “Nios II command shell”.

Por el momento se pretende programar de manera no volátil los diferente archivos de compilación final tanto hardware (multicore_hc.sof), como software, este último referido al archivo de compilación software que se alcanza a apreciar en la Figura 41 como “QSC_1.elf” y “QSC_2.elf” correspondiente al segundo procesador que no se alcanza a apreciar en esta figura; estos tres archivos de compilación final son los que toma la herramienta “Nios II Flash Programmer” previamente mencionada para escribir sobre las diferentes memorias no volátiles.

A continuación se presenta la herramienta de programación flash de los procesadores Nios II, la cual pretende extraer la información de los procesadores a programar a través del archivo de extensión (*.sopcinfo).

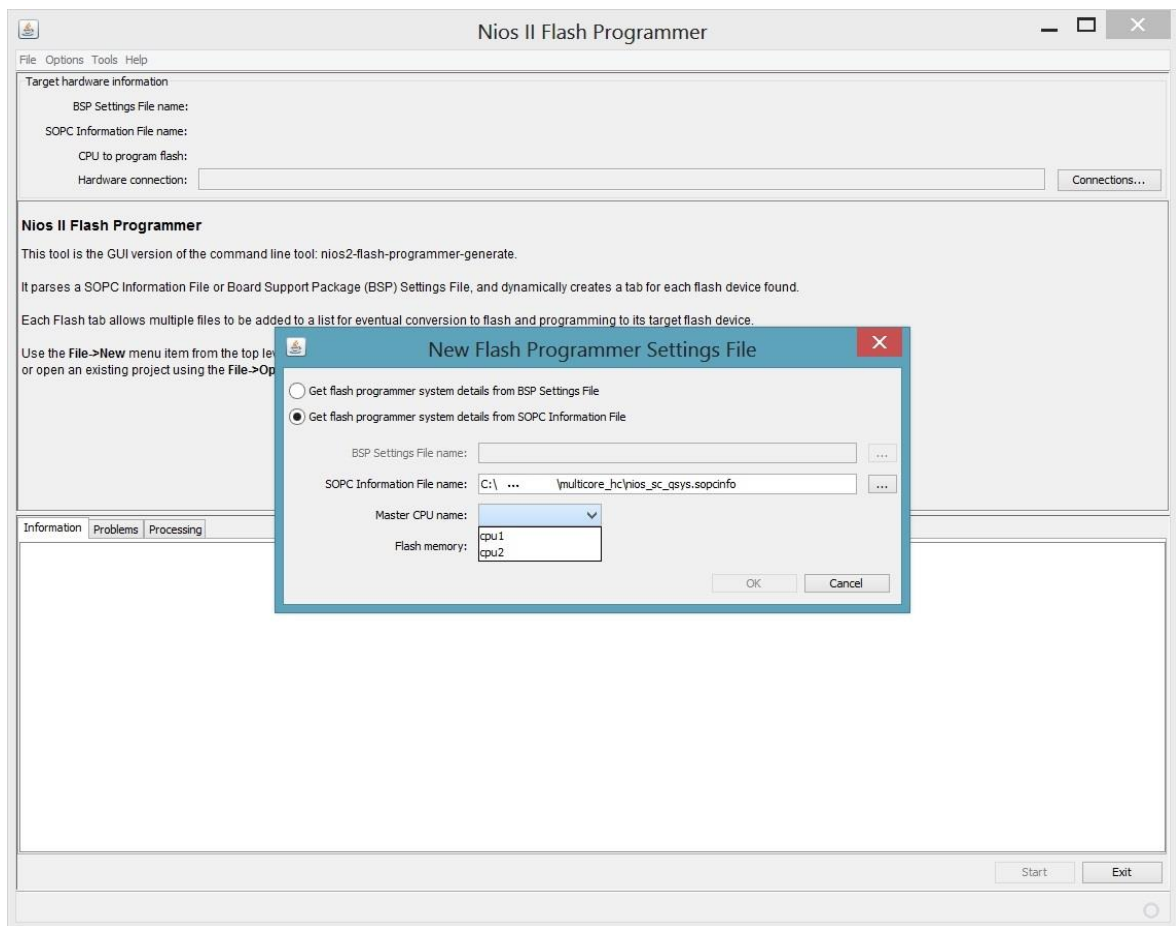


Figura 46. Interfaz gráfica de programación software de los procesadores Nios II.

Como se puede apreciar en la Figura 46 se extrae la información de los procesadores, se elige el que se desea programar y se establecerá la guía de programación automáticamente en esta herramienta, solo resta agregar (add) los archivos de compilación que cada memoria no volátil debe sostener; para el caso del primer procesador (CPU1) asociado a la primera memoria (EPCS), se debe programar tanto el archivo de compilación hardware (multicore.sof), así como el archivo de compilación software del primer procesador (QSC_1.elf) tal y como se observa a continuación.

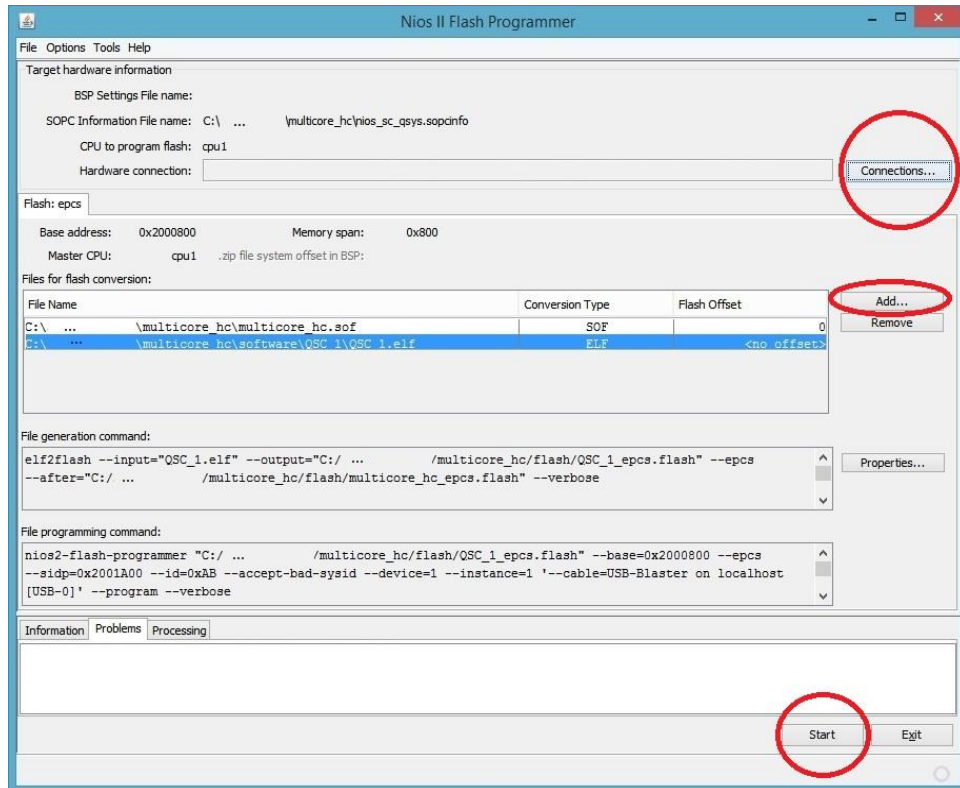


Figura 47. Ejemplo de programación no volátil sobre la herramienta “Nios II Flash Programmer”.

Estos archivos de compilación son transformados a otros archivos en formato hexadecimal de extensión “*.flash”, los cuales programaran el procesador indicado con ayuda de la información del archivo de extensión “*.sopcinfo”, mientras coincida con el procesador programado en una FPGA conectada a través de USB al computador, este último reconocimiento se logrará a través de esta herramienta en el botón superior derecho de “connections”. Cabe aclarar que la FPGA debe estar programada de forma volátil o no volátil con el archivo de compilación hardware “multicore_hc.sof” al momento de ser programada para lograr este reconocimiento, ya luego se creará una programación no volátil al clicar en la opción “Start” de la Figura 47, programación que siempre estará activa para la FPGA al momento de ser energizada ya sea vía USB o de cualquier otra manera.

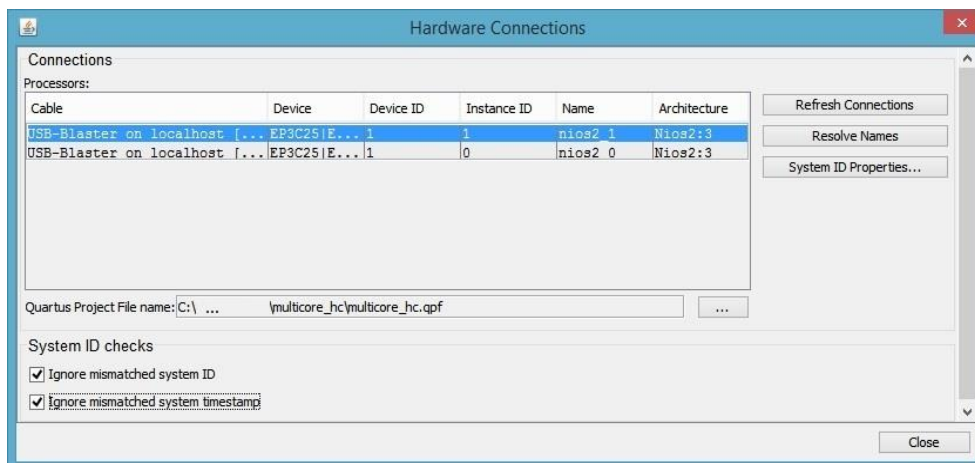


Figura 48. Verificación de la existencia de los procesadores hardware a programar de manera software dentro de la FPGA.

La herramienta de reconocimiento de los procesadores existentes en la FPGA (Hardware Connections) luce como se muestra en la Figura 48 y utiliza parámetros como “system ID” y “system timestamp” para verificar que el hardware programado en la FPGA sea equivalente al que se le desea programar el software indicado, sin embargo se pueden ignorar si se tiene la certeza de que aunque no es la programación hardware correcta, se sabe que la nueva programación hardware (*.sof) no volátil, complementará perfectamente a la programación software que permanecerá en la memoria (*.elf).

En el caso de la segunda memoria no volátil (EPCS_TWO) no es tan complejo, ya que solo albergará la programación software del segundo procesador “QSC_2.elf” y en definitiva deberá coincidir con la instancia del segundo procesador resultante de la pestaña “Hardware Connections”, la cual siempre debe ser seleccionada (Figura 48) antes de iniciar la programación de cualquier memoria no volátil (Figura 47).

Como último recurso se tiene la herramienta “Nios II command shell”, quien aunque funciona como compilador, constructor y hasta programador de los diferentes archivos de compilación de forma manual, es usualmente utilizada para verificar la correcta ejecución de la programación software a través de la información que pudiese brindar el código mismo que envía mensajes al hacer uso del estándar de su módulo “jtag_uart” programado en la FPGA así como protocolo “UART” a través del USB conectado al computador en forma de “debugging”, mensajes “UART” que son impresos más cómodamente a través de esta herramienta de comandos y la función “nios 2 terminal” que ofrece la compañía Altera entre sus muchas funciones de consola de comandos; un ejemplo la función de este terminal “UART” se muestra en la Figura 49.


```
0 [main] bash 6776 find_fast_cwd: WARNING: Couldn't compute FAST_CWD point
er. Please report this problem to
the public mailing list cygwin@cygwin.com
-----
Altera Nios2 Command Shell [GCC 4]
Version 12.1sp1, Build 243
-----
FCU@adriz /cygdrive/c/ ... /multicore_hc/software/QSC_1
$ nios2-terminal --instance=1 --cable=1
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-01]", device 1, instance 1
nios2-terminal: <Use the IDE stop button or Ctrl-C to terminate>

Mail Free! 125, Texlen! 9
card 25mhz
card 25mhz
image 0
image 1330
??;image 1330
image 417
image 417
image 474
image 474
image 471
image 471
image 478
Nuevo estado de bateria = 98
BATTERY Saved 5/54/28..Status=98..Carga=?

EUEENT Saved .. M:M_ON

Init ChargeMenu
Showing ChargeMenu
Drawing ChargeMenu
Ready Goldwei
PARAM Saved 8/8/14 5-54-30
PARAM Saved 8/8/14 5-54-30
PARAM Saved 8/8/14 5-54-30
PARAM Saved 8/8/14 5-54-30

Ready RTC
Goldwei SUPER!!!!
Init... Main
01..01..2000..
```

Figura 49. Consola de comandos “Nios II command Shell”.

Así como se realiza la visualización de envíos “stdout” desde el procesador hacia el computador gracias a la función “nios 2 terminal” también se pueden usar una serie de comandos muy útiles para la ejecución de compilaciones, programaciones y verificaciones de todo tipo sin el uso de otras herramientas, por ende es mayormente utilizada por los ingenieros en los desarrollos actuales dado a su rapidez de ejecución, motivo por el cual los entornos gráficos de desarrollo se omiten únicamente para la edición y creación de los proyectos software o hardware más que para programaciones compilaciones y debugging, ya que la consola de comandos de la Figura 49 tiene mayor cobertura de estas funciones y se facilita su manejo a través de la descripción de algunos “scripts” que usan los ingenieros en la FCV.

2.6 EDICIÓN SOFTWARE SOBRE LA HERRAMIENTA QT CREATOR

2.6.1 Presentación de la herramienta.

Es claro que ya se poseen todos los detalles y estudios necesarios para generar todo tipo de sistemas hardware, software e interacciones entre estos, así que es momento de

empezar a abarcar el mundo software de manera más amplia y enfocada al desarrollo, dejando de lado los temas técnicos; es por esta razón y dado a la evidente cantidad de librerías y desarrollo actual sobre el entorno software, que se hace necesario explorar mejores opciones de edición del código, dejando atrás lo referente al entorno Eclipse para que surja la herramienta “Qt Creator”; solo basta hacer un par de búsquedas e investigaciones en línea sobre esta herramienta de edición, para evidenciar que es bastante completa e intuitiva con el programador, sus colores, personalización, fácil manejo y búsqueda en movimiento de todo tipo de variables, funciones y demás, la hacen tan atractiva para editar. Como un abre bocas a la misma se plantea la Figura 50.

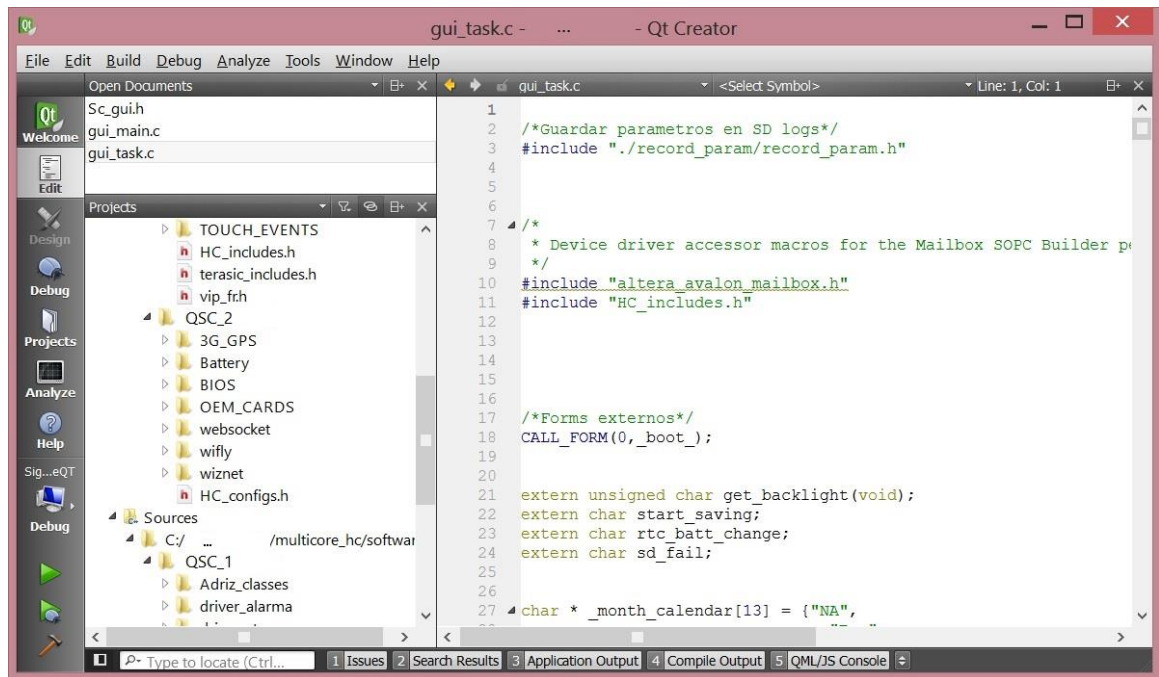


Figura 50. Herramienta de desarrollo software “QT Creator”.

Es a partir de esta herramienta que se han logrado las diferentes modificaciones al sistema software de ambos procesadores existentes en la FPGA, luego se usa el entorno Eclipse para compilación y verificación de errores de manera más interactiva y por último la consola de comandos quien hará las veces de terminal UART y programador al mismo tiempo, reemplazando a la herramienta gráfica del “Nios II Flash programmer” para usar la versión de comando.

2.6.2 Edición software de las librerías controladoras.

Se empiezan modificando las diferentes librerías de más bajo nivel del sistema software, conocidas como “controladores” de aquellos sistemas hardware previamente descritos en el entorno Qsys, estos controladores necesitan ser modificados por la misma razón que los módulos hardware fueron modificados para resolver problemas de estabilidad en la comunicación interna, externa, el video de pantalla y la integración de los diferentes

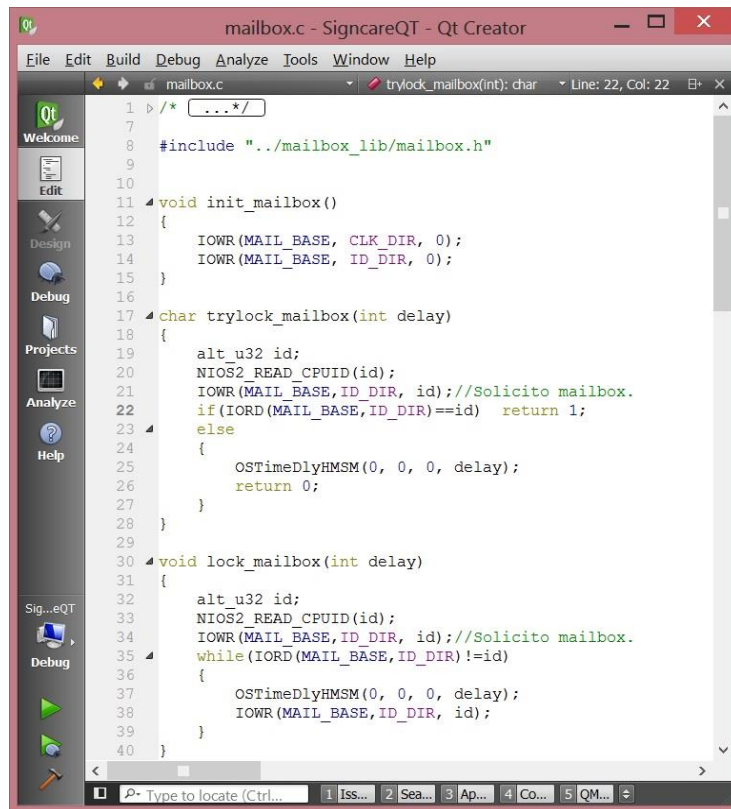
periféricos que tuvieron lugar sobre la plataforma Qsys como se alcanzan a evidenciar en la Figura 25; son modificaciones y avances software que pretenden complementar el actuar de los módulos hardware y no pueden ser reveladas en su totalidad debido a la confidencialidad que la organización maneja.

Las librerías software referidas a complementar la funcionalidad hardware son orientadas a la compatibilidad con las funciones físicas de estos módulos; por otro lado se evidencia al módulo de comunicación entre procesadores “mailbox” como completamente nuevo, de propia autoría, carente de revisión y creación de sus librerías controladoras, es así que el proceso de modificación de estas librerías puede ser ejemplificado muy bien a partir del código sobre la Figura 38, en donde se describen siete (7) funciones básicas de uso y un par de estructuras software que definen constantes relacionadas con la propia estructura hardware del módulo previamente descrito en lenguaje Verilog y la herramienta de edición de componentes de la Figura 27. Es así que el módulo mailbox funciona de la siguiente manera:

En primer lugar el módulo mailbox si puede ser editado en su bandeja de mensajes por cualquiera de los dos procesadores y en cualquier momento de ser necesario, es por ello que ambos procesadores deben poseer esta librería y respetar una serie de permisos antes de acceder a tal bandeja, evitando así la acción paralela de ambos procesadores sobre tal módulo; para ello se utiliza la función del módulo hardware “mutex” encargado de recibir un identificador “ID” proveniente de cualquier procesador y con ello gestionar el control temporal del mailbox a cada procesador, es así que mientras el mailbox esté siendo usado por un procesador, el otro debe esperar su turno antes de dejar un mensaje en la bandeja de entrada (memoria On-chip). Una vez que el procesador posee el mailbox, puede decidir si dejar un mensaje o revisar su parte de bandeja de entrada o sencillamente liberar el mismo a través de un identificador nulo en el mutex, permitiendo que otro procesador pueda acceder y tome el control para leer un mensaje u otra decisión autónoma respecto a su actuar en dicho mailbox.

Toda esta secuencia de tomas de control se repite una y otra vez para que exista dicha comunicación entre procesadores; funciones de control de bloqueo (lock) y desbloqueo (unlock) que posee la librería que interactúa directamente con el módulo hardware a través de protocolo avalon, no sin antes ejecutar algunas inicializaciones correspondientes en el módulo a través de la función “init_mailbox” (Figura 38), cada vez que sea encendido el dispositivo de monitorización vital. Ya para culminar con un proceso más sencillo de escritura y lectura hacia la memoria On-chip que a su vez maneje los respectivos bloqueos y desbloqueos ya explicados, se crean las funciones “read_mailbox” y “write_mailbox”, las cuales traducen toda la complejidad del proceso en la simple acción de recibir o escribir un vector de caracteres más conocido como “string” al módulo de mailbox directamente.

Parte del código fuente (.c) que describe esta librería controladora con encabezado (.h) en la Figura 38 se muestra a continuación.



```
1 > /* ... */
2
3 #include "../mailbox_lib/mailbox.h"
4
5
6 void init_mailbox()
7 {
8     IOWR(MAIL_BASE, CLK_DIR, 0);
9     IOWR(MAIL_BASE, ID_DIR, 0);
10 }
11
12 char trylock_mailbox(int delay)
13 {
14     alt_u32 id;
15     NIOS2_READ_CPUID(id);
16     IOWR(MAIL_BASE, ID_DIR, id); //Solicito mailbox.
17     if(IORD(MAIL_BASE, ID_DIR)==id) return 1;
18     else
19     {
20         OSTimeDlyHMSM(0, 0, 0, delay);
21         return 0;
22     }
23 }
24
25 void lock_mailbox(int delay)
26 {
27     alt_u32 id;
28     NIOS2_READ_CPUID(id);
29     IOWR(MAIL_BASE, ID_DIR, id); //Solicito mailbox.
30     while(IORD(MAIL_BASE, ID_DIR)!=id)
31     {
32         OSTimeDlyHMSM(0, 0, 0, delay);
33         IOWR(MAIL_BASE, ID_DIR, id);
34     }
35 }
```

Figura 51. Parte del código fuente de la librería controladora sugerida para el módulo mailbox “mailbox.c”.

Desarrollos de este tipo son los que abarcan las correcciones de las diferentes librerías controladoras para gestionar el control de ventanas de la interfaz gráfica, reinicios de video que cambian los diferentes espacios o “frames” de mayor resolución para la representación de los diferentes entornos gráficos en el monitor final y otros arreglos de copiado de memoria en el guardado de “sprites” o depósito de figuras que dibujan los botones y menús por toda la GUI. Desarrollos que van paso a paso bajo una metodología de implementaciones, pruebas y rediseños que terminan subidos al servidor respectivo con la facilidad de administración que presta la plataforma “SourceTree” previamente descrita.

2.7 COMPRENSIÓN SOFTWARE DE LA DINÁMICA DEL SISTEMA OPERATIVO

Ya se ha comprendido parte del desarrollo software, sin embargo en la Figura 51 también se pudo notar la utilización de la función del sistema operativo uC/OS II “OSTimeDlyHMSM”, es entonces momento de comprender a fondo la dinámica del sistema operativo antes de continuar con la descripción de los desarrollos, ya que el sistema software embebido vendrá trabajando dicho OS en gran parte del código actual

para un mejor aprovechamiento de la ejecución software, sin entrar en complejos desarrollos sobre máquinas de estado semejantes a un OS, pero que no estarían exentas de fallas humanas.

2.7.1 Uso del sistema operativo uC/OS II como herramienta útil en la ejecución de las tareas software.

Para resumir rápidamente lo que abarca un sistema operativo de sistemas embebidos como lo es uC/OS II, se podría decir que es un juego de librerías con funcionalidades especiales para lograr que un sistema software compuesto por los recursos dados por su respectivo procesador a cierta velocidad, sean también aprovechados para la ejecución de múltiples proceso o hilos “al mismo tiempo”, esto se logra a través del uso de los “timers” o temporizadores internos en dicho procesador, los cuales logran ejecutar varias funciones “al mismo tiempo” al realizar ejecuciones secuenciales que saltan entre los diversos hilos a través de interrupciones que administra el OS mismo, es por ello que no se trata realmente de una ejecución paralela (al mismo tiempo) como si se realiza en el mundo hardware, sino de una única ejecución secuencial saltando por variadas funciones o hilos que simulan paralelismo bajo la administración que pueda suministrar el OS. Algunos parámetros pueden ser asignados por el desarrollador al momento de utilizar funcionalidades especiales de OS que lo hacen bastante ventajoso respecto a la distribución y el gasto software en la escritura de un código de tal escala sin mayores preocupaciones de ejecución.

Con la creación de múltiples hilos y por ende ejecuciones desordenadas entre funciones que utilizan los mismos recursos hardware descritos en el sistema Qsys, se presentan nuevos problemas de ejecución, ya que si una tarea decide acceder a un recurso como lo podría ser guardar datos hacia una memoria SD y es interrumpida por otra tarea que también desea acceder al mismo recurso, existiría corrupción de datos y algunos otros problemas en el sistema de archivos de la misma memoria SD que podrían dejar sin funcionalidad a dicho recurso, es por esta razón que el sistema operativo posee muchos otros recursos software para la prevención de estas situaciones como lo son los recursos de exclusión mutua (OSMutex), semaforos (OSSem), timers (OSTmr), tareas (OSTask), tiempo (OSTime), memoria (OSMem) entre otros menos usados por el momento en la programación software del equipo médico; para entender más a fondo cada función y sus respectivas características y condiciones de ejecución, basta ir a la página oficial de los desarrolladores del sistema operativo “Micrium”, buscar y estudiar el respectivo manual de usuario.

A continuación una pequeña abstracción de la dinámica de la lógica implícita en las librerías del OS.

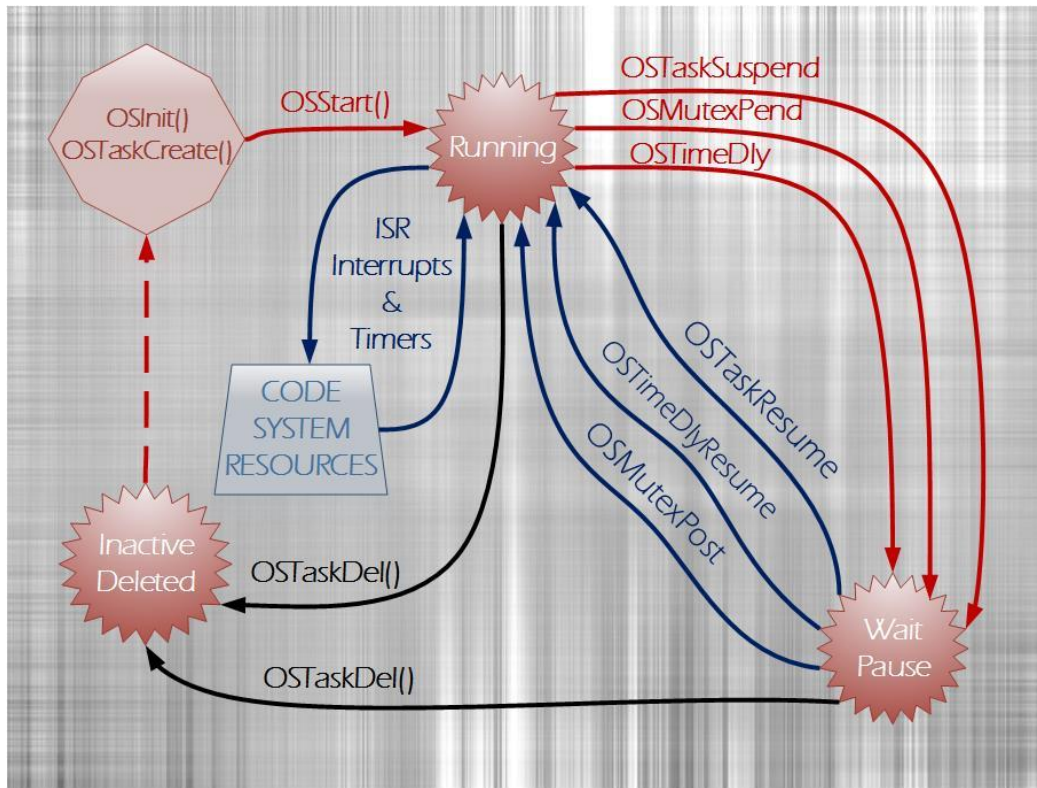


Figura 52. Dinámica de operación del sistema operativo uC/OS II de la compañía Micrium.

Este sistema operativo tiene compatibilidad con los desarrollos software tanto de la compañía Altera así como otras compañías de embebidos y la inclusión de sus librerías siempre puede realizarse de manera manual y si es necesario bajo el soporte de la compañía Micrium, una vez el producto es adquirido comercialmente por supuesto, sin embargo tanto el previo entorno de desarrollo IDE, así como el entorno Eclipse, poseen la funcionalidad del agregado automático de estas librerías, una vez sea seleccionada correctamente la plantilla o "Project Template" correcta referente al sistema operativo uC/OS II en el proceso de creación del proyecto (Figura 40), esta plantilla agregará las librerías correspondiente al uC/OS II en la carpeta UCOSII que se alcanza a apreciar en la Figura 41 sobre la carpeta del sistema que contiene información hardware de la tarjeta o "board support package (_bsp)"; de manera automática para esta plantilla también se crea un único archivo principal que contiene la función principal "main" de toda ejecución software, junto a un ejemplo de creación de tareas del sistema operativo "multi-tasking" y una inclusión del archivo de encabezado respectivo que hace la referencia a estas librerías del OS, lo cual impulsa bastante rápido el entendimiento y uso del mismo y aventaja a dichos entornos al momento de iniciar un proyecto con plantilla de UCOSII.

2.7.2 Dinámica software de una GUI junto al sistema operativo uC/OS II y lenguaje orientado a objetos (C++).

Ahora bien ya se tiene la herramienta de sistema operativo y se tiene la capacidad de

edición en código C a través de la práctica previamente realizada sobre las diferentes librerías controladoras, es entonces momento para ingresar en el campo de la programación software referida a objetos a través de código C++; esta nueva herramienta de desarrollo software fue previamente estudiada y explicada de manera abstracta a desde su concepto teórico, sin embargo se verá desde la perspectiva práctica para la creación de los objetos básicos que componen la base software de la interfaz gráfica de usuario (GUI), la cual sostiene actualmente al equipo médico.

La interfaz gráfica de usuario GUI, funciona a través de las clases que proporciona el lenguaje C++ para describir los objetos gráficos que el usuario final ve en pantalla, estos objetos tienen una serie de características propias que les permiten ser dibujados, borrados, accionados y refrescados en pantalla, dependiendo de cómo fueron descritos en sus características intrínsecas, esta interacción junto con otro tipo de características básicas de interacción de los recursos hardware, les permiten adquirir los diferentes datos vitales e historial de registros en memoria SD para mostrar en pantalla para con el usuario a través de un “Touch Screen” resistivo que posee esta versión del monitor de signos vitales como se explicaba a inicios de esta práctica.

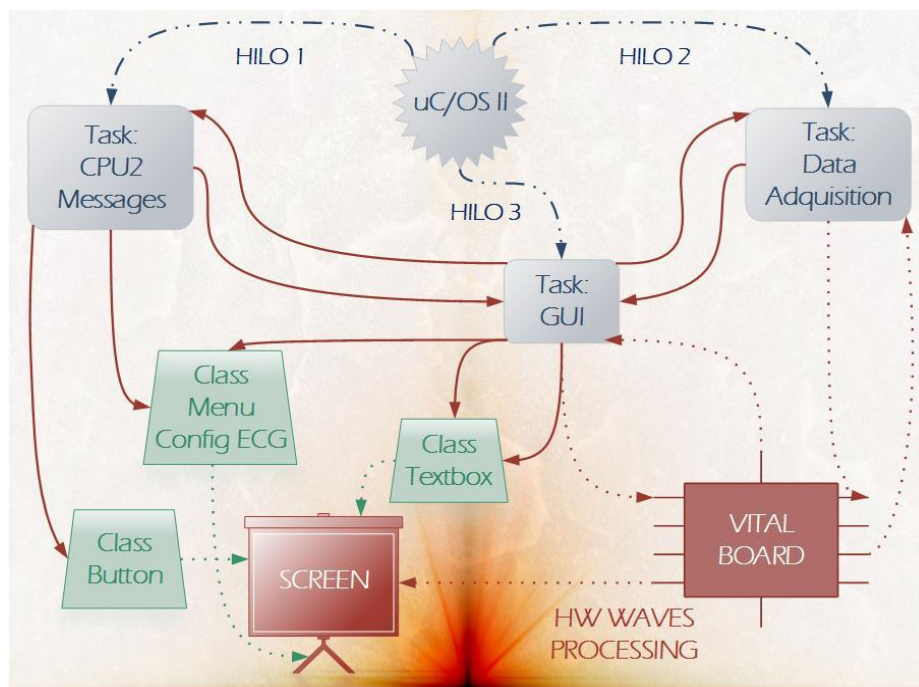


Figura 53. Dinámica de operación software del primer procesador.

La interacción entre el dibujo de pantalla y los datos que se adquieren del exterior se logra a través de la acción del sistema operativo, quien posee una tarea específica para la actualización de las imágenes, otra para la ejecución de lecturas y guardados a memoria SD, otra para la actualización de hora y fecha del sistema, otra para la lectura de los valores estáticos de la tarjeta externa de adquisición vital, otra para generar el arranque e inicio del sistema bajo modos de batería y modo de inicio normal, otra para el control de la luz de fondo o “backlight” en secuencias que lo requieran, otra para la revisión de los

mensajes entre procesadores (Librería mailbox), entre otras varias acciones de actualización para el primer procesador.

Sea labor de las tareas del OS y el programador los encargados de gestionar y usar los recursos hardware que provee la tarjeta comercial de signos vitales desde la perspectiva software y complementar la integración previamente descrita sobre el entorno Qsys para con los dispositivos externos; recursos hardware que se unen bajo la acción de puertos virtuales (PIO parallel input / output) y otros recursos conectados desde y hacia la FPGA con el procesador por el mismo bus de protocolo avalon previamente descrito como se ejemplifica en la Figura 54.

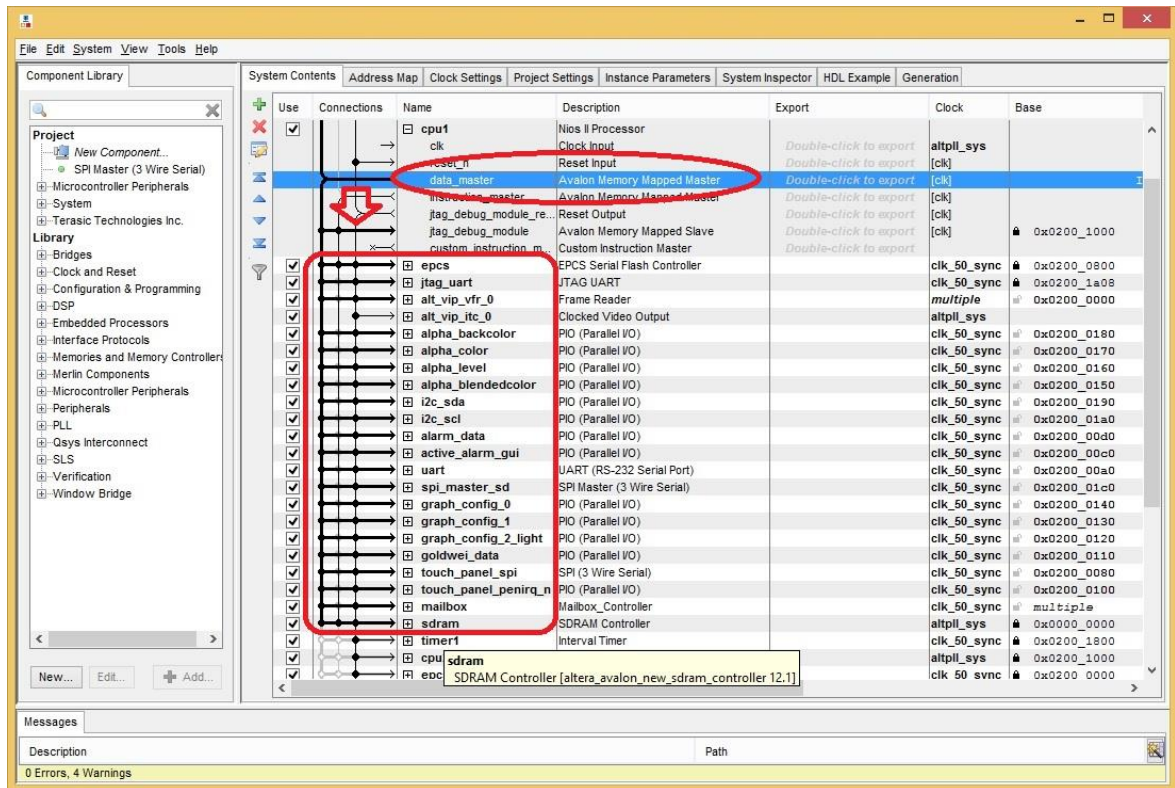


Figura 54. Dinámica de operación software del primer procesador.

Es el primer procesador el encargado de gestionar la función gráfica de signos vitales y en donde se centran los desarrollos software de gestión de ventanas, optimizaciones y fluidez apreciable a través del manejo del equipo mismo e informes de avance entregados a la FCV en el proceso de mejora de esta interfaz, la cual poseía demasiados desfases visuales, fallas de memoria de todo tipo, lentitud en el dibujado y otros factores en los que se centró la implementación de mejoras software de esta práctica.

2.7.3 Dinámica software de redes junto al sistema operativo uC/OS II y lenguaje orientado a objetos (C++).

Por otro lado existe un segundo procesador que aunque no utiliza este lenguaje de objetos muy a fondo dado a que le resulta más adecuado el código C en muchas de sus rutinas de sistema operativo, se encarga de otras labores de comunicación vía Internet, comunicación con la BIOS del sistema, chequeo de mensajes provenientes del primer procesador entre otras funciones adicionales de la capa de aplicación en cuanto a protocolos y paquetes de red que utilizan código C++ como puente entre los diversos estados de conexión del equipo (Figura 55). Cabe aclarar que este segundo procesador no tiene mayor impacto en la labor designada para esta práctica, sin embargo se ejemplifica con el fin de extender los usos del lenguaje referido a objetos C++, como una poderosa herramienta de desarrollo sobre todo en las situaciones más complejas, por otro lado tiene impacto en cuanto a la integración de periféricos de red con el softcore Nios II para el envío de comandos de red a través de chip comerciales como lo son el “Wiznet”, “Wifly” y el “SIM5218” para comunicación 3G.

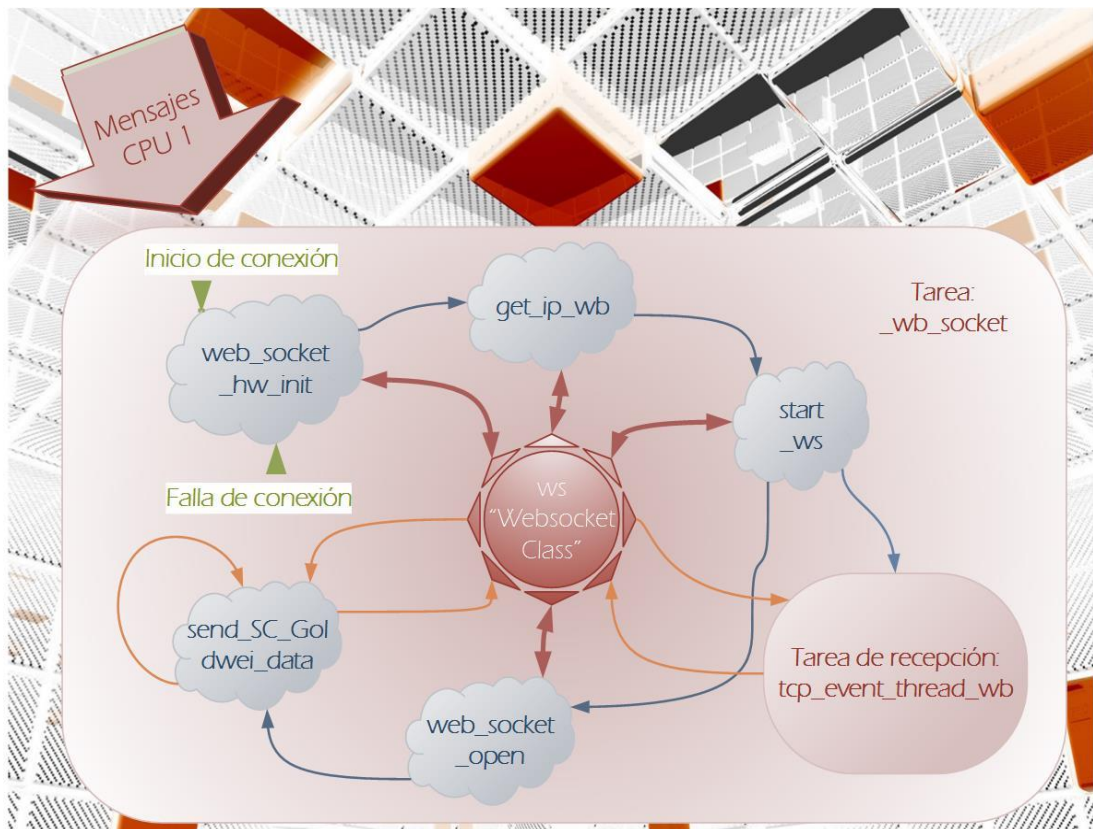


Figura 55. Dinámica de operación de red sobre el software del segundo procesador.

Debe destacarse que para el rápido desarrollo de esta múltiple interactividad, el sistema operativo que se viene usando juega un papel muy importante en la ejecución de los hilos que puede administrar, sin embargo no resuelve muchos de los problemas de filtraciones

de memoria, optimización del dibujado, Invasión de recursos hardware y mal-configuraciones del sistema operativo, entre otros problemas de comunicación externa y librerías controladoras que se lograron resolver en el proyecto de mano a los avances registrados en el administración de versiones software que utiliza el equipo de trabajo en la FCV, el cual fue planteado recientemente para disminuir las demoras y la brecha de fallas humanas.

2.7.4 Avances de desarrollo software en cuanto a interfaz gráfica de usuario (GUI).

Con el fin de mostrar algunos de los avances software en la interfaz respecto a cambios en los menús, funcionalidad de los objetos y demás sucesos gráficos se muestran a continuación las siguientes ilustraciones.



Figura 56. Antiguo menú de historial de alarmas.

Este es un menú informativo de las cosas que han pasado en el monitor durante el tiempo que ha permanecido encendido, menú que lee de la memoria SD una serie de registros para plasmarlos ante un usuario de asistencia médica, sin embargo posee un sistema de búsqueda algo engorroso y de más compleja implementación además de una tabla que no posee mayor información.

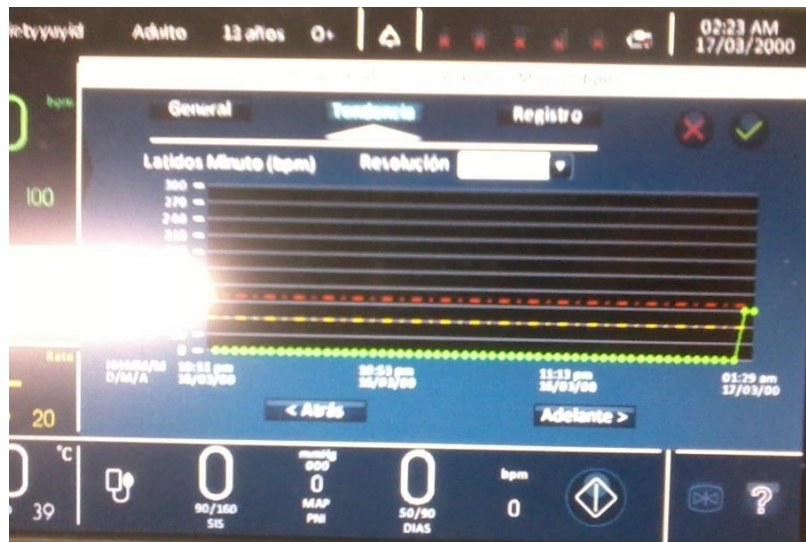


Figura 57. Antiguo menú de tendencias gráficas y representación de las variables estáticas sobre el número cero (0).

La Figura 57 representa la poca información en el eje del tiempo que poseía el menú de tendencias vitales y la errónea representación de los valores vitales estáticos a través de ceros en vez del literal “- -” que es el estándar para representar el formato de valor invalido.

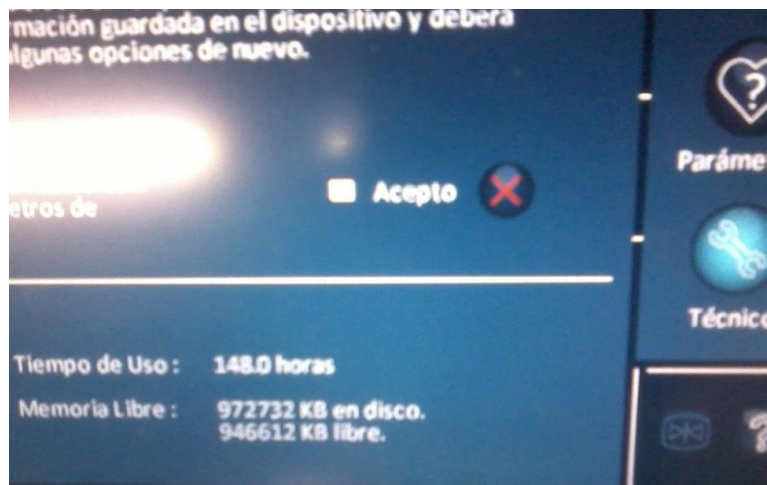


Figura 58. Antiguo menú técnico del monitor.

Menú técnico sin mayor implementación solo informaciones y funcionalidad e restauración de fábrica.

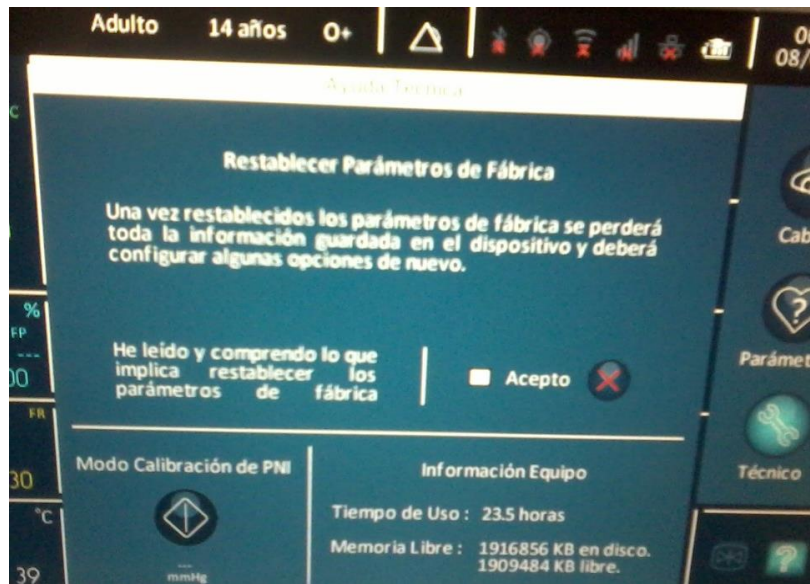


Figura 59. Nuevo menú de técnico con calibración de presión del monitor.

Nueva implementación de calibración técnica de presión para corroborar la correcta medición de este parámetro, en la actualidad también se posee información de las direcciones MAC e IMEI de los dispositivos de red que involucra una respectiva versión del monitor con acceso a Internet.

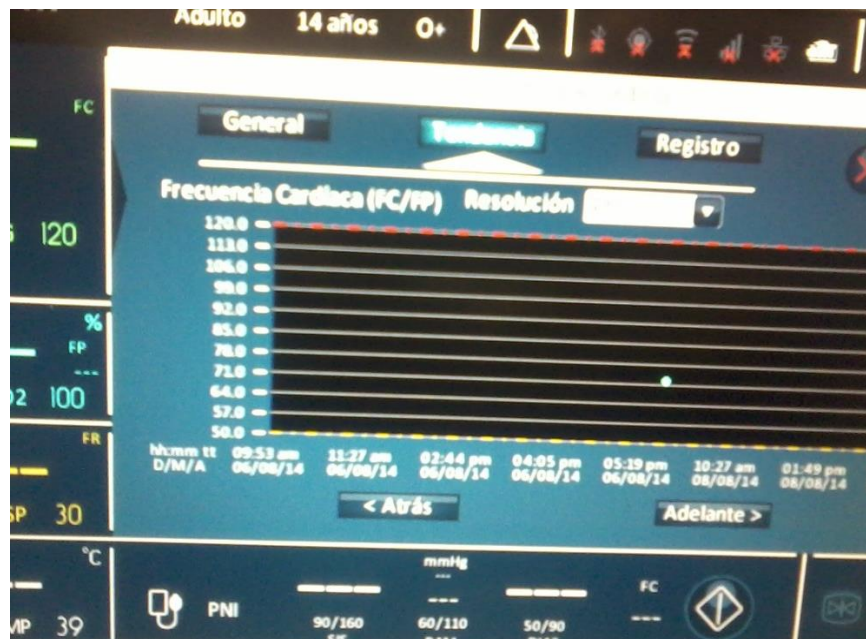


Figura 60. Nuevo menú de tendencias gráficas con mayor rango de tiempo y representación de las variables estáticas nulas a través del texto "- -" como sucede con los monitores comerciales en el mercado.

Mayor información del intervalo de tiempos en tendencia y correcta asignación del literal correspondiente “- - -” a valores inválidos de parámetro.

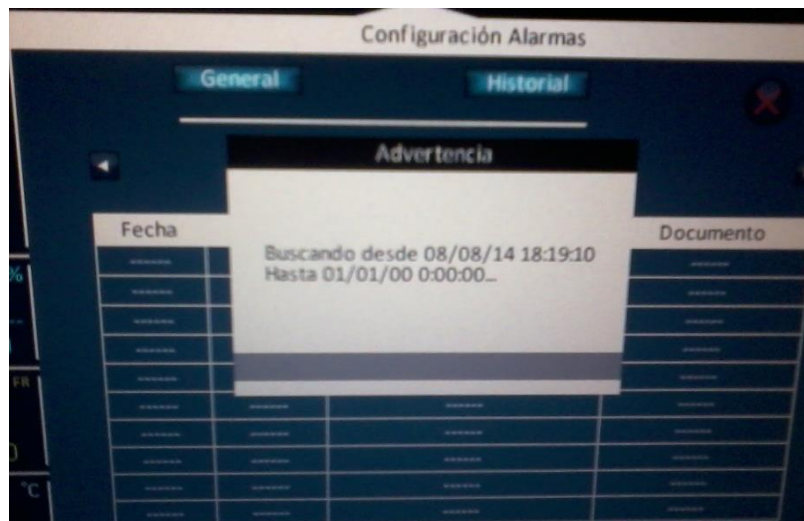


Figura 61. Nuevo menú de historial de alarmas en búsqueda.

Menú de historial de alarmas con información de la búsqueda inmediata que se lanza en la apertura del menú.

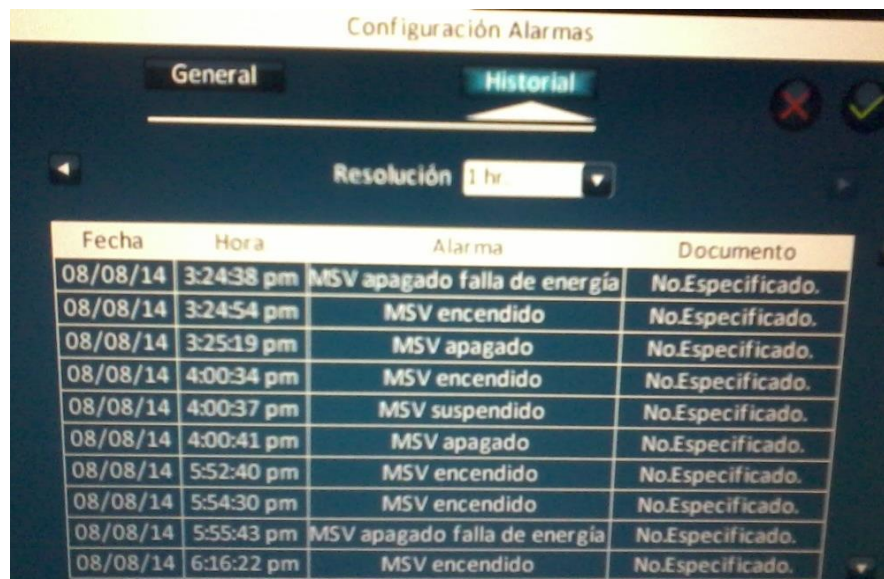


Figura 62. Nuevo menú de historial de alarmas final.

Tabla de información de los eventos de alarma más detallada así como búsqueda a través de resolución de tiempo más sencilla e inmediata con el accionar de las flechas.

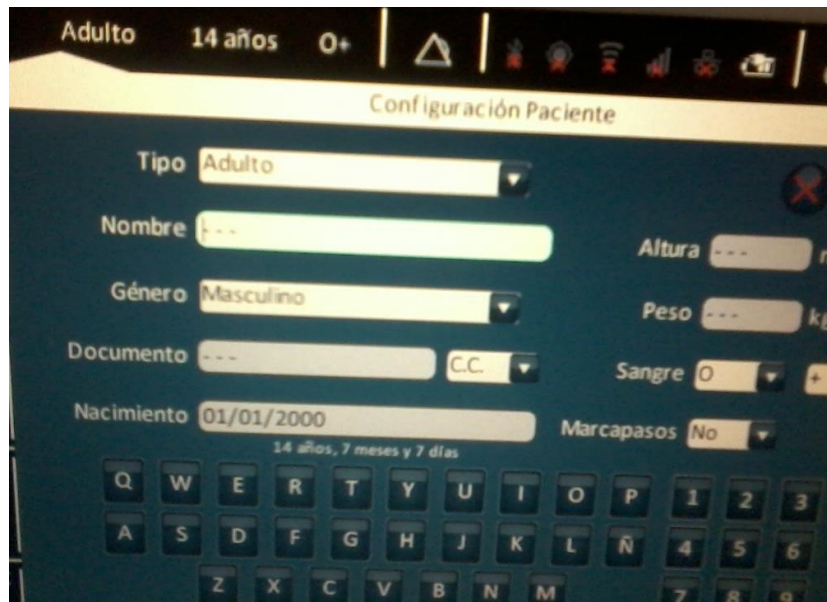


Figura 63. Modificaciones al menú de paciente.

Menú paciente con mayor opcionalidad en cuanto a selección inmediata del modo ECG de marcapasos y documento (No existían antes), así como líneas de invalidez para valores vacíos y restricciones de escritura, ya sea números en el nombre o letras en valores de peso y altura entre otras.

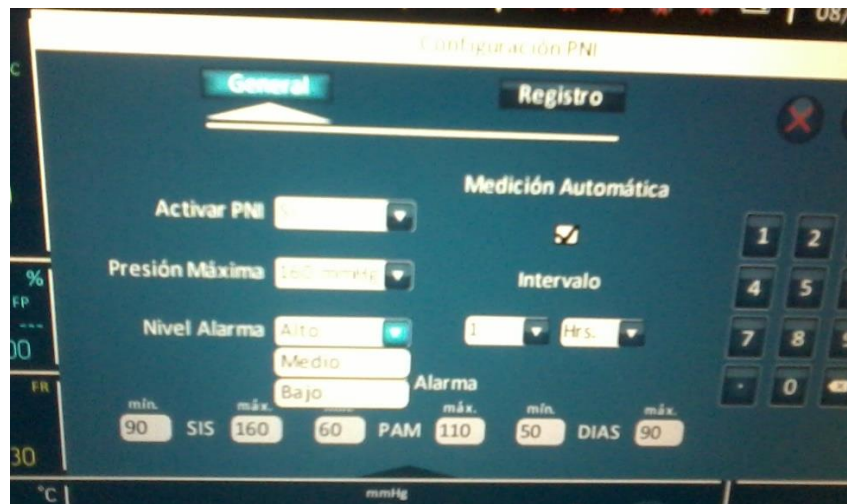


Figura 64. Modificaciones al menú de PANI (Presión Arterial no Invasiva).

Nuevo modo de intervalos de medición automática con accesibilidad a las diferentes opciones de horas y minutos que posee la tarjeta vital en tan solo un par de clics desde interfaz.

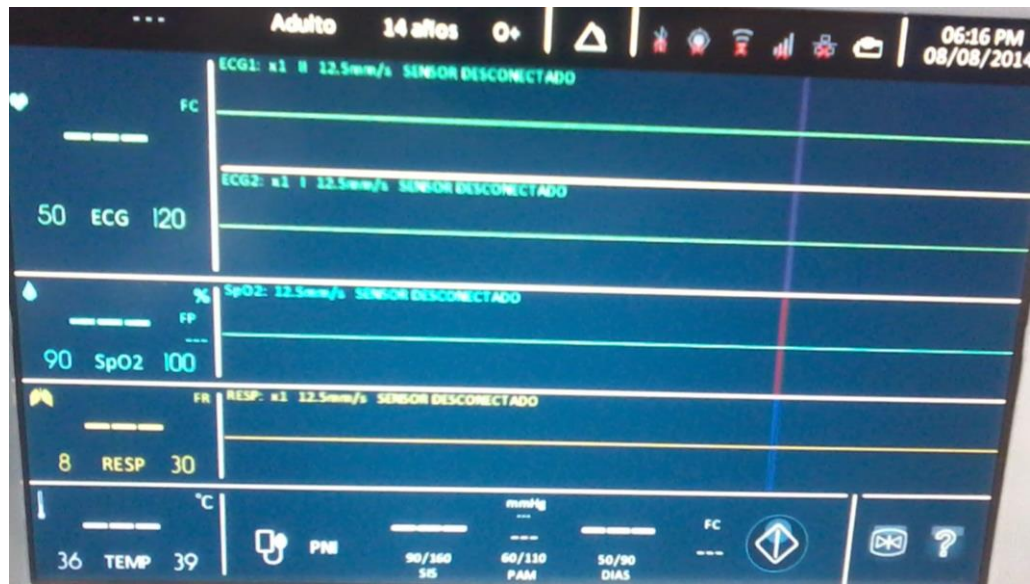


Figura 65. Modificaciones a los mensajes de ondas vitales.

Mayor información sobre los literales de las ondas en conjunto a efectos de parpadeo con mensaje informativo en caso de existir cualquier tipo de alarma del parámetro involucrado.

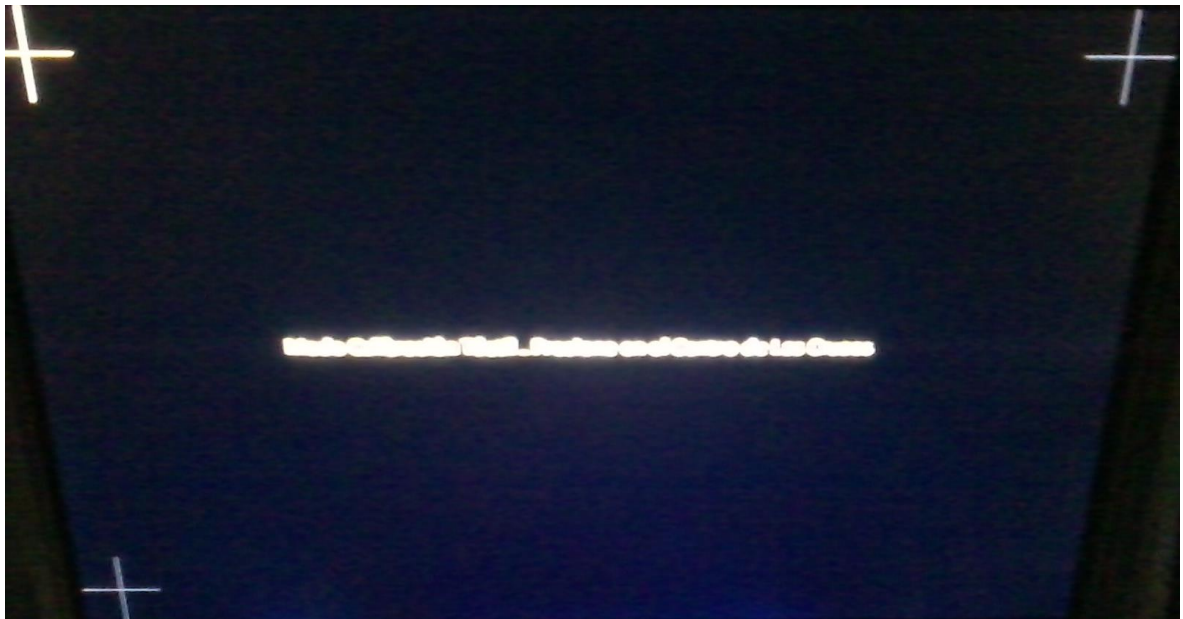


Figura 66. Nuevo modo calibración del "Touch" o táctil resistivo.

El hecho de que el monitor posea una pantalla táctil resistiva y las fallas de ubicación en la fabricación de estas membranas táctiles, orilla a que el monitor posea un modo de calibración para ajustar esta propiedad desde los botones de carcasa del equipo.

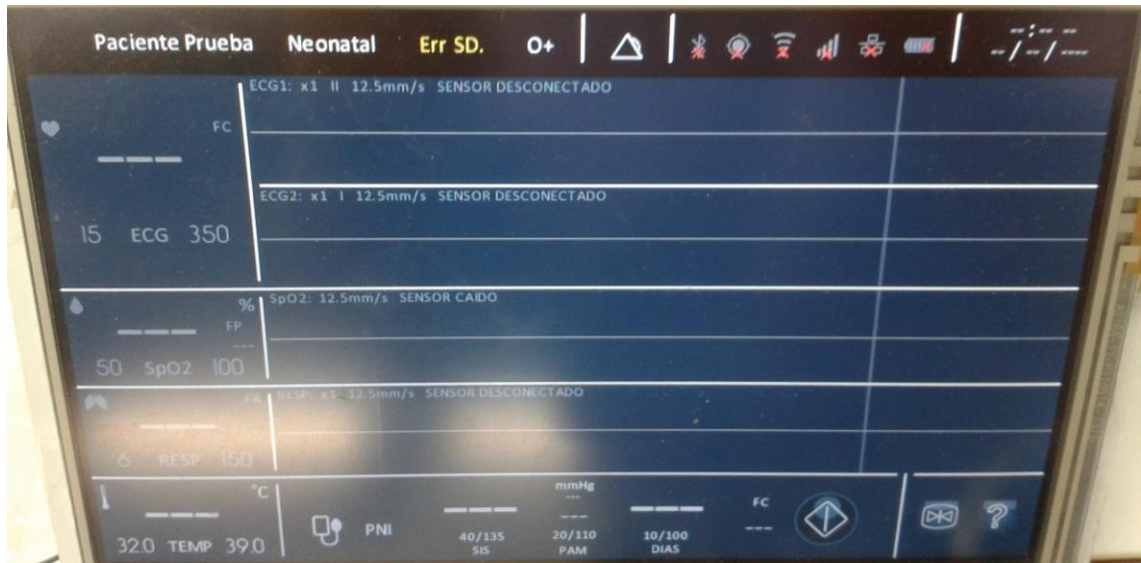


Figura 67. Nuevas técnicas de evasión de errores en memoria SD.

El funcionamiento de la memoria SD se hace crucial para mantener el monitoreo de las funciones del personal médico así como los datos del paciente, es entonces que una vez falla dicho recurso es imperativo que el personal note el cambio y pueda recurrir a cambiar el equipo mientras soporte técnico ajusta este recurso de nuevo.

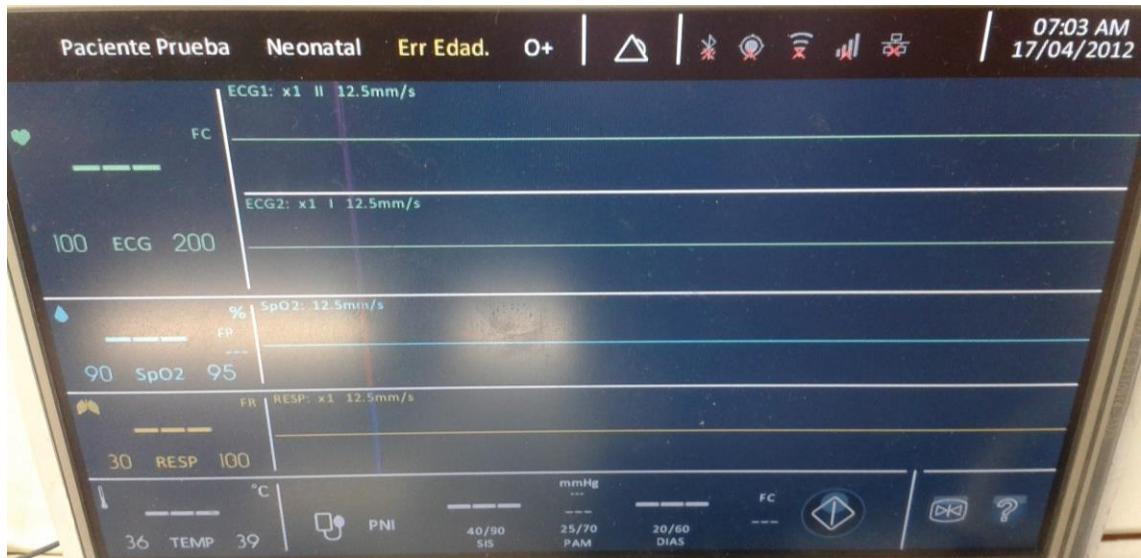


Figura 68. Nuevas técnicas de evasión de errores en cuanto a configuración de edad.

Al igual que el fallo en memoria SD, se hace indispensable que si existe un ajuste erróneo de la edad del paciente, este sea informado de manera explícita y clara desde la pantalla del monitor.



Figura 69. Nuevas técnicas de evasión de errores de lectura en archivos corruptos.

Al igual que la detección de errores de memoria SD y edad del paciente, es importante que el monitor pueda informar de fallas propias y que el usuario por lo menos pueda estar enterado para tratar que actualizar la información requerida.

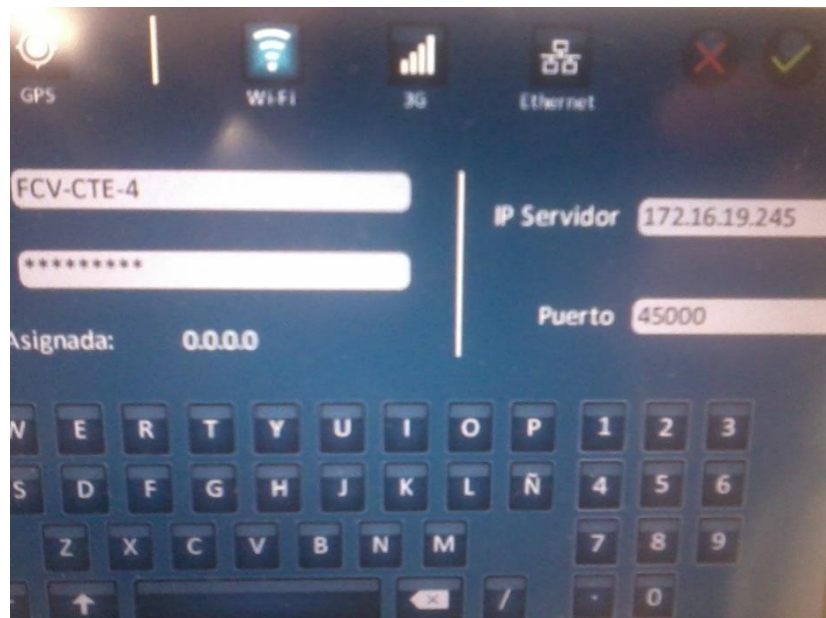


Figura 70. Creación de los menús de redes inalámbricas con funcionalidad real.

Los prototipos con funcionalidad inalámbrica eran manejados a través de consola de comandos, ahora ya han creado los diferentes menús para abordar esta funcionalidad desde una perspectiva de usuario más cómoda.



Figura 71. Corrección de la funcionalidad de marcapasos de la tarjeta de adquisición.

En la funcionalidad actual de marcapasos se había notado un mal cálculo de la medición de ECG debido al pulso de marcapasos que posee el paciente, aumentando al doble la verdadera frecuencia cardíaca del paciente y generando una situación falsa. Para comprobar la funcionalidad del filtro marcapasos se usó el software de apoyo de la compañía “Goldwei” a la que se compró la tarjeta de adquisición vital (Figura 72), resultando en un correcto funcionamiento de marcapasos bajo ciertos niveles de tensión de la onda y no casos extremos como en la Figura 71, para lo cual se debe plasmar dicha información en el manual de uso a través del Diseñador Industrial encargado.

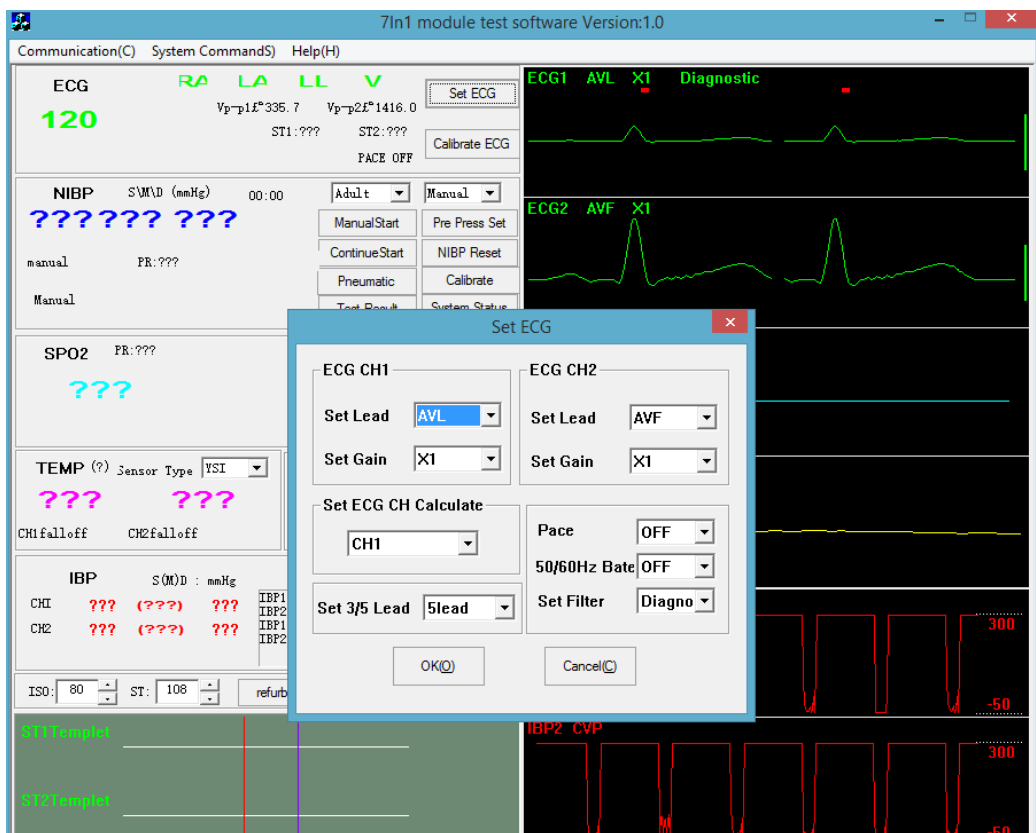


Figura 72. Variadas verificaciones de orden con ayuda del software interactivo propio de la tarjeta de adquisición vital vía terminal USB-UART.



Figura 73. Variadas verificaciones de orden con ayuda del comportamiento típico en monitores comerciales de adquisición de signos vitales.

Finalmente para concretar la funcionalidad de este filtro marcapasos, se compara la acción del mismo en un monitor Mindray comercial y se denota en la Figura 73 como el uso de este filtro si logra reducir la onda no deseada y permite una correcta medición del parámetro de frecuencia cardíaca.

2.8 GESTIÓN DE ADQUISICIÓN VÍA E-MAIL PARA COMPRA DE DIRECCIONES FÍSICAS DE RED O MAC IEEE STD 802

Aparte de los arreglos de interfaz gráfica que se han venido realizando, también se ha podido hacer la gestión de la compra de una asignación o paquete de direcciones MAC, a la organización reguladora de las mismas conocida como IEEE, esto en compañía del director financiero de la FCV y una serie de investigaciones en la compra y llenado de formatos que permiten lograr un contacto directo con ellos y por ende una compra sin mayor inconveniente y de fácil entendimiento para ambas partes; esta asignación de direcciones MAC a la FCV se puede verificar actualmente en la lista publica existente sobre la página web de dicha organización (standards.ieee.org/develop/regauth/oui36/oui36.txt).

La razón de dicha gestión y compra final de direcciones MAC por parte de la FCV, se da bajo la necesidad que tiene el controlador electrónico de Ethernet llamado "Wiznet" de poseer una dirección MAC adecuada, ya que este sistema posee una memoria volátil y necesita de una aplicación que le posibilite este número MAC único al momento de la configuración de este chip; por otra parte el módulo de Wi-Fi (MAC), así como el módulo de 3G (IMEI) poseen su propia dirección física valida asignada internamente de manera

no volátil y editable al mismo tiempo, por ende no requieren de mayores inconvenientes de asignación, sin embargo habría que pensar que la dirección MAC que poseen los dispositivos de Wi-Fi corresponden a la asignación que le pertenece a su fabricante (Roving Networks) en la lista pública de la IEEE y por ende no es propiedad de la FCV y en cualquier momento su fabricante podría reclamar derechos sobre el uso sobre dichas MAC de fábrica que poseen los dispositivos, así que aún se está analizando la posibilidad de asignar uno de los números MAC de la FCV para este dispositivo de protocolo Wi-Fi también.

2.9 “BIOS” LA IMPORTANCIA DEL CONTROL A BAJO NIVEL: AUTOMATIZACIÓN FINAL DEL DISPOSITIVO

Un último tema respecto al desarrollo del equipo médico, está referido a la programación y puesta en marcha del sistema conocido como BIOS, el cual debe realizar una correcta labor de bajo nivel que proporcione seguridad en cuanto a manejo de potencia, alimentación y carga de la batería del equipo, encendido, apagado, suspensión, LEDs de guía externos, reseteo físico de bloqueo, entre otras funciones que concretan la autonomía de manejo del sistema de monitorización vital con respecto a un usuario final.

2.9.1 Concepto de BIOS.

Antes de empezar con la descripción de las reglas de operación de este dispositivo, cabe aclarar un par de detalles en cuanto al concepto de BIOS; primero que nada muchas personas hablan y relacionan este dispositivo como un chip base de bajo nivel que relaciona los primeros pasos de un dispositivo en la ejecución de encendido, apagado y otras verificaciones muy sutiles pero concernientes para el correcto funcionamiento del mismo, sin embargo pocos saben que la BIOS es realmente un concepto que simplemente se comenzó a utilizar para abarcar cuestiones de bajo nivel en dispositivos comunes sin ningún compromiso de estándar y aun así empezó a ser adoptado por los desarrolladores de dispositivos sin ninguna norma o especificación clara de automatización.

BIOS abarca un pequeño software de aplicación conocido como “Firmware” corriendo en un pequeño dispositivo a gusto del desarrollador, pero no quiere decir que sea menos importante, ya que muchas veces es pasado por alto para concentrar la atención a los dispositivos más llamativos y funcionales, cuando es este chip quien cumple las funciones más básicas que permiten la permanencia de uso de cualquier dispositivo.

El concepto de BIOS guía temas tan importantes para la energización, sincronía de encendido, apagado, regulación e inicialización que nadie nota, pero sucede que ningún computador en el mundo podría sortear temas de inicio tan complejos para cada dispositivo físico en específico, los cuales requieren la lógica humana y la aplicación Firmware sobre tal concepto de BIOS; bajo esta premisa, ninguno de estos computadores del ejemplo anterior podría funcionar correctamente o como mínimo sufrirían de constantes problemas y daños aun estando “recién salidos de la caja”.

Para comprender este concepto más a fondo se plantea la Figura 74 como dinámica BIOS actuando sobre el dispositivo de monitorización vital que se viene desarrollando.

2.9.2 Dinámica BIOS sobre la lógica del equipo médico.

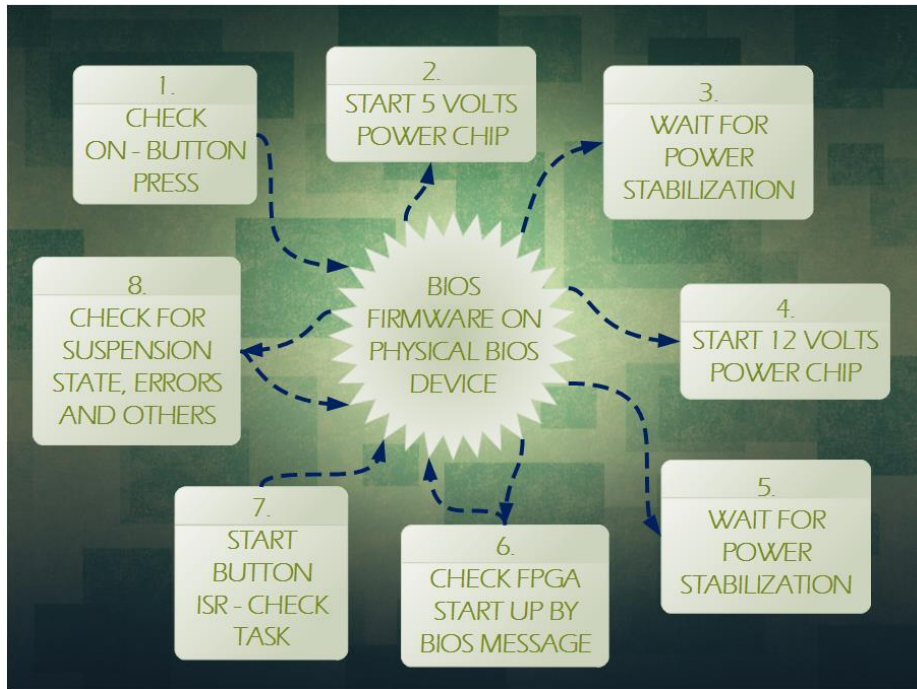


Figura 74. Dinámica de funcionamiento BIOS.

Ahora bien para reafirmar el ejemplo anterior con el ejemplo real de aplicación BIOS actual, se aclara que la BIOS del proyecto del equipo médico se viene trabajando como un concepto de microcontrolador Atmel, ligado a labores de regulación y establecimiento del hardware del equipo médico mediante instrucciones software base (Firmware) siendo ejecutadas por BIOS.

2.9.3 Estructura física y de conexión BIOS sobre el hardware que abarca el equipo médico.

A continuación se muestra parte del desarrollo asistido por computadora “CAD” de los ingenieros en la FCV sobre el software Altium Designer en la creación de los PCB que integra el equipo médico, este no es un desarrollo propio de la práctica ni mucho menos, pero centra el concepto de BIOS sobre una base realista de aplicación para la comprensión electrónica del mismo.

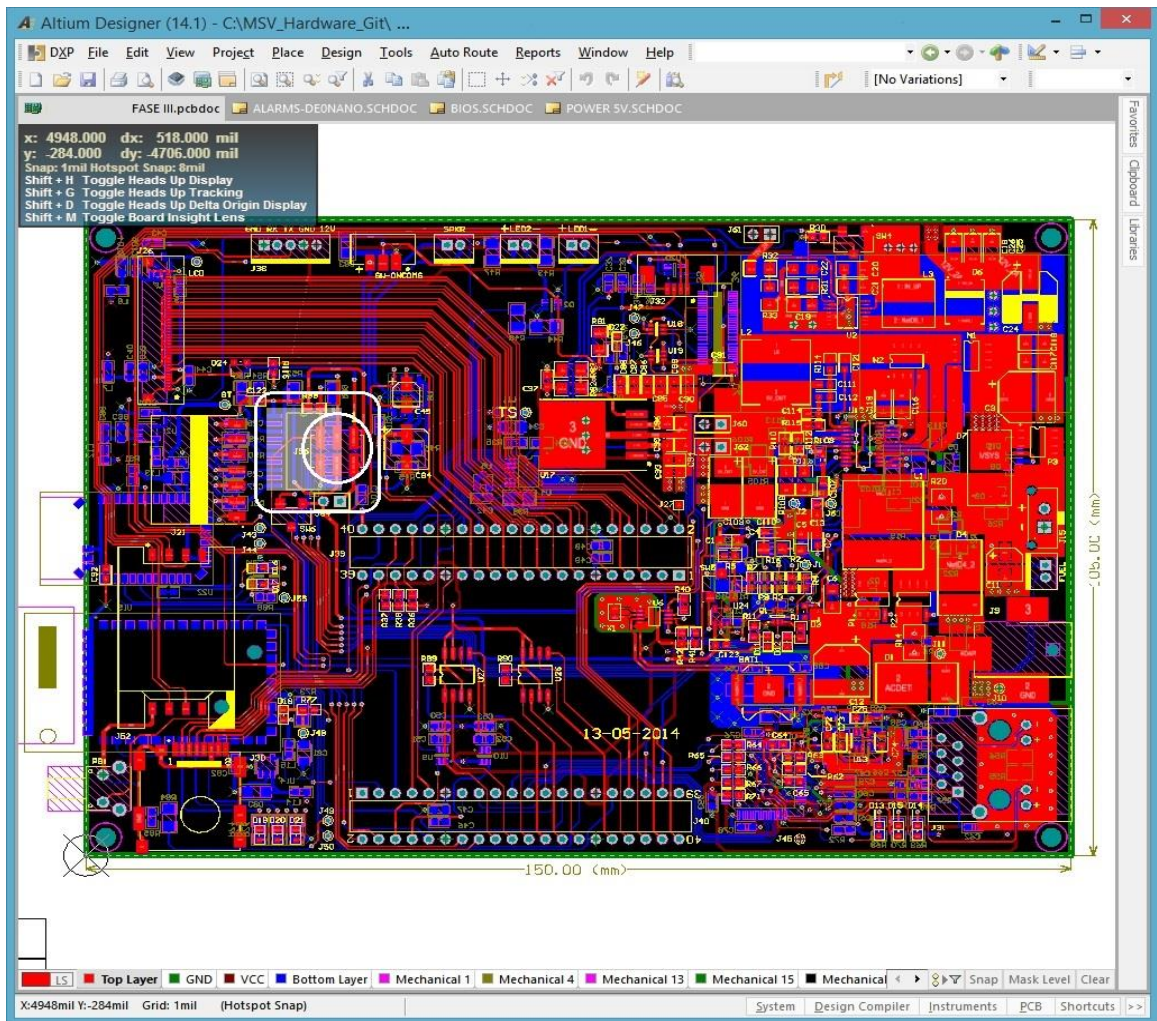


Figura 75. Espacio de la BIOS sobre el diseño PCB de la tarjeta electrónica principal del sistema de monitorización vital.

En la Figura 75 se puede observar al integrado Atmel sombreado en gris, rodeado por un cuadro de color blanco y de caminos de PCB en color azul; esto último indica que se encuentra en la capa inferior de la tarjeta electrónica en cuestión, por otro lado se tienen exactamente seis (6) pines de color rojo rodeados por un círculo de color blanco, estos pines están conectados entre capas a pines del integrado Atmel y se usan para la programación de las instrucciones Firmware del mismo, abarcando el concepto de BIOS de este equipo médico como un microcontrolador más conectado a una serie de salidas y entradas lógicas que controlan funciones de bajo nivel. Por último un pulsador de reseteo manual ejemplificado en la Figura 76 sobre un círculo de color rojo, permite desbloquear y reiniciar el equipo desde una perspectiva mucho más física en caso de bloqueo extremo y de ser necesario su uso...

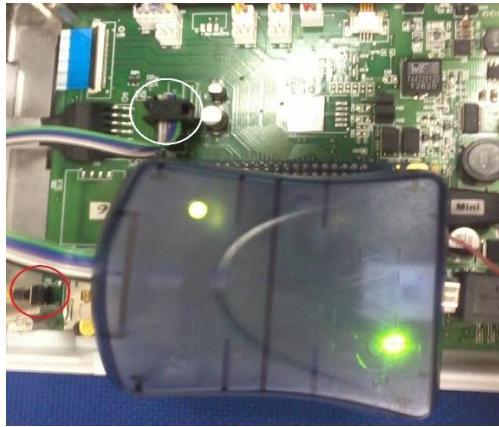


Figura 76. BIOS en proceso de programación - Conexión adecuada del programador Atmel.

En la Figura 76 también se ejemplifica la correcta conexión del programador Atmel sobre los pines de programación destinados para este fin; ha de aclararse que dicho programador se encuentra conectado al computador de escritorio con el software de desarrollo y controladores USB para PC de la compañía Atmel instalados correctamente para el respectivo programador.

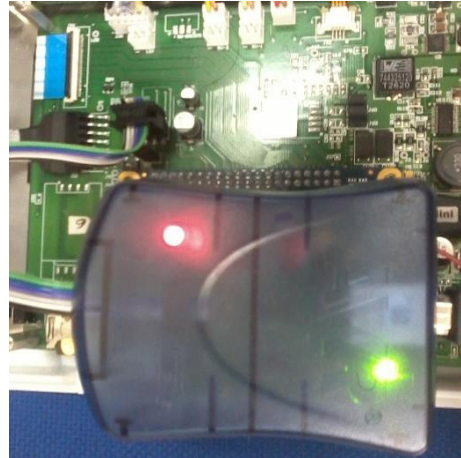


Figura 77. BIOS en proceso de programación - Conexión inadecuada del programador Atmel.

En un segundo ejemplo de conexión de la Figura 77, se ofrece una conexión correcta del dispositivo programador pero con la carencia de energía (sin adaptador o batería conectados al sistema de regulación de la tarjeta electrónica) respecto al chip Atmel, el cual se encuentra indispuerto para la programación por esta misma falta de poder.

Los ejemplos de las ilustraciones previas logran un entendimiento más amplio en cuanto a teoría y práctica real de los conceptos de BIOS, la falta de la energía en la regulación de la tarjeta electrónica de la Figura 77, reafirman el uso básico y ejecución permanente de la BIOS aun cuando el sistema se encuentra en estado de apagado (Solo batería).

El Firmware puede asemejarse a una serie de instrucciones BIOS atentas a cualquier cambio del exterior del equipo para iniciar procesos de inicio del sistema, carga de batería, fallas de energía de potencia como cortocircuitos, caídas de tensión o tensiones críticas de batería importantes, que merezcan un apagado prematuro del equipo en caso de estar encendido, secuencias de LEDs que informen rápidamente de la situación del equipo mismo sin requerimiento de terminales desconocidos o software especializados entre otras funciones interesantes que puede proveer dicha BIOS.

Como último destacar que el concepto de BIOS es mucho más amplio al de un par de instrucciones Firmware de bajo nivel del equipo médico en este caso, sino como todo un concepto mucho más flexible que incluye un cerebro atento sobre un dispositivo físico tras instrucciones Firmware, las cuales pueden realizar unas tareas específicas de entrada y salida que encierran al sistema, de eso se trata BIOS, un agente muy importante pasando desapercibido sobre las funciones de más alto nivel que le proceden.

2.9.4 Lógica software de BIOS como Firmware base del equipo médico.

Ya se ha comprendido un poco mejor el concepto de BIOS y su percepción como Firmware en un sistema embebido, sin embargo ahora es el momento de desarrollar sobre su lógica de instrucciones dispuesta en el entorno de desarrollo Atmel, parecida a la lógica de código C, pero con ciertos rasgos en lenguaje de más bajo nivel como lo es el código de descripción "assembler", dado al seteo y reseteo de banderas que habilitan registros de temporización en el integrado, proceso muy común en el desarrollo de instrucciones que componen al Firmware BIOS sobre el entorno de desarrollo de Atmel (Atmel Studio 6.0), del cual se puede hacer una idea con la Figura 78.

2.9.4.1 Entorno Atmel Studio como herramienta de desarrollo software.

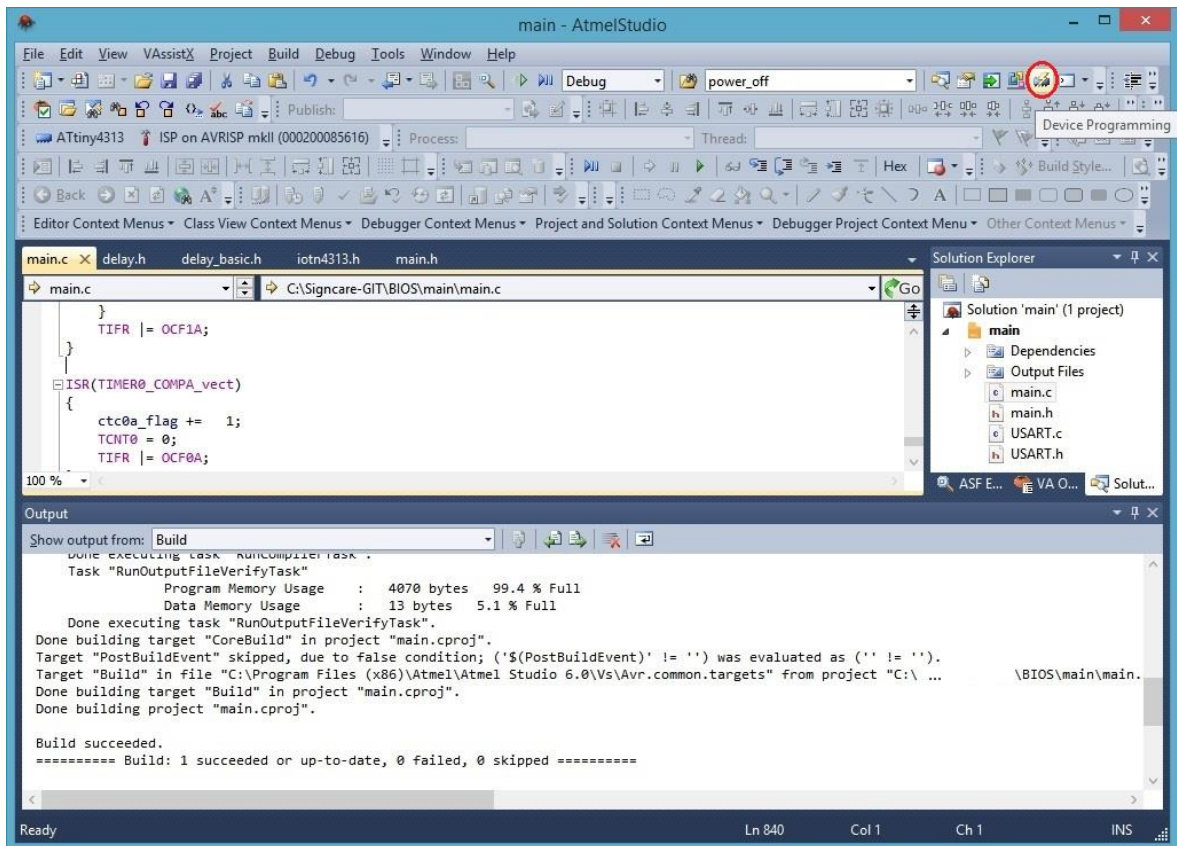


Figura 78. Entorno Atmel Studio versión 6.0 - Edición del proyecto de nombre “main”.

La programación dedicada al desarrollo de las instrucciones BIOS debe ser esencia sencilla, debe tener la menor cantidad de código posible para que sea fácilmente entendida y ajustada por los desarrolladores, o por lo menos debe ser muy organizada para evitar fallas de ejecución en tan básico nivel, imprimiendo un toque de robustez y confianza al desarrollo del Firmware BIOS.

2.9.4.2 Estructura de proyectos software en el entorno Atmel Studio.

Para lograr este tipo de programación, primero que todo se debe crear un proyecto en el entorno “Atmel Studio” haciendo referencia al chip en el que se desea empezar a desarrollar, el entorno creara una carpeta con el nombre del proyecto la cual contendrá los archivos concernientes al “solution”, el cual se puede asemejar al “workspace” que se utiliza en muchos entornos de desarrollo software, estos archivos de “solution” vienen acompañados una nueva carpeta con el nombre del proyecto nuevamente, carpeta que contiene los archivos fuente de compilación del proyecto y otros archivos propios del proyecto como tal, los cuales hacen referencia al hardware del chip que fue seleccionado en la creación de tal proyecto y enlazan las diferentes librerías Atmel necesarias para la

compilación; algo parecido a como sucede con la carpeta “Board Support Package (bsp) en el entorno Eclipse.

Lo referente a los archivos fuente de tal proyecto se reflejan en la creación automática del archivo “nombre_del_proyecto.c” el cual contiene la función principal de todo proyecto software “int main (void)” para empezar a agregar la lógica de programación en C e inclusive C++ como se ha probado para el compilador del entorno “Atmel Studio”, una vez que el software esté listo para funcionar, solo basta aplicar la respectiva programación al integrado como lo indican las respectivas hojas de datos, las cuales deben ser estudiadas con anterioridad para entender cómo manejar el chip que compondrá la lógica BIOS. Cabe aclarar que cada nuevo integrado es un nuevo estudio y el estudio de la hoja de datos referente al chip utilizado (ATtiny4313) fue una labor bastante tediosa y dedicada, para que en el momento de modificar la programación en el chip se tuvieran las bases necesarias para abarcar la programación evitando daños en la lógica software de ese momento.

Otra nueva aclaración respecto a este tipo de integrados, es que aunque son bastante robustos por su resistencia al ruido y otros factores de potencia que pueden manejar, también poseen una cantidad muy limitada de memoria que hay que saber sortear antes de programar, ya que como se alcanza a apreciar en la Figura 78, el chip está a su 99.4% de capacidad y tan solo utiliza un par de librerías “main” y “USART” desarrolladas previamente por los ingenieros en la FCV, las cuales fueron editadas para mejorar el funcionamiento en secuencia de encendido apagado y respuesta al teclado de membrana externo, proceso que evidenció que dichas librerías no contienen mayor extensión de código y aun así hay que aprovechar al máximo la capacidad disponible con técnicas de ahorro software, las cuales lograron identificar mas no reducir a menor capacidad dado a los nuevos requerimientos que se definieron en medio del proyecto.

2.9.4.3 Programación de la compilación software sobre el entorno Atmel Studio.

El software fue estudiado y modificado con los nuevos desarrollos, restando programar el chip BIOS, solo basta tener correctamente instalados los controladores que proporciona el entorno “Atmel Studio” en el computador de escritorio usado en la compilación, tener correctamente alimentado el chip y conectado el programador Atmel de ilustraciones anteriores. Un par de pasos en la utilización de esta herramienta software serán entonces necesarios para programar el dispositivo.

Los pasos para lograr una correcta programación de un chip fabricado por la compañía Atmel no son muy complejos, ya que después de hacer las correcciones respectivas y abrir la herramienta programadora de este entorno, basta seleccionar el dispositivo programador y el dispositivo que va a ser programado para aplicar la configuración a la herramienta, realizar un par de verificaciones extra y ejecutar la programación final del integrado desde el entorno vía USB con el programador, tal y como se aprecia en las siguientes ilustraciones.

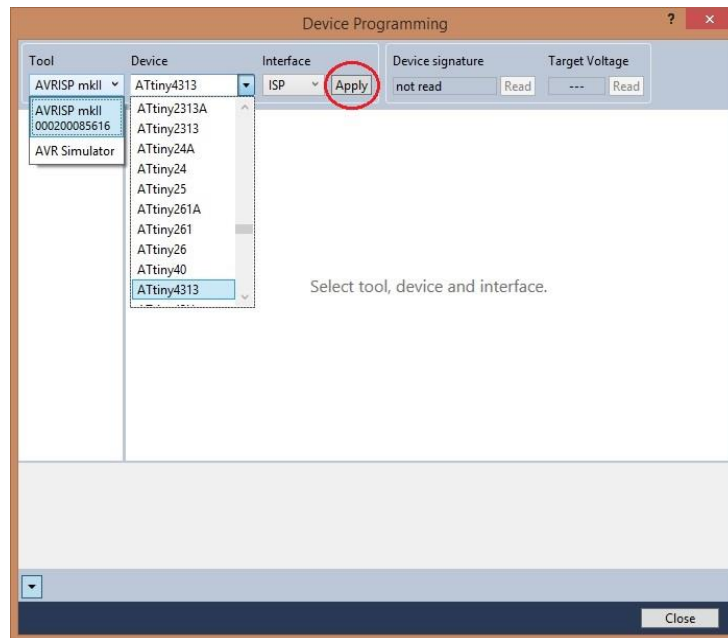


Figura 79. Aplicando la configuración de programación en el entorno Atmel para con el chip de la BIOS utilizado.

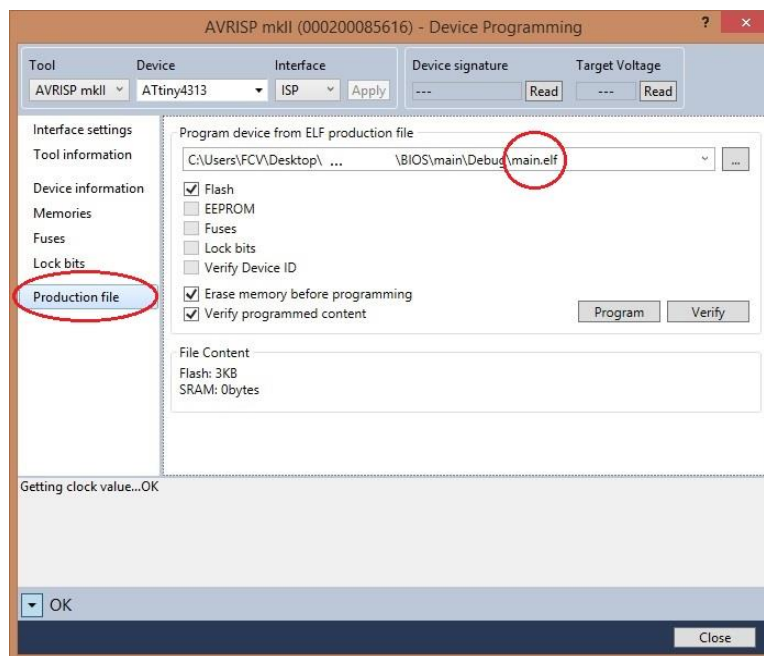


Figura 80. Verificando el archivo de programación software final "main.elf".

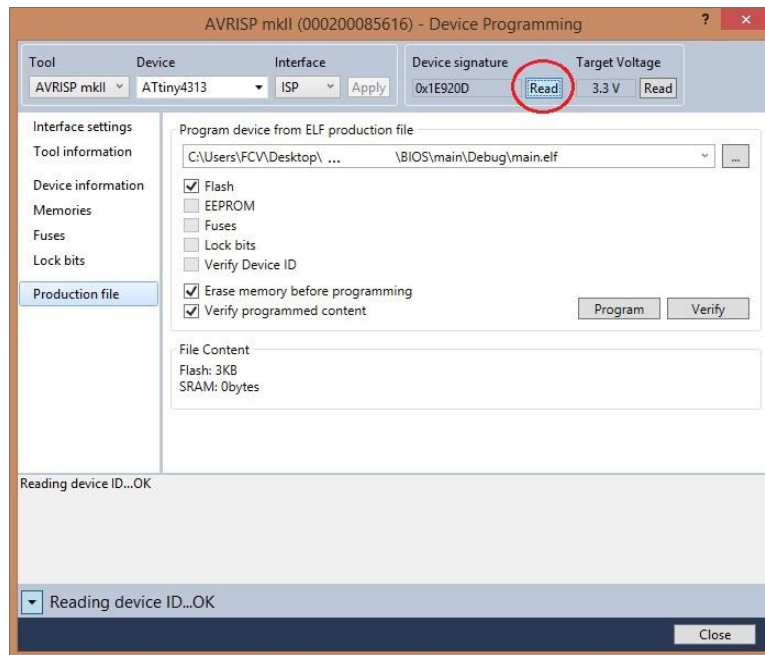


Figura 81. Verificando las conexiones hardware y disposición del chip físico desde el entorno Atmel.

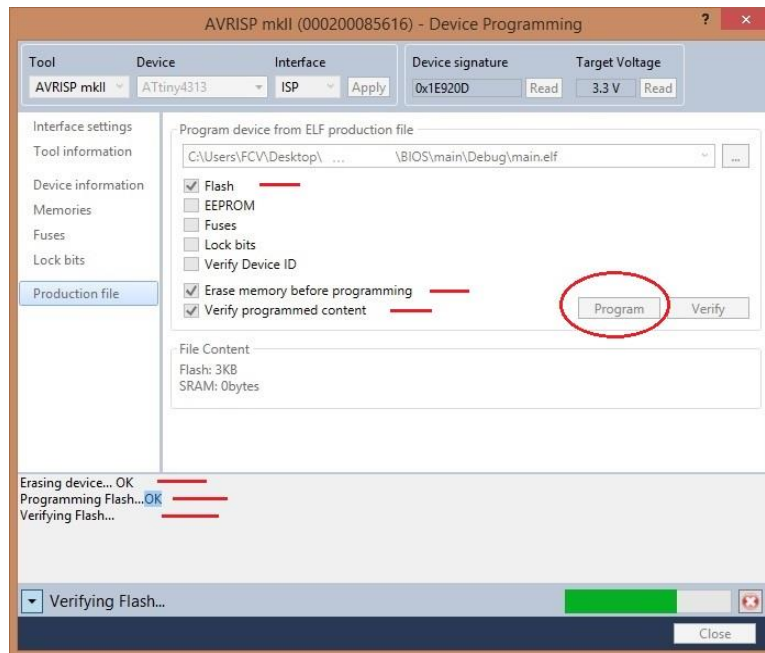


Figura 82. Ejecutando y verificando la respectiva programación vía programador Atmel USB.

El desarrollo sobre entornos Atmel (chip de BIOS) y otros conceptos bastante interesantes llegan a su fin y aunque en este pequeño espacio se logra resumir bastante bien los esfuerzos realizados, no se alcanza a apreciar el arduo trabajo en cuanto a estudio y horas de trabajo para entender todos y cada uno de los conceptos y procedimiento aquí explicados.

2.10 DEPURACIÓN Y TRABAJO CONTINUO EN LA CORRECCIÓN DE ERRORES

Esta última etapa es básicamente un recuento de los procesos ya explicados hasta el momento, en donde se unen dos recursos importantes de la UEE (Unidad Estratégica Empresarial) - Bioingeniería como lo son las áreas de Diseño y Desarrollo (D&D) con el área de Validación y Metrología (V&M) para así agilizar el proceso de depuración y corrección final de errores del monitor de signos vitales (MSV), proceso que abarca largas semanas de pruebas, nuevas implementaciones, más pruebas, atención de requerimientos que surgen de forma no premeditada y de carácter obligatorio, compilaciones, programaciones, informes de resultados y aporte al conocimiento del sistema de gestión de la calidad de la organización (FCV), solicitudes de verificación y calibración entre otros, dando como resultado un dispositivo más robusto, probado y verificado hasta fin de termino.

Aunque parezca el fin de esta práctica, mi trabajo ha sido fruto de mejoras importantes en cuanto a desarrollo en el campo de las FPGA, microcontroladores y otros dispositivos de lógica digital, en conjunto a nuevos aportes y aplicativos de diseño en cuanto a PCBs y electrónica de potencia que no abarcan esta práctica, los cuales continúan avanzando con la versión "Wireless" del monitor en cuestión, la cual agrega ciertos niveles de autonomía, portabilidad y accesos remotos que aplican en el campo de la telemedicina, esto además de un nuevo monitor de piso para cirugía, proyecto futuro que pretende reemplazar las aplicaciones de "LabVIEW" sobre computadores convencionales, las cuales abarcan los mayores desarrollos actuales de la FCV en este campo.

2.11 CARGA DE BATERIAS EN BASE AL DISPOSITIVO FUEL GAUGE E INTERACCIÓN CON LA INTERFAZ GRÁFICA.

Como un agregado que incluyo el trabajo con la interfaz gráfica para la interacción del usuario respecto al estado de carga de la batería, fue el entendimiento de uso del dispositivo de censado "Fuel Gauge DS2782", el cual posee una serie de parámetros no volátiles que permiten gestionar un perfil de batería que aplica como administrador de la carga de la misma, así como comunicación de tipo I2C con el software del equipo médico.

Una vez se tuvo un entendimiento más amplio del mismo, hubo una serie de pruebas en cuanto a condiciones limite que no se tenían muy en cuenta en el software del sistema equipo médico y que se evidenciaron como falla una vez tuvo cabida la etapa de

verificación de las funcionalidades del mismo, así que se realizaron los respectivos cambios que permitían evitar estados erróneos de censado y por ende daños internos y sobrecargas no provocadas, por otro lado el dispositivo de censado es muy completo en cuanto a la detección de múltiples condiciones y estados sin mayor intervención, sin embargo no tenía manera de establecer cuando una batería se encontraba algo descargada mientras su contador interno de culombios fuera erróneamente alto (Caso de cambio de batería sobre el mismo chip de censado); es entonces que se implementa una condición que permite este reconocimiento en base a tensiones y corrientes medidas por el mismo chip y así se realiza el cambio del contador de culombios junto a una secuencia de reprogramación del perfil de la batería llegado el caso.

Estas nuevas mejoras de software incluyen una secuencia más intuitiva y automatización del proceso intrínseco de la batería, no solo evita riesgos reales de potencia y carga, sino que elimina por completo la programación y revisión de la carga inicial de una batería nueva en base a una serie de archivos y complejos procesos de reconocimiento, solo bastará conectar la batería y esperar un 100% de carga; en el proceso, este porcentaje podría cambiar abruptamente hacia abajo en una batería nueva, pero una vez llegado a su tope en 100%, ya estará lista para funcionar sobre el mismo monitor al que fue ensamblada, por otro lado en funcionamiento normal el usuario puede detectar como su equipo empieza a tener caídas de porcentaje amplias anormales que le darían pista de algún defecto en la batería del equipo si este proceso es repetitivo, visualización que se logra a través de la información del menú varios del monitor (Figura 83).

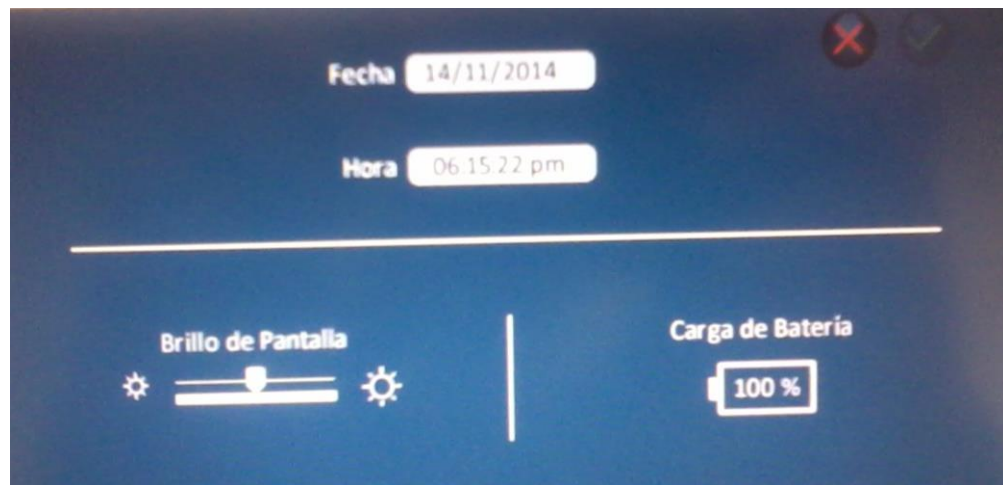


Figura 83. Información de porcentaje del monitor de signos vitales sobre la interfaz gráfica en el menú varios.

Además de los desarrollos funcionales hubo lugar a investigaciones respecto al manejo de las baterías, almacenamiento, precauciones entre otros temas importantes que se agregarán al manual de usuario y servicio del monitor; información resultante de un artículo de investigación de la Universidad de Texas A&M que se adjuntará a los diversos manuales en conjunto a la acción del diseñador gráfico a cargo del proyecto.

3. APORTE AL CONOCIMIENTO

El amplio estudio e investigación de técnicas de desarrollo hardware y software, junto al continuo desarrollo de informes de avance acerca de las funcionalidades del sistema y demás inconvenientes, es un aporte que crece día a día para los ingenieros que reemplacen los desarrollos en la organización cuando llegue a ser necesario.

La implementación de un nuevo administrador de versiones con la opcionalidad, versatilidad y facilidad ofrecida por la plataforma "SourceTree" a través del protocolo Git, es una inmensa herramienta que imprime ese toque de organización, ahorro de tiempo y simplicidad, que quedará disponible para los ingenieros que desarrollen sobre todo tipo de código y administración de archivos.

La planeación es el proceso más importante que todo proyecto debe tener, no debe ser pasado por alto y debe tener un nivel de complejidad mayor o igual al desarrollo práctico del proyecto mismo, ya que se incurre en pérdidas considerables de tiempo.

La investigación y el estudio previo al desarrollo, son primordiales para el desarrollo mismo, ya que si no se tienen las bases del conocimiento adecuadas para abordar un tema, no se podrá ejecutar eficazmente para quienes solo pedirán resultados. Comprometer labores de las que no se tienen claridad, solo traerá presión al desarrollador; lo que se traducirá en pérdida de tiempo y bajo desempeño que afectará a todo el equipo de trabajo.

Otro nuevo aporte al conocimiento referido a la implementación de nuevos entornos de desarrollo hardware como lo fue el proceso de desarrollo a través de la plataforma Qsys, es casi invaluable al saber que pocos se atreven a ingresar en un nuevo entorno de trabajo, dado a miedos, fallas y pérdidas de tiempo que todo proyecto debe soportar.

Gestiones respecto a monitores de signos vitales, comunicación con las diferentes organizaciones y compañías en el asesoramiento de trabajo sobre dispositivos comerciales, contacto con el campo médico y experimentación con el entorno mismo, así como las demás interacciones con el sistema administrativo y de la gestión de la calidad de la organización, es de ese tipo de cosas que para un profesional son fundamentales y no se aprenden en un salón de clase, siendo un gran aporte a nivel personal.

Deja un gran aporte para la comunidad en general abarcar temas tan poco conocidos en cuanto a creación de componentes Avalon y uso de entornos de desarrollo hardware para FPGAs de la compañía Altera, como lo es el entorno Qsys descritos de forma esencial y práctica a lo largo de esta práctica.

Abarcar proyectos tan complejos y de alta responsabilidad como lo es el diseño y desarrollo "CAD" gráfico-funcional del monitor en cuestión a través del apoyo de un único ingeniero desarrollador restante en la FCV, obliga por completo al estudiante a absorber y aprender la mayor cantidad de técnicas de desarrollo práctico en cuanto a electrónica se

refiere; aportes de todo tipo que lo impulsan a trabajar a través del conocimiento y la educación, lo cual sería muy provechoso para los interesados mientras exista el tiempo libre disponible necesario.

El concepto de BIOS y funcionalidad de la instrucción de bajo nivel y acción del Firmware, son conceptos clave en la automatización de cualquier dispositivo y en este documento se trata de explicar muy a fondo la esencia real de estos conceptos evitando confusiones e interpretaciones erróneas de todo tipo que se encuentran sobre discusiones en la Internet.

Depuración de todo tipo (Hardware y Software), han aportado inmensamente en las bases del conocimiento de múltiples lenguajes de programación a niveles bastante altos que ciertamente no se profundizan mucho en la academia, niveles de abstracción que llegan a relacionar la gran variedad de lenguajes como uno solo y su finalidad explícita para con los desarrollos; semejanzas, ventajas y limitaciones están muy bien identificadas en esta etapa final de la práctica y abren un nuevo mundo aplicativo que complementa perfectamente la funcionalidad electrónica que se ha venido explotando a lo largo de estos semestres de estudio en la UPB.

El concepto de carga de baterías a través de la administración de un chip electrónico de censado, es un concepto que debe tener el estudiante que se enfrente a este tipo de casos en donde un perfil que incluye temas como afectación por temperatura y contador de culombios para evadir sobrecargas, son clave en la implementación de sistemas de control de empresas que operan con energía.

4. CONCLUSIONES

El desarrollo y la unión de todos los periféricos existentes en el proyecto sobre la plataforma Qsys, renuevan la labor de descripción del sistema modular hardware-software del dispositivo de monitorización vital, proceso que requirió avances de investigación acerca de protocolos de comunicación industrial involucrados, programación, descripción de hardware y comprensión del sistema en general.

Las múltiples correcciones al sistema hardware y más específicamente de video, lograron resolver los problemas de sincronización respecto a la graficación de las señales vitales con el reinicio momentáneo del módulo de lectura de los pixeles, así como algunos inconvenientes con puentes sincronizadores, los cuales aportaron a generar ruido en modularidad adyacente dada la gran cantidad de lógica disponible usada.

Con la creación del módulo de comunicación entre procesadores “Mailbox”, se logra una comunicación más estable, sin ningún tipo de fallas, completamente personalizada y rápida, además de una independencia total de los módulos de propiedad intelectual de la compañía Altera en este campo, los cuales requirieron modificación inadecuada de las librerías originales para el correcto funcionamiento dado a los fallos presentados.

La implementación del softcore Nios II como herramienta principal del procesamiento software y los cambios de mejora y optimización de procesos de sincronización y gasto de lógica embebida, implicaron la modificación de las diferentes librerías software para el manejo de dichos periféricos, dando como resultado una reestructuración de las librerías básicas de interacción con el medio externo en conjunto con la acción de los comandos de comunicación avalon y el sistema operativo que proporciona el entorno software.

Una vez las bases físicas y virtuales del sistema de monitorización vital fueron optimizadas con los múltiples arreglos, se realiza la modificación netamente software del flujo e interacción con el usuario entre la interfaz gráfica, tarjeta de adquisición vital, registro de datos en memoria SD entre otras funcionalidades gráficas disponibles en la base del sistema en general previamente descrito, dando como resultado una plataforma completamente funcional y lista para ser usada en pacientes y demás personal médico calificado, restando las pruebas finales de uso con el área de validación y metrología junto a pequeñas correcciones que deban ser implementadas que surgan de tal labor.

El sometimiento a pruebas y rediseños de todo tipo han sido clave para encontrar nuevas fallas que adentraron la investigación al manejo e investigación oportuna acerca del chip BIOS, sobre el cual se lograron varios avances y mejoras en la funcionalidad que nunca se contemplaron para esta práctica pero fueron necesarios dado a las condiciones en que se presentaban las fallas; avances que se presentan en este documento y agregan valor en la experiencia de manejo sobre dispositivos embebidos de todo tipo.

Nuevos objetivos que no incluía esta práctica respecto al sistema de censado de carga de la batería del dispositivo y manejo del procesador de los dispositivos de comunicación vía

internet se han venido alcanzando junto al equipo de trabajo en la FCV; objetivos que proponen mejoras respecto a fallas en diseño de PCBs y potencia, así como monitores con acceso a comunicación inalámbrica.

Lenguajes de programación software y uso de dispositivos electrónicos de cualquier gama están a la disposición real de cualquier ingeniero electrónico que pueda apartar una buena parte de su tiempo, dedicación a estudiar las diferentes hojas de datos, hacer las respectivas investigaciones y aportar parte de su conocimiento y lógica cotidiana en el manejo de los mismos, como en este caso la adopción de nuevas tecnologías logra darle un giro al proyecto superando los inconvenientes evidenciados a lo largo de la práctica. Habilidades de investigación y desarrollo de proyectos que se forman e impulsan día a día en las diferentes asignaturas que entrega la Universidad Pontificia Bolivariana - Seccional Bucaramanga hacia sus estudiantes; no hay lugar para temores ni falta de experticia, ya que cada persona posee un pequeño experto dentro de sí buscando sobresalir a través de la formación, solo hay que intentarlo una y otra vez.

La comparación, comprobación, depuración y perfeccionamiento en un proceso de desarrollo es probablemente la etapa más extensa del mismo, subprocesos de implementación, pruebas, rediseños se ven evidenciados a lo largo de esta práctica y gran variedad de empresas, para lograr la correcta acción de las funcionalidades de sus productos y que al lanzarlos al mercado sean un éxito. Ha de saberse que la Fundación Cardiovascular de Colombia desea ingresar en el campo de la electrónica y destacar sus productos en el área de la telemedicina guiados por la prestación de los servicios que ofrece la UEE-Bioingeniería, sin embargo y sabiendo que es una empresa dedicada netamente a establecer productos y servicios de salud, los cuales opacan un poco el avance de las nuevas tecnologías, la organización ha prestado un apoyo incondicional a la UEE-Bio porque confía y da aval a los ingenieros locales en este tipo de proyectos para así impactar en toda América Latina.

Por último y dejando de lado los conceptos electrónicos, el manejo de las situaciones con el equipo de trabajo y el personal asistencial en la FCV fueron bastante enriquecedores para lograr un conocimiento más amplio acerca del manejo de los recursos en una empresa, horarios, disponibilidad del trabajador, la importancia del paciente en un entorno clínico sin importar los sectores de actuación del trabajador, entre otras ganancias de tipo cualitativo a manera de experiencia profesional que se logran con esta práctica.

BIBLIOGRAFÍA

Comunidad de cplusplus. Base de referenciación software alusiva al código C++ - Múltiples aportes de la comunidad en general y ejemplificación en línea sobre el lenguaje C++. Internet: (en.cppreference.com/w/ <<http://en.cppreference.com/w/>>).

Corporación Altera. Múltiple material de documentación, manuales de uso y guías de desarrollo sobre dispositivos electrónicos fabricados por la compañía Altera. Internet: (www.altera.com/literature/lit-index.html <<http://www.altera.com/literature/lit-index.html>>).

Fundación Cardiovascular de Colombia. Master de documentos servidor – Base de datos restringida para empleados, dedicada a albergar la gestión del conocimiento de la organización a través de previos documentos que los diferentes colaboradores expiden como parte de su trabajo. Floridablanca Santander Colombia.

MOYER, Bryon. Real World Multicore Embedded Systems, Waltham MA, Elsevier, 2013, 648 p.

NOERGAARD, Tammy. Embedded Systems Architecture - A Comprehensive Guide for Engineers and Programmers, Burlington MA, Elsevier, 2005, 656 p.

Personal de cplusplus. Completo blog informativo y alusivo a lenguajes de programación C y C++ - Referencias, guías, foros, material descriptivo de las librerías clave utilizadas para el desarrollo sobre código C y C++, entre otros. Internet: (www.cplusplus.com <<http://www.cplusplus.com>>).

Personal de stackoverflow. Base de programación referida a todo tipo código software, compilación y otros cuestionamientos - Múltiples referencias y ejemplificación en línea. Internet: (stackoverflow.com/ <<http://stackoverflow.com/>>).

Personal de WiBit.Net. Clases de programación software referidas al código C++ - Curso gratuito de software en línea. Internet: (www.wibit.net/course/CPlusPlus <<https://www.wibit.net/course/CPlusPlus>>).

STRINGHAM, Gary. Hardware/Firmware Interface Design: Best Practices for Improving Embedded Systems Development, Burlington MA, Elsevier, 2010, 376 p.

SIMPSON, Philip. FPGA Design - Best Practices for Team-based Design, San Jose CA, Springer, 2010, 168 p.

Texas A&M University. Procedimientos para el manejo seguro de baterías de litio polímero – Manual de buenas prácticas y procedimientos que indican cuestionamientos tan importantes como la temperatura, carga de almacenamiento y posibles errores de las personas al manipular y disponer de dispositivos químicos de almacenamiento de energía litio-polímero. Internet: (oes.tamu.edu/web/guidelines/battery/LiPo%20Procedures.pdf/
<<http://oes.tamu.edu/web/guidelines/battery/LiPo%20Procedures.pdf>>).