PRÁCTICA EMPRESARIAL: PROTOTIPO DE MONITOR DE SIGNOS VITALES USANDO UN SISTEMA EMBEBIDO BASADO EN LINUX

FUNDACIÓN CARDIOVASCULAR DE COLOMBIA

SHERNEYKO PLATA RANGEL

UNIVERSIDAD PONTIFICIA BOLIVARIANA
ESCUELA DE INGENIERÍA
FACULTAD DE INGENIERIA ELECTRONICA
BUCARAMANGA
2013

PRÁCTICA EMPRESARIAL: PROTOTIPO DE MONITOR DE SIGNOS VITALES USANDO UN SISTEMA EMBEBIDO BASADO EN LINUX

SHERNEYKO PLATA RANGEL

Práctica empresarial para obtener el título de

INGENIERO ELECTRÓNICO

Supervisor académico:
Mag. SERGIO ALEXANDER SALINAS

Supervisor empresarial:
Ing. LEONARDO RODRÍGUEZ SALAZAR

UNIVERSIDAD PONTIFICIA BOLIVARIANA
ESCUELA DE INGENIERÍA
FACULTAD DE INGENIERIA ELECTRONICA
BUCARAMANGA
2013

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

RESUMEN

TITULO: **PRÁCTICA EMPRESARIAL: PROTOTIPO DE MONITOR DE SIGNOS VITALES USANDO UN SISTEMA EMBEBIDO BASADO EN LINUX**

AUTOR: **SHERNEYKO PLATA RANGEL**

FACULTAD: **FACULTAD DE INGENIERIA ELECTRONICA**

DIRECTOR: **SERGIO ALEXANDER SALINAS**

La presente práctica fue desarrollada en la Unidad Estratégica de Negocios, departamento de Bioingeniería de la Fundación Cardiovascular de Colombia, la cual tiene como misión "contribuir al desarrollo científico y tecnológico en salud, mediante la producción de conocimiento, innovación, transferencia y apropiación de tecnologías, dirigidas al mejoramiento de las condiciones de vida de la población colombiana". El principal objetivo de esta práctica fue desarrollar controladores para un sistema embebido, de las tarjetas de adquisición de datos OEM (*Original Equipment Manufacturer*), que junto a la implementación de una interfaz gráfica de usuario (GUI), muestren los diferentes signos vitales obtenidos, usando un monitor con puerto DVI-D. Para este diseño se utiliza la tarjeta de desarrollo *Pandaboard*, basada en un SoC OMAP4. Como tarea adicional el desarrollo de módulos hardware, para otros equipos en desarrollo, basados en FPGAs.

Basándose en la información obtenida de la documentación de las tarjetas de adquisición de parámetros fisiológicos, se procede a desarrollar controladores con la capacidad de establecer comunicación con las mismas (decodificación y codificación de datos). Posteriormente se inicia la implementación de la interfaz gráfica, inicialmente en un ordenador portátil, luego portada hacia el sistema embebido, donde se concluye su desarrollo, y se integra con los controladores, dando así como resultado una interfaz gráfica donde se muestran los datos obtenidos por cada una de las tarjetas OEM que pueden conformar un monitor de signos vitales.

De forma simultánea, se desarrollaron módulos hardware con sus respectivos controladores para el equipo de gama media en etapa avanzada de desarrollo.

Las actividades fueron realizadas satisfactoriamente, bajo la supervisión del jefe de Diseño y Desarrollo, y constituyen un recurso intelectual importante tanto en el desarrollo de equipos médicos basados en sistemas embebidos con sistema operativo GNU/Linux, como en el desarrollo modular de equipos basados en FPGA.

PALABRAS CLAVES:

ARM, GNU / LINUX, INTERFAZ GRAFICA DE USUARIO, TARJETA OEM, SIGNOS VITALES, FPGA, OMAP4

VoBo DIRECTOR DE TRABAJO DE GRADO

ABSTRACT

TITLE: **INTERNSHIP: VITAL SIGNS MONITOR USING A LINUX-BASED EMBEDDED SYSTEM**

AUTHOR: **SHERNEYKO PLATA RANGEL**

FACULTY: **ELECTRONIC ENGINEERING FACULTY**

DIRECTOR: **SERGIO ALEXANDER SALINAS**

This internship had place at the Strategic Business Unit, part of the department of Bioengineering of the cardiovascular foundation of Colombia, whose mission is “contributing to the scientific and technologic development in the health sector, by producing knowledge, innovation, transference and appropriation of technologies, towards the improvement of the life conditions of the Colombian population”. This internship main objective was developing drivers for an embedded system, capable of communicating with the OEM (Original equipment manufacturer) data acquisition modules, which together with the implementation of a GUI (Graphical user interface), can be able to display their vital signs, using a screen with DVI-D port. This design uses a development single-onboard computer, called Pandaboard, based in a System-on-a-chip (SoC) OMAP4. As an additional, also the development of hardware description of task specific modules, to be included in current developments FPGA based.

Based in the obtained information, from the physiologic acquisition cards documentation, proceeds the development of driver capable of establishing communication with the cards (codification and decodification of data). Thereafter the implementation of the graphic interface starts, initially in a laptop (running Ubuntu Linux), then ported to the embedded system, where its development concludes, and also is integrated with the drivers previously implemented, giving as a result a graphic interphase capable of showing data obtained from each of the OEM Modules, that may form a high range vital signs monitor.

Simultaneously, the development of two hardware modules with their respective controllers, for the mid-range monitor, (the one in advanced stages of development currently) took place,

The activities were developed successfully, under the supervision of the design and development team leader. These activities are important intellectual resources, as in the development of medical equipment based in embedded systems with GNU/Linux, as in the modular development of FPGA based equipment

KEYWORDS:

ARM, GNU / LINUX, FPGA, OMAP4,
GRAPHICAL USER INTERFACE,
OEM MODULE, VITAL SIGNS,

VoBo THESIS DIRECTOR

INTRODUCCIÓN

La unidad estratégica de negocios (UEN) de la Fundación Cardiovascular de Colombia (FCV) tiene como objetivo el desarrollo de dispositivos médicos enfocados hacia la supervisión y control de parámetros fisiológicos, entre ellos, se encuentra en producción el monitor de signos vitales MSV1300 y la unidad de cuidados intensivos móvil UCIM 1200 situados hoy en día en muchos hospitales y clínicas a nivel nacional, (incluyendo departamentos como Santander, Choco , Magdalena ,San Andrés y Providencia , a nivel local ubicados en el Instituto del Corazón), los cuales se usan principalmente para atender a pacientes con problemas cardiovasculares, adquiriendo datos como Electrocardiografía (ECG) , Pulsioximetría (SpO2), Presión no invasiva (PNI), Respiración (RESP) y Temperatura (TEMP).En base a estos productos se desarrollan nuevos equipos con capacidades extendidas, orientados a monitoreo de pacientes tanto de bajo como de alto riesgo (de lo cual dependen los parámetros a monitorear, y la calidad necesaria en las señales obtenidas para un adecuado diagnostico).

Los sistemas en producción actualmente en la UEN son basados en PCs convencionales, acarreando las desventajas de los mismos, algunas veces su temperatura es elevada y sufren de bloqueos, todo debido a que estos no se encuentran diseñados para un funcionamiento continuo. Además, la adquisición de datos en algunas situaciones puede llegar a ser deficiente, como por ejemplo no filtrar frecuencias de equipos médicos usados durante intervenciones quirúrgicas (tales como el electrobisturí), o patologías cuyas señales eléctricas son muy débiles (sobre todo cardiacas), estos son problemas que impiden su uso en salas de cirugía. Se ha trabajado entonces en el desarrollo de equipos basados en FPGAs, cuyo desempeño y reducido consumo puede dar mayor confiabilidad a este tipo de dispositivos. Ahora bien, observando las tendencias de los equipos médicos desarrollados recientemente, se ve un extenso crecimiento en el uso de sistemas embebidos basados en GNU/Linux, específicamente en procesadores con arquitectura ARM, por lo cual se toma esta tecnología como alternativa al desarrollo en FPGAs, de mano del SoC OMAP4, fabricado por Texas Instruments.

La arquitectura ARM (*Advanced Risc Machine*), se basa en la estrategia de diseño RISC (*Reduced Instruction Set Computer*), y se caracteriza por su reducido consumo eléctrico, lo cual lo hace apropiado para ser usado en dispositivos portátiles, haciendo de esta familia de microprocesadores de 32 bits una de las más utilizadas.

OMAP4, por otra parte, es un SoC (*System On A Chip*) que a grandes rasgos incluye un procesador ARM Cortex, un acelerador grafico 3D, 2D, un ISP (*Image Signal Processor*), junto a los dispositivos y hardware controlador básicos encontrados en cualquier tarjeta madre de un PC convencional, en un solo chip, permitiendo así un significativo ahorro de espacio, potencia y tiempo de desarrollo, ya que estas características facilitan el diseño de las tarjetas donde se use (se reduce el número de chips, de conexiones y probablemente, de capas).

Gracias a que se ha implementado este SoC en una tarjeta de desarrollo *Open Hardware*, y en ánimos de evaluar las capacidades como alternativa al desarrollo de equipos médicos innovadores , la UEN Bioingeniería ha encargado la tarea de realizar un desarrollo inicial a un estudiante de la Universidad Pontificia Bolivariana, posibilitando que él pueda aportar al desarrollo de un monitor de signos vitales orientado a cuidado crítico, basado en un SoC OMAP4, tarea que fue realizada satisfactoriamente : mediante una interfaz gráfica de usuario (GUI) se logró mostrar los diferentes parámetros fisiológicos relacionados con cuidado critico (ECG, SPO2, RESP, TEMP, PNI, PI y CO2) y a su vez interactuar con los diferentes módulos OEM especializados en la adquisición de dichos parámetros.

Adicionalmente se desarrollaron algunos módulos hardware con sus respectivos drivers, para el monitor basado en FPGAs, actualmente en desarrollo, los cuales se integraron en el sistema de forma satisfactoria.

1. DESCRIPCION DE LA EMPRESA

1.1 RESEÑA HISTORICA

La historia de la Fundación Cardiovascular de Colombia se remonta al año 1982 cuando el *Variet Childrens Life*, inspiró la creación del comité Corazón a Corazón de Nueva York liderada en Colombia por el Dr. Franklin Roberto Quiros. Siguiendo esta pauta, en 1985 un grupo de especialistas y personalidades de Bucaramanga se propuso crear una entidad privada sin ánimo de lucro dedicada a tratar las enfermedades del corazón, logrando que un grupo de médicos iniciara las actividades de consulta y prueba de esfuerzo en la Fundación Tercera Edad de la Congregación Mariana, y las primeras cirugías cardiovasculares en la Clínica Bucaramanga en 1987.

En el año de 1992 esta entidad entró a formar parte de la Clínica Carlos Ardila Lulle, ubicándose en el cuarto piso de sus instalaciones, lo cual permitio ampliar todos los servicios diagnósticos e intervencionistas de cardiología y cirugía vascular periférica, utilizando salas de cirugía, unidad de cuidados intensivos y hospitalización.

Posteriormente, en octubre de 1997 se inauguró la nueva sede del Instituto del Corazón, un edificio de 14 pisos con capacidad para 123 camas de hospitalización distribuidas entre la unidad de Cuidados Intensivos Post-quirúrgica, unidad de Cuidados Intensivos Pediátrica, unidad de Cuidados Intermedios Adultos, tres pisos de hospitalización, 4 salas de cirugía, 2 salas de Hemodinámica y 1 del servicio de urgencias durante las 24 horas del día cumpliendo así con todos los requisitos y normas exigidas por el Ministerio de salud, relacionadas con enfermedades cardiovasculares. Obtiene en este mismo año el Premio nacional de Cardiología; y se crea la Corporación Instituto Colombiano de investigaciones Biomédicas (ICIB).

En el año 2000, se propone la diversificación concentrada en la satisfacción de las necesidades del sector Salud, creando 5 nuevas empresas (unidades estratégicas de negocios) FCV Software, FCV Comercializadora, FCV Administración Hospitalaria, FCV Productos Hospitalarios y FCV Instituto de Investigaciones.

En el año 2006 Se inaugura el Centro Tecnológico Empresarial, con más de 5500 metros cuadrados construidos, espacio otorgado a las unidades estratégicas de negocio productivas con el fin de desarrollar tecnología y conocimiento, entre ellas la UEN Bioingeniería. También se inicia el desarrollo de los productos base de esta unidad estratégica de negocios, orientados a suplir de una mejor forma las necesidades de las entidades de salud a nivel nacional.

En el 2007 la Fundación Cardiovascular de Colombia recibe la visita de Recertificación ISO 9001 por parte del ente certificador ICONTEC, abre su nueva sala de neonatos, la nueva unidad de cuidado crítico y se consolida la UEN FCV Telemedicina como la más grande institución en este campo a nivel nacional. Su estructura actual se muestra en la figura 1.

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

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

Datos de la empresa:
El instituto del corazón se ubica actualmente en la Calle 155A #23-58, Floridablanca, Santander. Su número telefónico es 6399292.

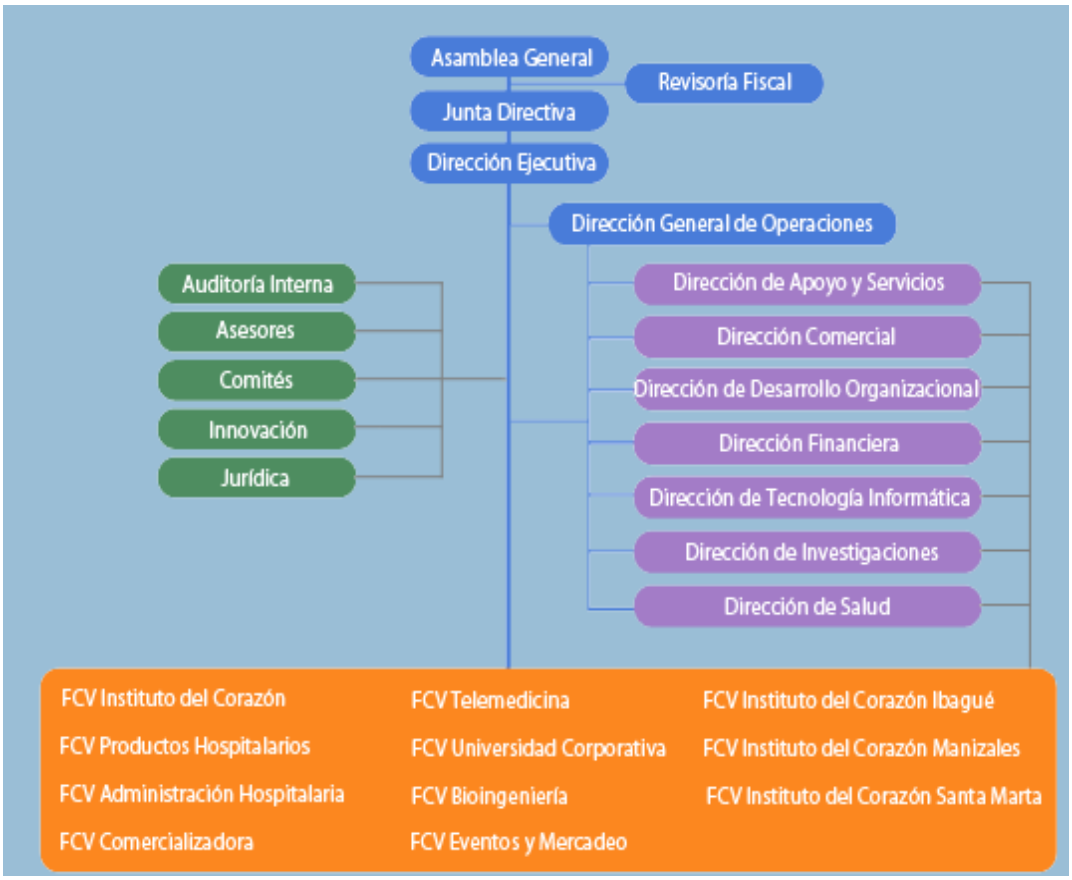

Figura 1. Organigrama de la Fundación Cardiovascular [1]

1.2. DESCRIPCION DEL AREA DE PRÁCTICA

Misión:

La UEN FCV Bioingeniería es una unidad estratégica empresarial de la Fundación Cardiovascular de Colombia, que contribuye al desarrollo científico y tecnológico en salud, mediante la producción de conocimiento, innovación, transferencia y apropiación de tecnologías, dirigidas al mejoramiento de las condiciones de vida de la población colombiana.

Visión:

En el año 2020 la unidad estratégica empresarial FCV Bioingeniería será una entidad reconocida a nivel nacional e internacional por la confiabilidad, calidad e innovación en sus desarrollos y productos.

Datos de la UEN FCV Bioingeniería:

El centro de desarrollo tecnológico-empresarial, donde se encuentra la UEN Bioingeniería, está ubicado en la carrera 5 #6-33, Floridablanca, Santander. Su número telefónico es 6497304 ext. 4151

1.3 DIAGNOSTICO DE LA EMPRESA

La FCV actualmente proporciona equipos médicos de calidad, pioneros en la región, orientados al bienestar de la comunidad. Su unidad estratégica de negocios de Bioingeniería, ha realizado estudios y determinado las características de los nuevos equipos a desarrollar, en base principalmente a la retroalimentación obtenida por las entidades de salud, usuarias de dichos equipos. Siguiendo la creciente tendencia del mercado actual, se ha identificado la necesidad de implementar equipos médicos sobre sistemas embebidos basados en procesadores ARM, bajo el sistema operativo GNU/Linux. Los procesadores ARM se caracterizan por su consumo, precio y temperatura, características que les permiten ser incorporados en electrónica de consumo, sin tomar en cuenta la enorme ventaja de trabajar en base a un sistema operativo preexistente, con muchas características básicas ya incorporadas y funcionales.

Bioingeniería actualmente está produciendo monitores basados en PC convencionales, esto ha probado ser problemático debido a la poca estabilidad, gran consumo energético y necesidad de licencias para estos sistemas. Debido a esto, se inició hace aproximadamente un año el desarrollo de nuevos equipos basados en FPGAs, aun así es necesario abordar tecnologías nuevas que puedan llevar a un desarrollo más rápido y producción más económica.

La UEN Bioingeniería de la FCV acepta al practicante con el objetivo de realizar un primer acercamiento al tema, enfocado al desarrollo de un equipo médico, evaluando las capacidades de la tecnología para dicha tarea y permitiendo al estudiante en proceso de grado aplicar sus conocimientos.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Desarrollar librerías software para los diferentes dispositivos de la *Pandaboard* así como una GUI (Interfaz Gráfica de Usuario) orientada a mostrar signos vitales desde tarjetas OEM(*Original Equipment Manufacturer*).

2.2 OBJETIVOS ESPECIFICOS

- Demostrar las capacidades de los sistemas embebidos basados en Linux hacia el desarrollo de equipos biomédicos.

- Implementar el sistema operativo Linux en la *Pandaboard*, así como las librerías software para comunicación con sus dispositivos, usando ANSI-C.

- Desarrollar una interfaz gráfica con capacidad de mostrar los parámetros de una tarjeta OEM (*Original Equipment Manufacturer*) de signos vitales

- Implementar las aplicación basada en GTK (*Gimp Tool Kit*) en la *Pandaboard* y evaluar sus capacidades usando la salida de video

- Apoyar el desarrollo de módulos o tareas específicas de proyectos afines usando FPGAs

2.3 ACTIVIDADES A DESARROLLAR

- Estudiar las funcionalidades de la tarjeta de desarrollo *Pandaboard*

- Estudiar e implementar Linux sobre la plataforma *Pandaboard*, utilizando el terminal disponible en el puerto RS232 y hacer uso de los periféricos básicos, diseñando drivers utilizando C y Python (Leds, SD, USB)

- Portar las librerías Graficas (GTK) Linux, para diseñar una Interfaz Gráfica sobre la plataforma utilizando la salida de video

- Diseñar una GUI(sobre Linux) que muestre los parámetros fisiológicos arrojados por una tarjeta OEM de signos vitales

- Apoyar en el diseño y el desarrollo de otros dispositivos Médicos

2.3.1. Cronograma de actividades

Tareas	Semana															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Estudiar las funcionalidades de la tarjeta de desarrollo *Pandaboard*																
Estudiar e implementar Linux sobre la plataforma *Pandaboard*, utilizando el terminal disponible en el puerto RS232 y hacer uso de los periféricos básicos, diseñando drivers utilizando C y Python (Leds, SD, USB)																
Portar las librerías Graficas(GTK) Linux, para diseñar una Interfaz Gráfica sobre la plataforma utilizando la salida de video																
Diseñar una GUI(sobre Linux) que muestre en tiempo real los parámetros fisiológicos arrojados por una tarjeta OEM de signos vitales																
Apoyar en el diseño y el desarrollo de otros dispositivos médicos																

Tabla 1. Cronograma de actividades

3. MARCO TEORICO

3.1 TARJETAS OEM DE SIGNOS VITALES

Los dispositivos OEM son hardware producido por una compañía con el objetivo de ser revendido o incorporado en un equipo bajo la marca del comprador del dispositivo. En este caso, las tarjetas OEM de adquisición de datos (parámetros fisiológicos), poseen las características necesarias para tomar señales físicas y eléctricas generadas por el cuerpo humano, procesarlas e interpretarlas, convirtiéndolas así en información útil, parámetros como electrocardiografía (ECG), pulsioximetría (SpO2), respiración (RESP), temperatura (T°), presión invasiva (PI), presión no invasiva (PNI) y Capnografía (CO2), de importancia en el área médica de cuidado crítico. Las tarjetas se encargan de realizar la adquisición y filtrado de las señales usando diferentes sensores, enviando esta información a un dispositivo host usando un protocolo de comunicación serial (RS232, convertido a USB usando módulos FTDI) y una capa de comunicación que permite la verificación de la integridad de los paquetes, el envío de diferentes tipos de datos y comandos.

Estas tarjetas provienen de empresas reconocidas en el ámbito médico, y generalmente cumplen con los estándares electrónicos y médicos necesarios para su uso y comercialización a nivel internacional, siendo esto ventajoso ya que se reduce la probabilidad de errores (en tareas tan criticas como el monitoreo de pacientes), permiten al equipo ser modular (facilitando así su mantenimiento y reparación) y ahorran tiempo de desarrollo, ya que la parte de procesamiento, adaptación y filtrado de las señales viene implementada desde el fabricante, e incluso en algunas los accesorios a usar para tomar las señales vienen incluidos.

3.2 NORMAS PARA EL DISEÑO DE DISPOSITIVOS MEDICOS

La UEN Bioingeniería se encuentra certificada en las normas ISO 9001 y ISO 13485, sin embargo para posibilitar la comercialización de los equipos en otros mercados, estos deben basarse en la norma AAMI (*Association for the Advancement of Medical Instrumentation*), específicamente en AAMI 60601-1-2:2007(Compatibilidad electromagnética) y AAMI HE75:2009 (Ingeniería de factores humanos).

La asociación para el avance de la instrumentación médica (AAMI), tiene como objetivo el desarrollo seguro y efectivo de tecnología médica. Es la organización líder en su campo con más de 7000 individuos, hospitales y fabricantes de dispositivos médicos miembros.

Se encarga de desarrollar, revisar y mejorar estándares a ser alcanzados por los dispositivos médicos.

3.3 ARQUITECTURA ARM

Arquitectura desarrollada por Acorn Computers en 1983 con el objetivo de usarse en computadoras personales. Se caracteriza por manejar un conjunto de instrucciones simple (RISC) y por el bajo número de transistores necesarios para su implementación (lo cual lleva también a un bajo consumo energético). La arquitectura y sus procesadores son licenciados a diversas empresas para permitirles incluirlos en sus diseños, aplicables desde electrónica de consumo (Apple, RIM, Samsung, LG) hasta la industria de semiconductores y aplicaciones industriales (Texas Instruments, Qualcomm, NXP, STMicroelectronics).[2]

El auge de los dispositivos móviles y sistemas embebidos, ha incrementado el uso de esta arquitectura, promoviendo su compatibilidad con las nuevas versiones de sistemas operativos (Windows 8, Windows RT y gran cantidad de distribuciones GNU/Linux).

A nivel técnico, la arquitectura ARM posee instrucciones de tamaño fijo, y el acceso a memoria de datos está restringido a las instrucciones de carga y almacenamiento. Esto hace que la arquitectura sea más predecible. También se caracteriza por dividir instrucciones complejas en partes más sencillas, de menor duración. A pesar de requerir mayor número de líneas para realizar ciertas tareas en comparación a una arquitectura CISC, esto es de poca importancia ya que el incremento en el tamaño de los archivos binarios es insignificante en comparación a las capacidades de almacenamiento actuales, y además la traducción a código máquina, como en muchas otras arquitecturas es realizada por un compilador.[3]

En la figura 2 se muestra la diferencia en el modelo de aplicación de los procesadores Intel (CISC x86) vs ARM (RISC ARM).

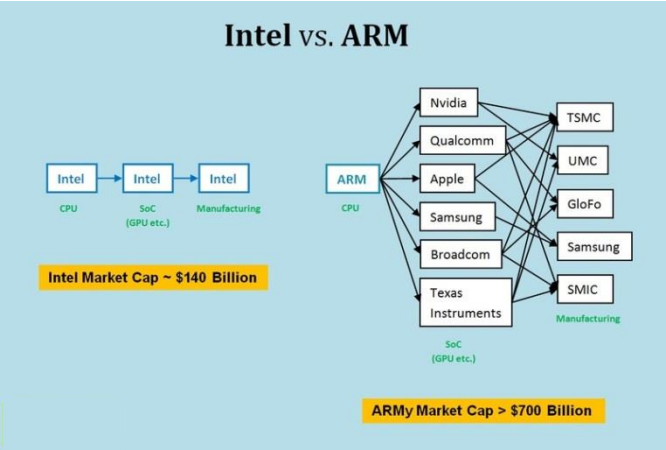

Figura 2. Modelo de aplicación de Intel vs ARM [4]

3.4 SoC

Circuito integrado que posee múltiples componentes de un ordenador personal u otro sistema electrónico dentro de un solo chip. Usualmente contienen:

- Uno o varios microprocesadores
- Bloques de memoria, interfaces para memorias externas (volátiles y no volátiles)
- Relojes
- PLLs
- Contadores
- RTCs
- Interfaces externas siguiendo estándares industriales como USB, Firewire, Ethernet, UART, USART, SPI
- Interfaces análogas (ADC,DAC)
- Reguladores de voltaje
- GPUs, Interfaces externas de audio y video

Su estructura es modular (ver figura 3), suelen tener la capacidad de ejecutar versiones de escritorio como Windows o GNU/Linux. Tienen una relación muy cercana con los procesadores ARM y los sistemas embebidos (su principal aplicación). Son más económicos, sencillos de producir y de implementar en comparación a un sistema modular (PC de escritorio, por ejemplo).

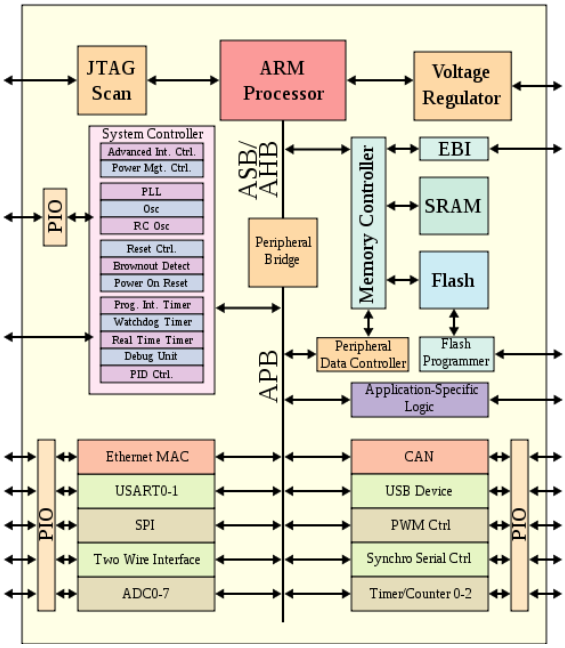

Figura 3. Diagrama de bloques de un SoC[5]

3.5 GNU/LINUX

Sistema operativo derivado de UNIX, de licencia libre y abierta, basado en el núcleo Linux e integrado con las utilidades del proyecto GNU. Fue desarrollado inicialmente para ordenadores personales basados en Intel-x86, sin embargo ha sido portado a muchas otras arquitecturas [6]. Es líder en el mercado de los servidores, mainframes y supercomputadores. En los últimos años también se ha integrado en sistemas embebidos, siendo las distribuciones más populares en los mismos Android y derivados de Debian (Ubuntu). El proyecto ha sido desarrollado colaborativamente, su código puede ser usado, modificado y distribuido, de forma comercial o no comercial, por cualquiera bajo la licencia GNU. El sistema operativo es modular y como tal puede funcionar con un sinnúmero de configuraciones según las necesidades del usuario, los recursos disponibles en el equipo (generalmente no existe un mínimo de especificaciones para correr el sistema ya que esto depende de los módulos que se usen), y las aplicaciones instaladas. Se hace énfasis en el uso de gran cantidad de pequeños programas con tareas específicas, en vez de grandes programas con tareas globales (a diferencia de otros sistemas operativos), y también en el paradigma de las familias UNIX: todo es un archivo (*"Everything is a file"*).[7]

Inicialmente fue escrito en ASM, una práctica común en la época, sin embargo fue reescrito en C (excepto el núcleo y las entradas/salidas) lo cual permitió que este sistema fuera portable a diferentes plataformas computacionales. Actualmente existen múltiples distribuciones con diferentes enfoques y aplicaciones, su licencia libre ha propiciado su expansión en muchos países y gobiernos que lo ven como una alternativa transparente y más económica a los sistemas operativos privativos[8].

Dentro de sus componentes básicos se encuentran:

- *Bootloader*: (GRUB, LILO) programa que se ejecuta cuando el computador es encendido, y carga el núcleo de Linux en memoria
- *Init*: proceso lanzado por el núcleo Linux, raíz de todos los demás procesos. Todo proceso es lanzado por *init*, desde servicios del sistema hasta diálogos de *login*
- Librerías software: contienen código que puede ser utilizado por procesos en ejecución. La más usada es la librería GNU C Library
- Aplicaciones de interfaz de usuario sean gestores de ventanas (gráfico) o consolas (texto)

3.6 GTK+

Gimp Toolkit, es un set de herramientas multiplataforma para crear interfaces gráficas. Esta licenciado bajo LGPL, permitiendo su uso tanto por software libre como propietario. Originalmente diseñado para dotar de funcionalidad al programa de manipulación de imágenes GIMP, fue mejorado para tener la capacidad de crear interfaces gráficas. [9]

Este set de herramientas está basado en objetos, y se encuentra escrito en el lenguaje de programación C. A pesar de que se encuentra diseñado para ser usado en plataformas que usen el sistema de ventanas X, funciona en otras plataformas incluyendo Windows y Mac OS. Su implementación base está hecha en C, y existen *bindings* a otros lenguajes (C++, Java, Perl, Python, por nombrar algunos), sin embargo estos *bindings* en su mayoría no poseen todas las características de la librería original.

Las capacidades de este set de herramientas se demuestran en varios entornos gráficos usados en GNU/Linux. Sus elementos gráficos son llamados Widgets, visualmente son similares a los encontrados en otros sistemas operativos (figura 4). Es utilizado en Gnome, XFCE y LXDE, entornos de escritorio con prioridades muy distintas (Mientras la prioridad de Gnome es apariencia, LXDE se enfoca en rendimiento). Los programas hechos en GTK+ no requieren que el entorno del escritorio donde se ejecutan este basado en GTK+.

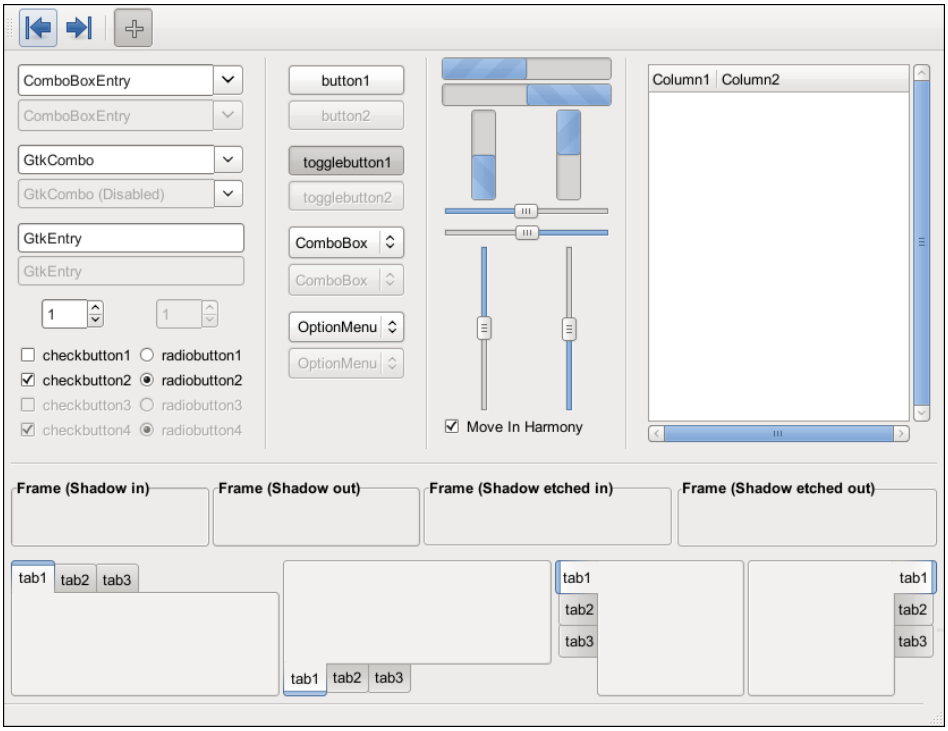

Figura 4. Widgets Básicos de GTK+ [10]

3.7 PANDABOARD

Es una placa ordenador (*Single-board computer*) de bajo consumo y precio. Está basada en el SoC OMAP 4460 de Texas Instruments, que contiene un procesador ARM Cortex-A9 MPCore. Es utilizada en etapas tempranas de desarrollo de sistemas embebidos basados en la tecnología OMAP. Tiene la capacidad de correr múltiples distribuciones Linux portadas a ARM. Posee los siguientes puertos:

- HDMI
- DVI-D
- USB
- SD
- Mini-AB USB
- E/S de audio 3.5mm
- Puerto DB9 para *Debugging*
- Puertos de expansión para dispositivos opcionales (cámara, LCD, etc)

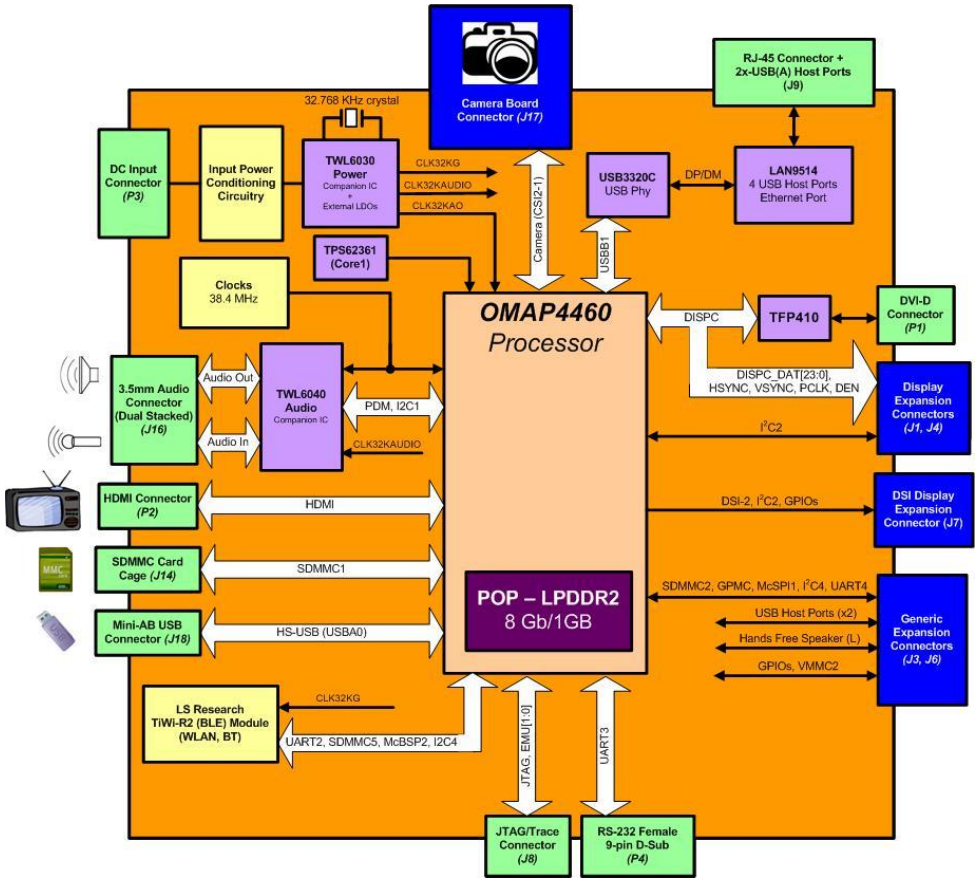

Figura 5. Diagrama de bloques de la tarjeta Pandaboard [11]

3.8 OMAP

Open Multimedia Applications Platform, es una categoría de SoCs desarrollada por Texas Instruments, para aplicaciones móviles portables. Generalmente incluyen un procesador ARM de propósito general, trabajando junto a coprocesadores especializados (figura 6). Se encuentran agrupados en 3 grupos dependiendo de su rendimiento:

- Procesadores para aplicaciones de alto rendimiento

  Fueron desarrolladas inicialmente para uso en *smartphones*, con procesadores lo suficientemente poderosos para correr diversos sistemas operativos (como Linux o Symbian), poseen soporte para conectividad con ordenadores personales, y soportan también diversas aplicaciones de audio y video. En esta categoría se encuentran los SoC OMAP1, OMAP2, OMAP3, OMAP4 y OMAP5. Son utilizados en numerosas tarjetas de desarrollo, como la *Beagleboard*, *Pandaboard* y *Gumstix*. [12]

  Poseen una GPU, un ISP (*Image Signal Processor*), un acelerador multimedia e interfaces para comunicación con dispositivos externos como HDMI, SD, USB, SIM, SDRAM, memoria FLASH, GPS, WiFi, Bluetooth, y protocolos como I2C, SPI y UART.

- Procesadores para aplicaciones multimedia básicas

  Solo se comercializan a fabricantes de teléfonos celulares. Se pretende que sean chips de alta integración y bajo precio, para productos de consumo.
  Por otra parte la serie OMAP-DM está orientada a ser usada como coprocesador multimedia para dispositivos móviles con alta resolución y video cámaras. [13]

- Procesadores para aplicaciones con modem integrado

  Solo se comercializan a fabricantes de teléfonos celulares. Muchas de las versiones nuevas están altamente integradas para su uso en terminales de bajo costo. [14]

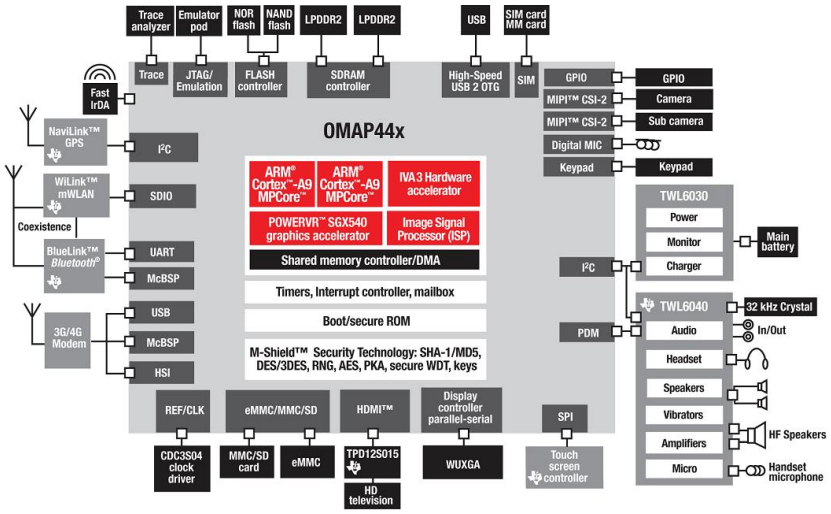

Figura 6. Diagrama de bloques de la serie OMAP44x[31]

3.9 POWERVR SGX

GPU (*Graphical Processing Unit*) diseñada por *Imagination Technologies*, quien en los últimos años sigue el modelo de ARM Holdings, licenciando sus desarrollos a empresas para permitirles integrarlos dentro de sus circuitos integrados. La serie SGX posee soporte para *Pixel shader*, *Vertex shader* y *Geometry shader*, y es compatible con APIs para *rendering* 2D y 3D como OpenGL ES y DirectX. Es muy popular en integrados SoC, usados en dispositivos móviles, entre ellos: Ax de Apple, OMAP de Texas Instruments, Hummingbird de Samsung y Medfield de Intel.[15]

3.10 FPGA

Circuito integrado programable, con capacidad de implementar cualquier función lógica que un ASIC pueda implementar. Son configurados en base a un lenguaje de descripción de hardware (HDL), de forma similar a los circuitos ASIC. Sus capacidades de actualizar su funcionalidad después de fabricada, su bajo coste en comparación a la fabricación en baja escala de un ASIC, y su alto paralelismo ofrece ventajas para varias aplicaciones.

Las FPGA contienen elementos lógicos programables, y un conjunto de conexiones reconfigurables que permite a los bloques conectarse entre sí, permitiéndoles interactuar de múltiples formas. Las operaciones que estos pueden realizar van desde operaciones sencillas binarias, como AND y XOR, hasta funciones complejas secuenciales que hagan uso de *flip-flops* y bloques de memoria (figura 7). Además de estas características, muchas poseen elementos análogos que complementan las capacidades digitales, como la capacidad de configurar el *slew rate , drive stength* , usar PLLs y diversos estándares electrónicos digitales(TTL, CMOS, LVDS, ECL, HSTL, LVPECL).

Su funcionamiento difiere del de un microcontrolador o microprocesador; mientras que estos ejecutan una lista de instrucciones (programa), la FPGA implementa circuitos que cumplan un conjunto de funciones lógicas. Su capacidad de implementar diversas funciones de forma simultánea le permite ser más rápida que un microprocesador de mucha mayor frecuencia en algunas tareas, ya que este debe repartir su tiempo de procesamiento entre sus aplicaciones.

Debido a estas características, esta tecnología es ampliamente utilizada en el procesamiento digital de señales, en interfaces de alta velocidad, en aplicaciones militares (también por la fiabilidad del hardware respecto al software en alteraciones electromagnéticas), y para el desarrollo de electrónica de consumo (la cual después se lleva a ASIC).

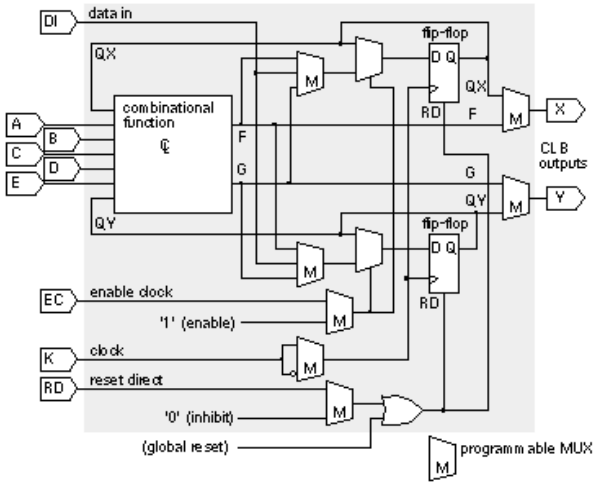

Figura 7. Estructura básica de un elemento lógico [16]

3.11 CONTROLADOR

Un controlador (*driver*), es un programa o librería que controla un tipo particular de dispositivo conectado al ordenador. Típicamente un controlador se comunica con el dispositivo a través del bus de sistema o subsistema de comunicación a través del cual se conecta el dispositivo hardware. Cuando un programa invoca una rutina del controlador, este envía comandos al dispositivo, y cuando el dispositivo envía datos al *host*, el driver decodifica dicha información para su uso en la aplicación. Los controladores difieren según el hardware, y usualmente son específicos de un sistema operativo.

El objetivo de un controlador es simplificar el trabajo del programador actuando como traductor entre un dispositivo hardware y las aplicaciones que lo usan. Los programadores pueden entonces escribir código de alto nivel independientemente del hardware que tiene el usuario final.

Existen tres tipos diferentes de controladores:

- Controlador de dispositivo físico (PDD) :

  Interactúa con el dispositivo a nivel hardware. Lee y escribe a direcciones del mismo asignadas a diversas tareas, con poca o nula intervención de un protocolo de transmisión. Se usa generalmente en dispositivos conectados directamente al microprocesador. [17]

- Controlador de dispositivo lógico (LDD):

  Interactúa con el dispositivo a nivel software. Se comunica mediante un estándar eléctrico y un protocolo software para empaquetar la información. Generalmente posee características para revisar integridad de paquetes y es bidireccional. [18]

- Controlador de dispositivo virtual (VDD):

  Emula la conexión de un dispositivo al sistema, para que este interactue con el mismo de forma estándar. Generalmente el otro extremo de la conexión posee un tipo de archivo o hardware especializado para interactuar como el dispositivo emulado. [19]

3.12 PROTOCOLO DE COMUNICACIÓN

Conjunto de reglas para el intercambio de mensajes entre sistemas computacionales. Un protocolo debe tener una descripción formal en la cual se describan: identificadores, datos, autenticación y detección de errores (como se muestra en la figura 8).

El protocolo también define la sintaxis, sincronización de la comunicación y parámetros de la capa física a utilizarse (muchas de ellas tienen parámetros que pueden diferir de equipo a equipo, como la velocidad de transmisión o el número de bits). Un protocolo puede implementarse como hardware, como software o como ambos, el objetivo es que se transmita y reciba siguiendo las mismas especificaciones. Una práctica común para lograrlo es el desarrollo de un estándar técnico (ejemplo de ello es el estándar USB).

La información intercambiada entre dispositivos se rige por reglas especificadas por un estándar que puede ser único o genérico dependiendo de la complejidad del dispositivo.

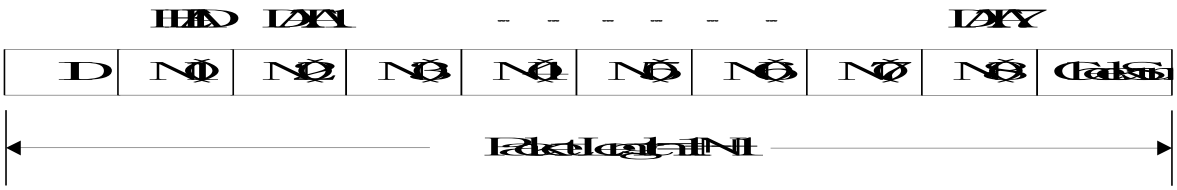

Figura 8. Ejemplo de protocolo de comunicación serial [20]

3.13 GCC

Sistema de compilación desarrollado por el proyecto GNU con soporte para múltiples lenguajes. Es un componente clave de las herramientas de programación de GNU. Se considera el compilador oficial del sistema GNU y ha sido adoptado por muchos sistemas operativos basados en UNIX, incluyendo Linux y la familia BSD. Es utilizado en muchos entornos de desarrollo (tanto libres como privativos) y está disponible para una gran cantidad de arquitecturas (x86, x64, ARM). Su uso esta tan extendido que los fabricantes de integrados consideran el hecho de portar GCC esencial para el éxito de una arquitectura. [19]

Se llamó originalmente GNU C Compiler, ya que solo estaba destinado a compilar el lenguaje C. Posteriormente se incluyó soporte para más lenguajes. Su funcionamiento es estándar en relación a otros compiladores UNIX. El usuario invoca el programa, llamado GCC, que interpreta los argumentos pasados al programa para archivo de entrada, corre el ensamblador en su salida y finalmente corre el *linker* para producir un ejecutable binario.

El *debugging* de los programas compilados con GCC se realiza generalmente con la herramienta GNU *Debugger* (GDB). Funciona desde línea de comandos, facilitando su integración dentro de cualquier IDE y se caracteriza por suministrar una notable cantidad de información de los ejecutables (siempre y cuando se compile en el modo de *debugging*). Para hallar fugas y errores de memoria, se usan programas más especializados (interrumpen el flujo de ejecución del programa, bajando su rendimiento) como Valgrind.

De forma más detallada, GCC tiene un ejecutable compilador para cada lenguaje, que toma código fuente y produce un árbol de sintaxis abstracta (AST), desde donde el proceso de compilación es idéntico para todos los lenguajes soportados. GIMPLE simplifica el código fuente en expresiones de no más de 3 operandos, combinaciones de declaraciones condicionales, y operadores *goto*. Luego se realiza otra conversión a SSA (single static assignment) donde se ajusta el comportamiento de las variables en memoria y se optimiza el código para su traducción final a código máquina. Se convierte de nuevo a GIMPLE donde se pasa a un árbol RTL (*Register Transfer Language*), que es una representación basada en hardware correspondiente a la arquitectura destino. Finalmente se genera el código ensamblador para la arquitectura usando la representación RTL usando un *back-end* específico de la arquitectura. El proceso se ilustra en la figura 9.

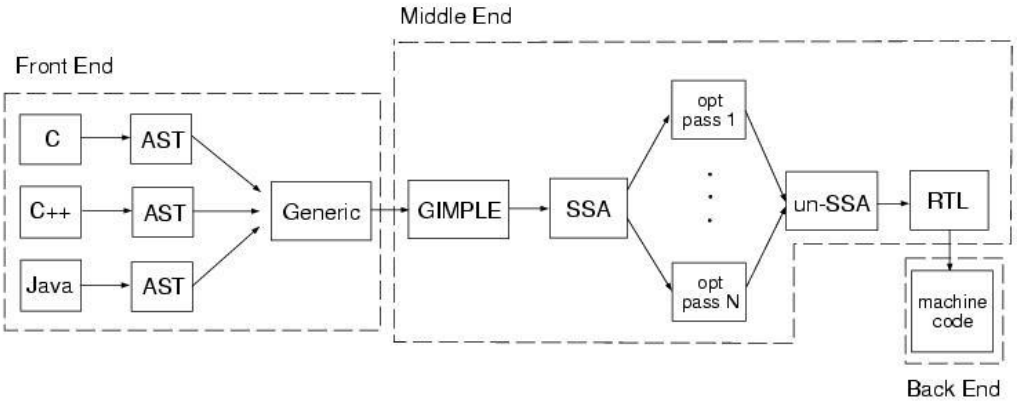

Figura 9. Diagrama de flujo del proceso de compilación usando GCC [21]

3.14 PANGO

Librería orientada al diseño y dibujo de texto, basada en el motor HarfBuzz. Facilita el uso de texto multilenguaje. Realiza renderizado de texto y posee soporte multiplataforma. Se ha incluido en numerosas APIs, entre ellas GTK+. El texto renderizado con esta librería se mostrara de forma muy similar bajo diferentes sistemas operativos. Su integración con CAIRO API ha complementado esta librería permitiendo su uso en el desarrollo de interfaces graficas de usuario GUIs.

3.15 CAIRO API

Librería orientada al dibujo de gráficos vectoriales. Provee primitivos para dibujo 2D, se encuentra escrita en C y posee numerosos *bindings* a otros lenguajes. Fue diseñada inicialmente para su uso en el sistema de ventanas X, y después fue modificada para ser multiplataforma.

Posee un modelo grafico basado en sustantivos y verbos, de forma análoga a una oración, los verbos definen acciones u operaciones en los sustantivos (tabla 2).

Sustantivos	Verbos
Destino Fuente Mascara Ruta Contexto	Trazar Rellenar Mostrar texto Pintar Enmascarar

Tabla 2. Sustantivos y verbos de Cairo API

Las rutas son uno de los elementos base de cairo. Siempre hay una ruta activa que contiene una o más coordenadas en el área de dibujo (puntos).

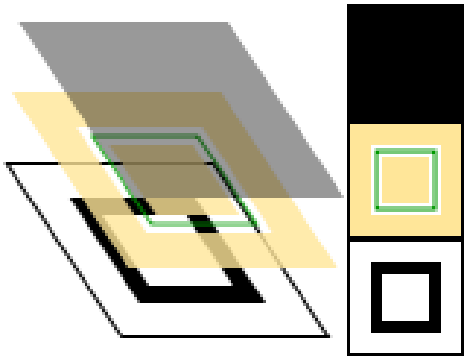

Figura 10. Ejemplo del modelo grafico de Cairo API[22]

En la figura 10, se traza una ruta en una máscara, que luego se rellena con una capa superior hacia una capa destino.

3.16 CROSS-COMPILING

Compilador con capacidad de crear código ejecutable para una plataforma diferente a la cual en que corre el compilador. Se usa para compilar hacia una arquitectura donde no es práctico compilar, generalmente por motivos de rendimiento. Un ejemplo de ellos son los microcontroladores, los cuales no pueden realizar el proceso de compilación debido a que carecen de sistema operativo.

Sus usos son los siguientes:

- Sistemas embebidos: Poseen recursos limitados, y por ello pueden ser incapaces de ejecutar un compilador, un sistema de archivos o un entorno de desarrollo.

- Compilar para múltiples arquitecturas: Puede usarse para que un solo entorno de compilación pueda crear ejecutables para multiples sistemas. Una compañía puede necesitar extender la compatibilidad de sus aplicaciones con diferentes sistemas operativos o diferentes versiones de los mismos.

- *Boostrapping* a una plataforma nueva: Al desarrollar software para una nueva plataforma o su emulador, es necesario compilar las herramientas básicas hacia el nuevo sistema, como el sistema operativo y el compilador nativo.

- Compilar código nativo para emuladores: Similar a la compilación para sistemas embebidos, los sistemas emulados suelen ser viejos u obsoletos, lo cual hace poco práctico compilar en el sistema emulado (bajo rendimiento y soporte).

3.17 FLUXBOX

Gestor de ventanas para el sistema de ventanas X, basado en Blackbox, y orientado a ser ultraliviano. Su interfaz de usuario solo tiene una barra de tareas, un menú pop-up accesible al hacer clic en el escritorio y soporte mínimo para iconos. Todas las configuraciones básicas de este sistema son controladas por archivos de texto, incluyendo la estructura de los menús y atajos de teclado.[23]

Su reducido consumo y rápido tiempo de carga hacen que sea popular en muchos Live CDs (Knoppix y GParted), así como de distribuciones como DSL (*Damn Small Linux*) Linux Mint y antiX, de igual forma puede integrarse en casi cualquier distribución de Linux.

Todo usuario que use al menos una vez Fluxbox, posee en su directorio *home* un directorio oculto (.fluxbox) donde se accede a todas las configuraciones del mismo.

3.18 U-BOOT

Su nombre completo es Das U-Boot, es un cargador de arranque (*bootloader*) universal, usado principalmente en dispositivos embebidos. Está disponible para varias arquitecturas entre ellas ARM y MIPS.

3.19 BOOT LOADER

Programa encargado de inicial el sistema operativo después de realizar las pruebas de arranque (POST). Su ejecución es fundamental ya que cuando el ordenador se enciende, no posee ninguna aplicación cargada en RAM, para cargar en ella el sistema operativo debe ejecutarse código (guardado en una memoria ROM) que descomprima el *kernel* y cargue el sistema operativo. Pueden coexistir uno o más *bootloaders* en el mismo sistema, en este último caso deben cargarse encadenados (*chain loading*).[24]

3.20 TTY

Terminal de texto utilizado para operar un sistema basado en UNIX. Su nombre y funcionamiento derivan del teletipo utilizado en los primeros dispositivos UNIX.

En él pueden ejecutarse aplicaciones y se visualizan las respuestas de las mismas, generalmente se muestra en combinaciones de colores de alto contraste. Fue diseñado para interactuar tanto con el núcleo del sistema como con el hardware necesario para establecer la conexión a los *mainframes* (figura 11), desde terminales sencillos, donde cada usuario tenía una cuenta (*login/password*).

Actualmente puede utilizarse como herramienta para manipular o realizar mantenimiento a equipos (de forma local o remota) sea bajo conexión a internet o protocolo serial RS232.

Puede emularse desde un sistema de ventanas o accederse directamente según la configuración del sistema.

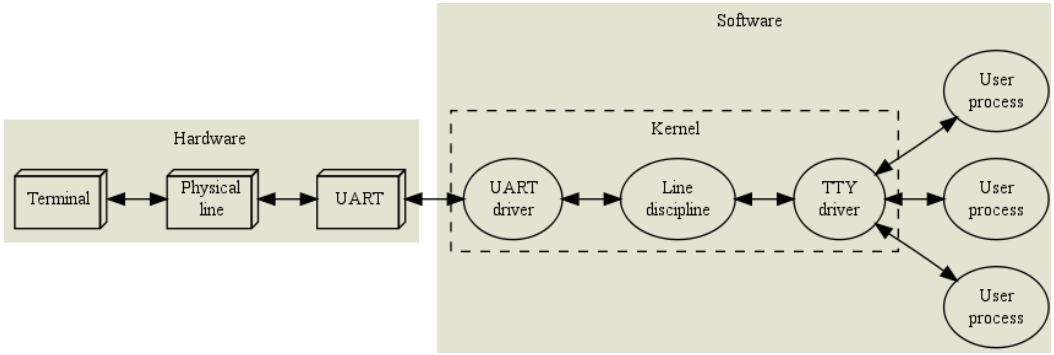

Figura 11. Interacción de TTY con hardware y software.[25]

3.21 THREAD

Secuencia de instrucciones que puede ser planificada por un sistema operativo. Esta característica permite la ejecución de varias tareas de forma concurrente, distribuyendo el tiempo de procesamiento de forma equitativa. Cada *thread* posee su contador de programa, pila de ejecución y estado de CPU. Se diferencian de los procesos por tener la capacidad de comunicarse entre sí sin necesidad de usar otros recursos.

3.22 NIOS II

Arquitectura embebida de 32 bits diseñada para la familia de FPGAs de Altera. Su set de instrucciones es RISC y es personalizable usando el SOPC Builder de Quartus IDE. Es licenciable a terceros, lo cual permite llevar un desarrollo de etapa de prototipo (FPGA) a producción en masa (ASIC).

3.23 GLADE

Diseñador de interfaces gráficas, basada en elementos de Gnome. Permite la edición de un archivo XML donde se relaciona la jerarquía de los objetos en la aplicación con sus funciones a ejecutar. Es software libre y se encuentra bajo la licencia GNU. Es desarrollado por la fundación Gnome.

3.24 ANJUTA IDE

Entorno de desarrollo integrado escrito para el proyecto Gnome. Soporta multiples lenguajes, e incluye todas las herramientas necesarias para editar tanto la parte grafica como el código de una aplicación basada en GTK. Posee ademas plugins que le añaden capacidades de debugging y permiten personalizar el perfil de compilación. Usa GCC y autotools para compilar.

4. DESARROLLO DE LA PRACTICA

4.1. CUMPLIMIENTO DE OBJETIVOS

Para comprender el enfoque de la UEN Bioingeniería, es necesario, que antes de iniciar actividades en la empresa, se conozcan sus objetivos, perfil, misión, visión y su diagnóstico actual. Con base en ello y con colaboración de los integrantes del equipo de Bioingeniería, se definen los rasgos principales del desarrollo.

Se inicia entonces identificando el dispositivo principal, la *Pandaboard*, sus capacidades, y más importante aún los dispositivos y procesos necesarios para su funcionamiento. Basado en la documentación, se define el sistema operativo que se utilizará, el lenguaje de programación en el cual se hará la aplicación y las posibles librerías graficas a utilizar. Se realizaron pruebas y se buscó documentación de dos librerías que cumplían los requerimientos del diseño.

Posteriormente, se selecciona e implementa una distribución GNU/Linux en la tarjeta de desarrollo, en búsqueda de darle funcionalidad básica (ejecutar la distribución con su configuración por defecto). Se identifica la necesidad de un gestor de ventanas y se selecciona entre varias opciones buscando la que mejor se adapte al proyecto.

Usando las librerías graficas seleccionadas previamente, se hicieron programas pequeños en busca de dominar sus características básicas y también las características necesarias para implementar la interfaz gráfica. Los programas se desarrollaron y ejecutaron en un ordenador portátil(x86) corriendo una distribución de Linux.

Se hizo uso de los periféricos básicos de la tarjeta de desarrollo, configurando elementos como el WIFI, se probaron los *jacks* de audio, y se hizo uso también del puerto DB9 que permite ver mensajes del *kernel* y puede usarse como interfaz para hacer mantenimiento al equipo sin necesidad de conectarle dispositivos adicionales. La tarjeta de desarrollo también posee *headers* de expansión con aplicaciones específicas, su funcionalidad fue documentada para su posterior uso en el desarrollo del equipo.

Muchas de las conexiones físicas de las tarjetas no se encontraban implementadas, no poseían una tarjeta interfaz para alimentarlas ni establecer comunicación, por tanto se realizaron conexiones temporales a todas las tarjetas OEM.

Luego de realizar estas tareas de configuración inicial del equipo, fue realizada la identificación del protocolo de transmisión de las tarjetas OEM, usando la documentación suministrada por cada fabricante. Individualmente se probó la comunicación con datos en bruto (cada tarjeta tiene configuraciones distintas del protocolo rs232).

A continuación se realiza la decodificación del protocolo de transmisión de las tarjetas, creando algoritmos que a su vez posibiliten su ejecución de forma

simultánea, con el objetivo de no bloquear el procesamiento de ninguna trama de datos. Adicionalmente se hizo necesario identificar los componentes fundamentales de una aplicación grafica basada en GTK+, y como estos deben interactuar con las funciones de decodificación de datos.

Finalmente se implementaron los datos decodificados en la interfaz gráfica, la cual se desarrollaba de forma paralela a los anteriores procedimientos. Dicha interfaz fue portada al sistema embebido recompilando en dicha arquitectura de forma nativa. Se solucionaron problemas menores que surgieron por la diferencia de potencia computacional de los equipos (algunos algoritmos no eran suficientemente eficientes para su ejecución en el sistema embebido).

Usando la documentación de la tarjeta de desarrollo e información recolectada de sitios dedicados al sistema operativo Linux y sus aplicaciones, se logró realizar un proceso de aprendizaje e investigación sobre sistemas embebidos en procesadores ARM, del cual se poseía poco conocimiento en la compañía, y aún más importante se usaron dichos sistemas en aplicaciones relevantes a los desarrollos del equipo de Bioingeniería (monitoreo de signos vitales). Dando a la FCV un aporte importante al conocimiento, posibilitando el uso de tecnologías punta que actualmente son la tendencia en los equipos médicos a nivel internacional.

4.1.1 TAREAS DESARROLLADAS

4.1.1.1 Identificación de la tarjeta de desarrollo *Pandaboard*

Pandaboard, es una plataforma de bajo costo y consumo, basada en tecnología móvil OMAP4. Su *system on a chip*, OMAP4460, posee un procesador ARM de rendimiento significativo (en comparación a otros sistemas de desarrollo) a una frecuencia de 1.2 GHz. Tiene una GPU *PowerVR* SGX540 a una frecuencia de 384 MHz, DSP C64x, y memoria SDRAM DDR2 de 1GiB.

Esta tarjeta de desarrollo está especialmente diseñada para ejecutar una distribución Linux portada a ARM, desde una tarjeta SD. Carece de memoria flash donde guardar un gestor de arranque (a diferencia de otros sistemas de desarrollo como *Beagleboard*), siendo todo ejecutado desde la tarjeta SD. El primer paso es realizar una prueba general de hardware, para lo cual se usa el entorno de validación de la tarjeta, basado en Amstrong Linux, disponible en la página del fabricante. Usando una distribución Linux y algunos comandos se copió dicho entorno en una tarjeta SD. Se descargó la imagen del entorno de validación y se ejecutó el comando:

sudo dd bs=4M if=validation.img of=/dev/sdb

Donde *sdb* es el directorio donde se encuentra montada la tarjeta SD (se monta de forma automática).

Para ejecutar esta prueba es necesario disponer de:

- Monitor DVI-D o HDMI
- Parlantes
- Memoria USB
- Red WIFI abierta
- Ordenador con una distribución Linux, con puerto serial o adaptador serial USB.
- Cliente serial en el ordenador host.
- Adaptador 5V

La corriente máxima del adaptador recomendada por el fabricante es de 4A. En la práctica se usaron inicialmente adaptadores de 1,2 y 3 A, donde solo el último funciono correctamente. No puede alimentarse por el puerto USB-otg ya que los ordenadores normalmente solo suministran hasta 500 mA por sus puertos USB, y de ser alimentado por baterías debe considerarse el pico de corriente al arranque (1.2 A).
Antes de encender la tarjeta de desarrollo es importante realizar todas las conexiones. Debe iniciarse el cliente serial y configurarse con los parámetros del puerto DB9 de la tarjeta *Pandaboard* (115200 *Bauds*, 8 bits, sin paridad, 1 bit de parada). Si el sistema inicia correctamente y el puerto DB9 se encuentra configurado correctamente, en el cliente RS232 debe mostrarse el arranque de *U-boot* (figura 12).

```
U-Boot SPL 2011.12-rc1
Texas Instruments OMAP4460 ES1.1

CPU  : OMAP4460 ES1.1
Board: OMAP4 Panda
I2C:   ready
DRAM:  1 GiB
WARNING: Caches not enabled
MMC:   OMAP SD/MMC: 0
Using default environment

In:    serial
Out:   serial
Err:   serial
Net:   No ethernet found.
Hit any key to stop autoboot:  0
```

Figura 12. Arranque de U-boot

En el monitor conectado a la tarjeta de desarrollo, después del arranque se observara la secuencia de colores de la figura 13.

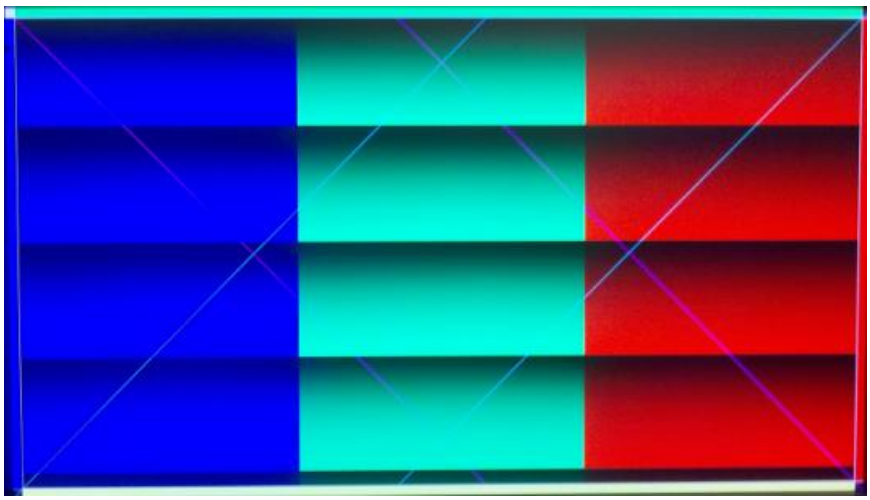

Figura 13. Secuencia de colores para prueba de video

Luego de lo cual se emitirán sonidos por la salida de audio, los LEDs de la tarjeta parpadearan uno a uno, se hará prueba de lectura/escritura a la memoria USB y finalmente se proporcionara una línea de comandos para ejecutar pruebas posteriores, como la prueba WIFI usando el script wlan-test.sh. Durante todo el proceso se ha mostrado el estado de las pruebas en el puerto serial.

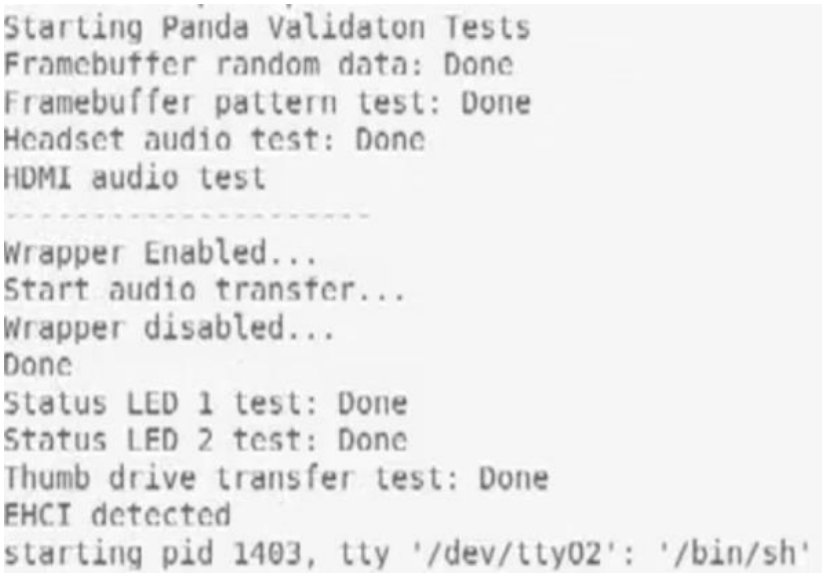

Figura 14. Estados de la verificación del sistema

Comprobado el correcto funcionamiento de todos sus elementos (figura 14), e identificadas sus características, se procede entonces con la implementación del sistema operativo.

4.1.1.2 Implementación de una distribución Linux en la tarjeta de desarrollo

Desde la página oficial de proyecto, se revisaron las diferentes distribuciones disponibles para la tarjeta de desarrollo: MinimalFS, Android y Ubuntu. Así mismo se revisaron otros proyectos como Linaro (que no es una distribución en si sino revisiones específicas de otras distribuciones para permitir su ejecución en la arquitectura ARM. Sus características se resumen en la tabla 3.

Distribución	Ventajas	Desventajas
Angstrom	• Basada en Debian • Orientada a dispositivos embebidos • Minimalista por defecto • Escalable a pequeños y grandes dispositivos • Posee un generador online de imágenes de instalación personalizable	• Carece de aplicaciones básicas para ejecución de GUIs • No posee casi documentación • Poco extendida • Es un proyecto joven en comparación a otras distribuciones
Android	• Extenso soporte • Orientada a dispositivos móviles • Alta compatibilidad con aplicaciones • Respaldada por Google • Aplicaciones base preinstaladas • Posee entorno de desarrollo propio	• Al ser orientada a comunicaciones, es vulnerable y no aconsejable para un equipo medico • Debido a su popularidad en dispositivos móviles, existe gran cantidad de malware para esta distribución • No es totalmente abierta
Ubuntu	• Extensa documentación oficial y extraoficial • Alta compatibilidad con aplicaciones • Derivada de Debian • Respalda por Canonical • Aplicaciones base preinstaladas • Drivers preinstalados	• Es la más conocida de las distribuciones para PCs, haciéndola vulnerable a malware • Las últimas versiones poseen entornos gráficos inadecuados para sistemas embebidos (costosos en recursos)
Linaro Ubuntu (Retoma las características de Ubuntu)	• Ajustada a las diferentes plataformas por ingenieros de las empresas que producen las plataformas	• Al centrarse en mejorar la compatibilidad de la plataforma, se olvidan detalles como módulos para interactuar con algunos dispositivos USB comunes y elementos del SoC.

Tabla 3. Comparación de distribuciones disponibles para la *Pandaboard*

Inicialmente fue seleccionada Linaro Ubuntu, la cual se ejecutó correctamente en el sistema, en etapa de desarrollo se hizo evidente su desventaja, con su incompatibilidad con los adaptadores rs232-usb basados en chips FTDI y con los drivers de la GPU incluida en el SoC, POWERVR SGX 540.

Se descargó una imagen preinstalada de Ubuntu Linux para ARM compilado por el proyecto Linaro. El tamaño de la memoria SD a donde se quemara la imagen debe ser igual o mayor al tamaño del sistema de archivos de la imagen, por ello se hizo compra de una memoria SD de 8GB. Se ejecutó el siguiente comando desde un ordenador con Ubuntu Linux con la memoria montada en /dev/sdb:

pv -tpreb ./imagen.img | dd bs=4M of=/dev/sdb

Debido a que el proceso puede tomar de 20 minutos a 1 hora, se ejecutan estos dos comandos encadenados para poder monitorear el progreso. Esta versión preinstalada de Ubuntu posee soporte extendido (LTS) por parte de canonical, lo cual la hace perfecta para su uso en desarrollos embebidos.

Posteriormente se inició sesión en el sistema operativo por primera vez. Se hizo notoria la incapacidad del sistema de correr adecuadamente la interfaz de usuario por defecto de la distribución, Unity. Se busca información acerca de ello en internet y se determinan dos posibles causas de la baja en rendimiento: La interfaz es costosa en recursos o el controlador de la GPU necesita ser instalado. Con el objetivo de avanzar en el desarrollo se postergan estas tareas y se sigue con la identificación y uso de los periféricos básicos de la tarjeta de desarrollo. En la etapa final del proyecto se hace necesario cambiar de la distribución compilada por Linaro a la distribución oficial de canonical, debido a una de sus desventajas: no era compatible con el adaptador rs232-usb ni con el driver de la GPU del SoC.

4.1.1.3 Identificación y uso de periféricos básicos de la tarjeta de desarrollo *Pandaboard*

Con Ubuntu Linux en ejecución, se probaron los dispositivos básicos de la tarjeta de desarrollo, con las siguientes tareas:

- Instalación los drivers del módulo WIFI desde los repositorios de Linaro

  Con la ayuda del centro de software de Ubuntu se seleccionaron e instalaron los drivers del módulo WIFI de la *Pandaboard*. Fue necesario incluir algunos repositorios en la lista para permitir dicha instalación. Se editó el archivo */etc/apt/sources.list* y se agregó la siguiente línea al final del archivo:

  ppa:linaro-maintainers/wireless-tools

  Posteriormente se reinició el sistema para que cargara de nuevo los repositorios, y se instaló el driver del módulo WIFI sin mayores inconvenientes.

- Reproducción de sonido usando los *jacks* de audio

El driver de sonido estándar ALSA (*Advanced Linux Sound Architecture*) venía preinstalado y totalmente funcional. Se probó reproduciendo sonidos desde el explorador satisfactoriamente.

- Prueba de conectividad a internet por cable Ethernet

  Se realizó la conexión a internet usando el puerto Ethernet. Fue detectado automáticamente sin inconvenientes.

- Prueba de conectividad del puerto DB9

  Se conectó un adaptador RS232-USB entre el puerto DB9 de la *Pandaboard* y un puerto USB del ordenador *host* bajo Ubuntu. Después de inspeccionar la documentación en busca de los parámetros de comunicación que usa el puerto (115200 *bauds*, 8 bits, sin paridad y 1 bit de parada), se estableció comunicación y se obtuvo una consola como *root*, desde donde se tiene acceso a todo el sistema, haciendo de este protocolo ideal para realizar mantenimiento al equipo.

- Control de los LEDs de la tarjeta

  Los LEDs en la tarjeta de desarrollo se denominan STATUS1 y STATUS2. La función original de STATUS1 es comportarse como “latido” del núcleo del sistema, es decir, parpadea a intervalos periódicos dos veces por segundo con el objetivo de informar al usuario que el núcleo se está ejecutando, mientras que STATUS2 informa que se lleva a cabo una operación de lectura en la tarjeta SD. Las funciones preasignadas de los LEDs son más útiles que cualquier otra función que se les pueda asignar posteriormente.

  Para manipular a voluntad los LEDs de la tarjeta es necesario acceder a los archivos del sistema asociados a su configuración. Estos se encuentran en */sys/class/leds*. Inspeccionando este directorio se muestran los archivos: Pandaboard:status1 y Pandaboard:status2.

Ejecutando los siguientes comandos se determinó los posibles *triggers* para cada LED. Los *triggers* son utilizados para relacionar un LED con un evento del *kernel*.

cat /sys/class/leds/pandaboard::status1/trigger
cat /sys/class/leds/pandaboard::status2/trigger

En la tabla 4 se muestran los *triggers* que pueden asignarse a cada LED.

Trigger	**Descripción**
None	Desactiva los triggers
Nand-disk	Parpadea cada vez que se accese la memoria *nand*
Mmc0	Parpadea cada vez que se accese la SD0
Mmc1	Parpadea cada vez que se accese la SD1
Timer	Parpadea a intervalos periódicos establecidos en ms
[heartbeat]	Parpadea periódicamente al ejecutarse el *kernel*
Backlight	Da el control del LED al *framebuffer* del sistema
Gpio	Asigna el estado del LED a un GPIO
Default-on	Establece el estado inicial del LED.

Tabla 4. *Triggers* para *status1* y *status2*

Para manipular los LEDs deben desactivarse primero los triggers, asignándoles *none*. Después de ello solo resta controlar su estado, editando la propiedad *brightness* con el estado deseado, siendo 1 encendido y 0 apagado.

echo 1 > /sys/class/leds/pandaboard::status1/brightness

El proceso de configuración del trigger y edición del estado del LED fue automatizado mediante un script de Python que editaba los archivos correspondientes a cada función.

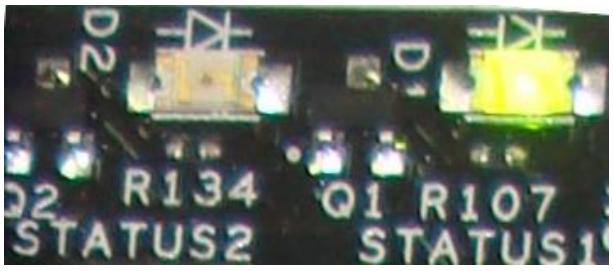

Figura 15. LEDs status de la tarjeta de desarrollo

4.1.1.4 Documentación e implementación de los puertos de expansión de la tarjeta de desarrollo *Pandaboard*

La tarjeta de desarrollo Pandaboard posee dos puertos de expansión llamados A y B (figura 16), asignados a múltiples funciones controladas por un multiplexor programable del SoC. Los conectores tienen 28 pines de 0.1'' *through-hole*, los conectores no vienen montados de fábrica, esto se deja al usuario del sistema de desarrollo, con el propósito de que tenga la libertad de elegir que conector se ajusta mejor a su aplicación. Las funciones de cada pin pueden variar según el multiplexado del OMAP4460, generalmente cada pin tiene dos o más posibles funciones, resumidas en las figuras 17 y 18.

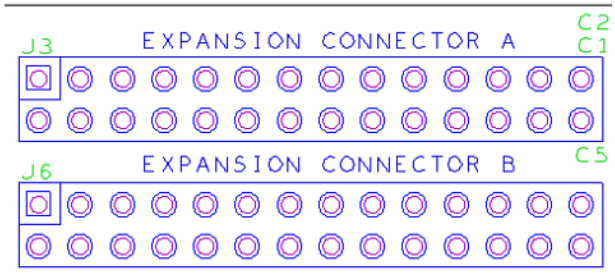

Figura 16. Puertos de expansión [11]

- Puerto de expansión A

J3 Pin #	OMAP Ball #	Primary Function	Secondary Function	Description of Pandaboard ES Usage
1	---	VIO_1V8	---	1.8V I/O Power
2	---	DCIN_JACK	---	5Vdc Input Power
3	B16	GPMC_AD7	SDMMC2_DAT7	GPMC Address/Data Bit 7
4	AH23	MCSPI1_CS3	GPIO_140	SPI1 Chip Select 3 (also UART1_RTS)
5	A16	GPMC_AD6	SDMMC2_DAT6	GPMC Address/Data Bit 6
6	AH19	UART4_TX	GPIO_156	UART4 Transmit Data
7	D15	GPMC_AD5	SDMMC2_DAT5	GPMC Address/Data Bit 5
8	AG20	UART4_RX	GPIO_155	UART4 Receive Data
9	C15	GPMC_AD4	SDMMC2_DAT4	GPMC Address/Data Bit 4
10	AF23	MCSPI1_CS1	GPIO_138	SPI1 Chip Select 1 (also UART1_RX)
11	D13	GPMC_AD3	SDMMC2_DAT3	GPMC Address/Data Bit 3
12	AG22	MCSPI1_SIMO	GPIO_136	SPI1 Slave In Master Out
13	C13	GPMC_AD2	SDMMC2_DAT2	GPMC Address/Data Bit 2
14	AG23	MCSPI1_CS2	GPIO_139	SPI1 Chip Select 2 (also UART1_CTS)
15	D12	GPMC_AD1	SDMMC2_DAT1	GPMC Address/Data Bit 1
16	AE23	MCSPI1_CS0	GPIO_137	SPI1 Chip Select 0
17	C12	GPMC_AD0	SDMMC2_DAT0	GPMC Address/Data Bit 0
18	AE22	MCSPI1_SOMI	GPIO_135	SPI1 Slave Out Master In
19	B12	GPMC_NWE	SDMMC2_CMD	GPMC Write Enable
20	AF22	MCSPI1_CLK	GPIO_134	SPI1 Clock Out
21	B11	GPMC_NOE	SDMMC2_CLK	GPMC Output Enable
22	D19	GPMC_AD15	GPIO_39	GPMC Address/Data Bit 15
23	AH22	I2C4_SDA	GPIO_133	I2C4 Serial Data
24	AG21	I2C4_SCL	GPIO_132	I2C4 Serial Clock
25	---	REGEN1	---	TWL6030 REGEN1
26	E7	SYS_NRESPWRON	---	Power On Reset
27	---	DGND	---	Digital Ground
28	---	DGND	---	Digital Ground

Figura 17. Funciones del puerto de expansión A[11]

- **Puerto de expansión B**

J6 Pin #	OMAP Ball #	Primary Function	Secondary Function	Description of Pandaboard ES Usage
1	---	VBUS_3	---	VBUS out from USB Host Port #3
2	---	VBUS_4	---	VBUS out from USB Host Port #4
3	---	USBH3_DM	---	USB Host Port #3 Data Minus
4	---	USBH4_DM	---	USB Host Port #4 Data Minus
5	---	USBH3_DP	---	USB Host Port #3 Data Plus
6	---	USBH4_DP	---	USB Host Port #4 Data Plus
7	---	DGND	---	Digital Ground
8	---	DGND	---	Digital Ground
9	C19	GPMC_AD14	GPIO_38	GPMC Address/Data Bit 14
10	D18	GPMC_AD13	GPIO_37	GPMC Address/Data Bit 13
11	AF7	SYS_NRESWARM	---	Warm Reset
12	---	PB_POWER_ON	---	Power on input to TWL6030 (ref. to VBAT)
13	---	HFL_P	---	Hands Free Left Speaker Out (+)
14	AG24	H_DMTIMER11_PWM	GPIO_121	Display PWM Control
15	---	HFL_N	---	Hands Free Left Speaker Out (-)
16	---	VDD_VAUX1	---	VAUX1 from TWL6030
17	C18	GPMC_AD12	GPIO_36	GPMC Address/Data Bit 13
18	C16	GPMC_AD8	GPIO_32	GPMC Address/Data Bit 8
19	B26	GPMC_WAIT0	GPIO_61	GPMC Wait input 0
20	D16	GPMC_AD9	GPIO_33	GPMC Address/Data Bit 9
21	C25	GPMC_NWP	GPIO_54	GPMC Write Protect
22	C17	GPMC_AD10	GPIO_34	GPMC Address/Data Bit 10
23	B22	GPMC_CLK	GPIO_55	GPMC Clock Out
24	D17	GPMC_AD11	GPIO_35	GPMC Address/Data Bit 11
25	B25	GPMC_NCS0	GPIO_50	GPMC Chip Select 0
26	D25	GPMC_NADV_ALE	GPIO_56	GPMC Address Valid/Address Latch Enable
27	C21	GPMC_NCS1	GPIO_51	GPMC Chip Select 1
28	C23	GPMC_NBE0_CLE	GPIO_59	GPMC Byte Enable 0/Command Latch Enable

Figura 18. Funciones del puerto de expansión B[11]

Para usar cualquiera de los pines disponibles con una de sus funciones específicas es necesario primero revisar la función que se encuentra realizando actualmente (es posible que se encuentre en el estado que nosotros requerimos y por tanto no necesitemos cambiarla). Para ello consultamos en las tablas de asignaciones el nombre oficial del pin (que corresponde a la función primaria). El ejemplo se realizara con una de las funciones más sencillas disponibles, un GPIO (específicamente el GPIO_138, función primaria mcspi1_cs1, pin 10 del puerto de expansión A), sin embargo es aplicable a cualquier otra funcionalidad disponible.

Se ejecuta el siguiente comando

cat /sys/kernel/debug/omap_mux/mcspi1_cs1

que regresara algo similar a:

name: mcspi1_cs1.gpio_138 (0x4a10013a/0x13a = 0x010b), b af23, t NA
mode: OMAP_PIN_INPUT_PULLDOWN | OMAP_MUX_MODE3
signals: mcspi1_cs1 | uart1_rx | NA | gpio_138 | NA | NA | NA | safe_mode

La línea *mode* describe la configuración actual y la línea *signals*, los posibles modos de funcionamiento del multiplexor, como se muestran en la tabla 5.

Modo	**Función**
OMAP_MUX_MODE0	Mcspi1_cs1
OMAP_MUX_MODE1	Uart1_rx
OMAP_MUX_MODE2	NA
OMAP_MUX_MODE3	Gpio_138
OMAP_MUX_MODE4	NA
OMAP_MUX_MODE5	NA
OMAP_MUX_MODE6	NA
OMAP_MUX_MODE7	*Safe_mode* (aísla el pin)

Tabla 5. Modos del multiplexor OMAP y sus funciones.

El otro modificador en *mode*, informa que se tiene activada una resistencia de *pulldown* (útil para su función de GPIO, pero no recomendada para otras como uart). Otros flags de configuración pueden encontrarse en el archivo mux.h, usado por el sistema instalado en la tarjeta de desarrollo *Pandaboard*.

Para modificar la funcionalidad, se debe acceder al siguiente archivo:

/sys/kernel/debug/omap_mux/mcspi1_cs1

Y escribir dos bytes en hexadecimal que reflejen la configuración deseada. La estructura de estos se muestra en la figura 19.

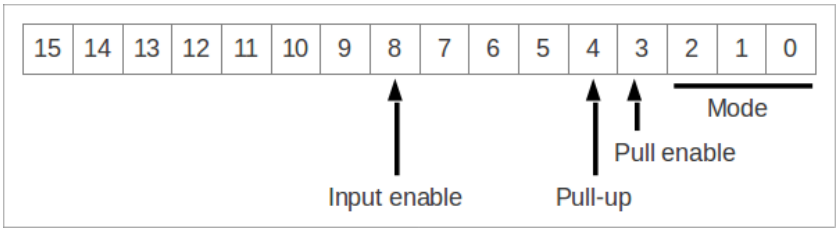

Figura 19. Palabra de configuración del multiplexor[26]

En este caso asignaremos la funcionalidad GPIO, pero cambiaremos el *pull-up* a *pull-down*. La palabra de configuración quedaría así:

0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1
Configuracion de *sleep mode*							*Input*	Sin uso			PD	PE	MODO GPIO		

La configuración del *sleep mode* se deja idéntica a su estado de fábrica (0x00). La palabra generada entonces corresponde a 0x010B.Se edita entonces el archivo con el nuevo valor de configuración:

echo 0x10b > /sys/kernel/debug/omap_mux/mcspi1_cs1

y se comprueba que haya cambiado satisfactoriamente

cat /sys/kernel/debug/omap_mux/mcspi1_cs1

debe regresar algo similar a:

name: mcspi1_cs1.gpio_138 (0x4a10013a/0x13a = 0x011b), b af23, t NA
mode: OMAP_PIN_INPUT_PULLUP | OMAP_MUX_MODE3
signals: mcspi1_cs1 | uart1_rx | NA | gpio_138 | NA | NA | NA | safe_mode

Se ha cambiado satisfactoriamente el *pull-down* a *pull-up*.

Con el procedimiento de cambio del multiplexor en cada pin, pueden activarse los siguientes dispositivos que usan uno o más pines:

- **Puerto de expansión A:**

GPMC: *General-Purpose Memory Controller*
Controlador OMAP unificado que sirve de interface con dispositivos de memoria externa como:
- Memorias SRAM asíncronas y dispositivos ASIC
- Modo página asíncrono y *burst* síncrono de NOR flash
- NAND flash

SDMMC: Controlador de tarjetas SD y MMC

MCSPI: *Multi Channel Serial Port Interface*. Permite realizar conexiones dúplex síncronas entre un local host y dispositivos SPI externos.

GPIO: *General-Purpose Input Output*. Maneja pines de E/S

UART: *Universal Asyncronous Receiver Transmitter*. Solo hay disponible un puerto UART(rx/tx) para propósito general , el UART4. Los demás son usados por la tarjeta para el funcionamiento normal de la misma.

I2C: Controlador de dispositivos con protocolo *Inter Integrated Circuit*

SYS_NRESPWRON: *Reset* global en frio. Apaga el sistema desconectando la energía

REGEN Salida del dispositivo conectada a circuitos integrados alimentados por el mismo. Es útil para establecer la secuencia de encendido ya que suministra energía antes de VIO.

- **Puerto de expansión B**

USB: Controlador de puerto USB. Permite establecer conexiones estáticas USB sin necesidad de puerto, sino con las señales conectadas directamente, permitiendo un ahorro de espacio significativo

GPMC: *General-Purpose Memory Controller*
Controlador OMAP unificado que sirve de interface con dispositivos de memoria externa como:

- Memorias SRAM asíncronas y dispositivos ASIC
- Modo pagina asíncrono y *burst* síncrono de NOR flash
- NAND flash

SYS_NRESWARM: Reset global en caliente. Reinicia el sistema sin desconectar la energía.

HANDSFREE STEREO OUT: Salida de audio stereo
GPIO: General-Purpose Input Output. Maneja pines de E/S

4.1.1.5 Modificaciones varias a la distribución Linux

Se inicia entonces el proceso de modificación de la distribución instalada (Ubuntu) con el objetivo de ajustarla a la aplicación final. El primer aspecto a modificar es el sistema de ventanas. Unity es un gestor de ventanas visualmente complejo, ideal para un usuario de escritorio pero no para una aplicación embebida.

Unity

Figura 20. Entorno de escritorio Unity

Fluxbox

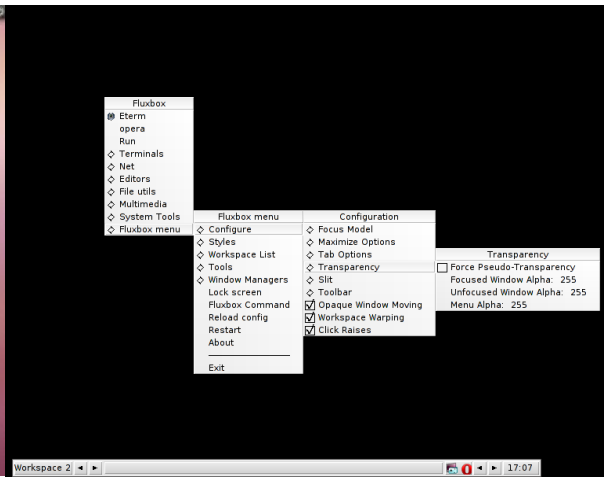

Figura 21. Entorno de escritorio Fluxbox

En contraste, Fluxbox es minimalista, solo posee una barra de tareas y un menú desplegable, accesible haciendo clic derecho en cualquier parte del escritorio, los cuales pueden ser desactivados para dejar en ejecución exclusivamente la aplicación.

Se decide entonces instalar Fluxbox para comparar su desempeño con el de Unity. Afortunadamente se encuentra disponible desde los repositorios de Ubuntu, por ello solo basta con ejecutar el siguiente comando, desde el terminal:

sudo apt-get install fluxbox

Luego de la instalación de Fluxbox, se seleccionó como gestor de ventanas y se inició sesión. El rendimiento del sistema mejoro significativamente, probando así que realizar dicho cambio era necesario, además, el minimalismo de Fluxbox hace menos probable que un usuario malintencionado tenga acceso a permisos de administración, característica importante en un sistema embebido.

A diferencia de Gnome, Fluxbox no posee casi herramientas para su configuración, ya que todo se realiza desde consola o editando archivos de texto. Para establecer la conexión a internet por WIFI se propusieron dos métodos:

- Editando archivos de configuración:
  Se establecieron permisos de lectura y escritura al archivo de interfaces:

chmod 0600 /etc/network/interfaces

Edición del archivo de interfaces

xedit /etc/network/interfaces

Definición de parámetros de conexión , SSID y PSK:

auto wlan0
iface wlan0 inet dhcp
wpa-ssid nombredelared
wpa-psk passworddelared

Se activa entonces la interfaz

ifup wlan0

Finalizando la edición del archivo se colocan configuraciones adicionales, por ejemplo la de un *proxy*, desde el terminal:

export http_proxy='http://proxy:puerto/

O remoción del proxy

export http_proxy=""

- Usando una aplicación que lo gestione gráficamente:
  La aplicación Nm-applet, parte del entorno Gnome, permite configurar los parámetros de la red de forma sencilla (figura 22), sin edición de archivos por parte del usuario.

Figura 22. Opciones de Nm-applet

Ambos métodos son útiles. En el caso de la etapa de desarrollo donde la importancia solo reside en realizar la conexión a internet, es ventajoso hacerlo gráficamente, sin embargo para un producto final la edición de archivos de configuración directa desde la interfaz gráfica es la manera ideal de establecer esta conexión.

Es necesario también reconfigurar el teclado para que se ajuste al idioma y distribución de las teclas del dispositivo conectado. Esto se puede lograr usando un asistente llamado con el siguiente comando:

sudo dpkg-reconfigure keyboard-configuration

Para mejorar su seguridad es aconsejable desactivar los terminales sobrantes. Estos terminales vienen por defecto activados debido a que en sus inicios el

sistema operativo GNU/Linux venia diseñado para ejecutarse en mainframes y para ser accedido desde terminales (esta tarea aun es común hoy en día). Dichos terminales pasaron a ser locales a los ordenadores de escritorio con la ventaja de que interactúan de forma más eficiente que un terminal emulado, que se lanza desde el gestor de ventanas. La combinación de teclas estándar para acceder a estos terminales son control + alt + f[1-6], para tty[1-6] (en teclados PC).
El siguiente procedimiento se usa para desactivar estos elementos sobrantes:

Obtener permisos de *root*

sudo su (el comando pregunta la contraseña de administrador)

Con estos permisos , ir a /etc/init y editar los archivos tty[1-6].conf

xedit /etc/init/tty[1-6].conf

Comentar todas las líneas precediéndolas del signo #.

Debe tenerse precaución de no desactivar alguna consola importante para el sistema embebido. Para ello asegurarse de conocer el nombre del TTY que corre en el puerto DB9 (*debugging*).

Ahora debe editarse la lista de consolas activas, para ello debe editarse /etc/default/console-setup. En la línea que dice ACTIVE_CONSOLES establecer los TTY activos:
ACTIVE_CONSOLES="/dev/tty[1-2]"
(activa consolas de la 1 a la 2)

Deben restringirse permisos, por tanto se colocan contraseñas a una cuenta administradora y a *root* (en caso de estar activada. Si esta desactivada es mejor dejarla así es más seguro).

El gestor de inicio de sesión por defecto de Ubuntu Linux se llama LightDM. Siendo su aplicación final no es necesario que se muestre una pantalla de inicio de sesión, es mas no es aconsejable. Por ello se hace necesario desactivar la aparición de dicho gestor. Se probaron dos alternativas para dicho procedimiento, evitar directamente la ejecución de LightDM o ejecutar LightDM configurándolo para iniciar automáticamente.

- Evitando la ejecución de LightDM:

Se crea el archivo lightdm.override en /etc/init/

xedit /etc/init/lightdm.override

Se escribe en el archivo
manual

Con ello LightDM no iniciara y en cambio lo hará una consola TTY. El número de esta consola debe identificarse para configurarla y permitirle hacer *autologin*. Para ello debe editarse el archivo ttyx.conf (siendo x el número de la consola).

xedit /etc/init/ttyx.conf

Y reemplazar la última línea por:

exec /bin/login –f usuario < /dev/ttyx > /dev/ttyx x>&1

Después de hacer *login*, la consola ejecuta un script dependiendo del usuario. Esto es útil para iniciar aplicaciones. Para ello es necesario editar el archivo:
/home/usuario/.bash_login

y en el escribir los comandos a ejecutar. En este caso lo que se necesita es ejecutar el servidor gráfico, para ello colocamos en la última línea:

startx

De no existir el archivo, es necesario crearlo.

Posteriormente solo queda iniciar el gestor de ventanas, Fluxbox y la aplicación. Para ello editamos el script ejecutado después de iniciar el servidor gráfico:

/home/usuario/.xinitrc

Y al final del mismo:

exec startfluxbox &
sleep 1
{
aplicación &
}

- Haciendo *autologin* desde LightDM

Para ello es necesario editar el archivo *lightdm.conf*:
xedit /etc/lightdm/lightdm.conf

al final del archivo añadir lo siguiente
[SeatDefaults]
autologin-user=usuario
autologin-user-timeout=0
user-session=fluxbox
greeter-session=unity-greeter

Activando así Fluxbox directamente sin necesidad del gestor de inicio de sesión. El procedimiento para iniciar automáticamente la aplicación es similar al descrito anteriormente: se edita *xinitrc*, con un *delay* un poco mayor con el objetivo de que Fluxbox alcance a arrancar antes de la ejecución de la aplicación.

4.1.1.6 Implementación del puerto serial en lenguaje C bajo Linux

A diferencia de Windows, Linux carece de una librería específica para uso del puerto serial, por compatibilidad todas las operaciones seriales del sistema se han realizado desde sus inicios usando la interfaz de terminal POSIX (desde impresoras, hasta conexiones de datos). Debido a ello la inicialización y configuración de un puerto serial bajo Linux requiere de extensa configuración y comprensión del funcionamiento de esta API.

El elemento principal usado para su configuración es la estructura Termios, a la que se asocia un puerto (representado por un archivo), y que contiene la configuración del mismo. La estructura termios contiene campos, que se configuran mediante *flags*, incluidas también en la librería.

La gran ventaja de esta API es que permite configurar manualmente algunos parámetros que normalmente son establecidos por el sistema operativo como el número de caracteres leídos por llamada, el *timeout* del puerto, y el uso de caracteres especiales para funcionar como terminal.

Se muestra la estructura Termios a continuación, con la función de cada uno de sus campos:

```
struct termios {
   tcflag_t c_iflag ;  // Input modes
   tcflag_t c_oflag ;  // Output modes
   tcflag_t c_cflag ;  // Control modes
   tcflag_t c_lflag ;  // Local modes
   cc_t c_cc[NCCS] ;   // Control characters
} ;
```

Y un resumen de sus funciones, con su sintaxis:

- *int tcgetattr (int fd, struct termios *termios_p);*
- *int tcsetattr (int fd, int optional_actions, struct termios *termios_p);*
- *int tcsendbreak (int fd, int duration);*
- *int tcdrain (int fd);*
- *int tcflush (int fd, int queue_selector);*
- *int tcflow (int fd, int action);*
- *int cfmakeraw (struct termios *termios_p);*
- *speed_t cfgetospeed (struct termios *termios_p);*
- *int cfsetospeed (struct termios *termios_p, speed_t speed);*
- *speed_t cfgetispeed (struct termios *termios_p);*
- *int cfsetispeed (struct termios *termios_p, speed_t speed);*
- *pid_t tcgetpgrp (int fd);*
- *int tcsetpgrp (int fd, pid_t pgrpid);*

4.1.1.7 Conexión y pruebas de comunicación básicas a las tarjetas OEM

Finalmente, se realiza la conexión a las tarjetas de signos vitales OEM. Debido a que muchas de ellas no tenían cables de alimentación, fue necesario hacer unos temporalmente que coincidieran con los conectores de cada una de las tarjetas. Primero se probaron una a una por separado en un ordenador portátil, a donde se conectaron usando un adaptador rs232-USB. Se usaron los parámetros de comunicación especificados en la documentación de cada una de las tarjetas, para su verificación. Algunas tarjetas enviaban las tramas de datos automáticamente mientras que otras solo respondían a solicitudes. Su configuración se resume en la tabla 6.

Tarjeta OEM	**Velocidad (bauds)**	**Configuración**			**Comando de prueba (hex)**
		Bits	Par	Stop	
Mindray	115200	8	E	1	Automático
Suntech	9600	8	N	1	3A 79 03 00 4A
Biomedian	115200	8	N	1	Automático
Goldwei	38400	8	E	1	Automático
Nell	19200	8	N	1	Automático

Tabla 6. Configuración de comunicación de las tarjetas OEM.

En la tabla 7 se muestran los esquemas de conexión utilizados para cada una de las tarjetas.

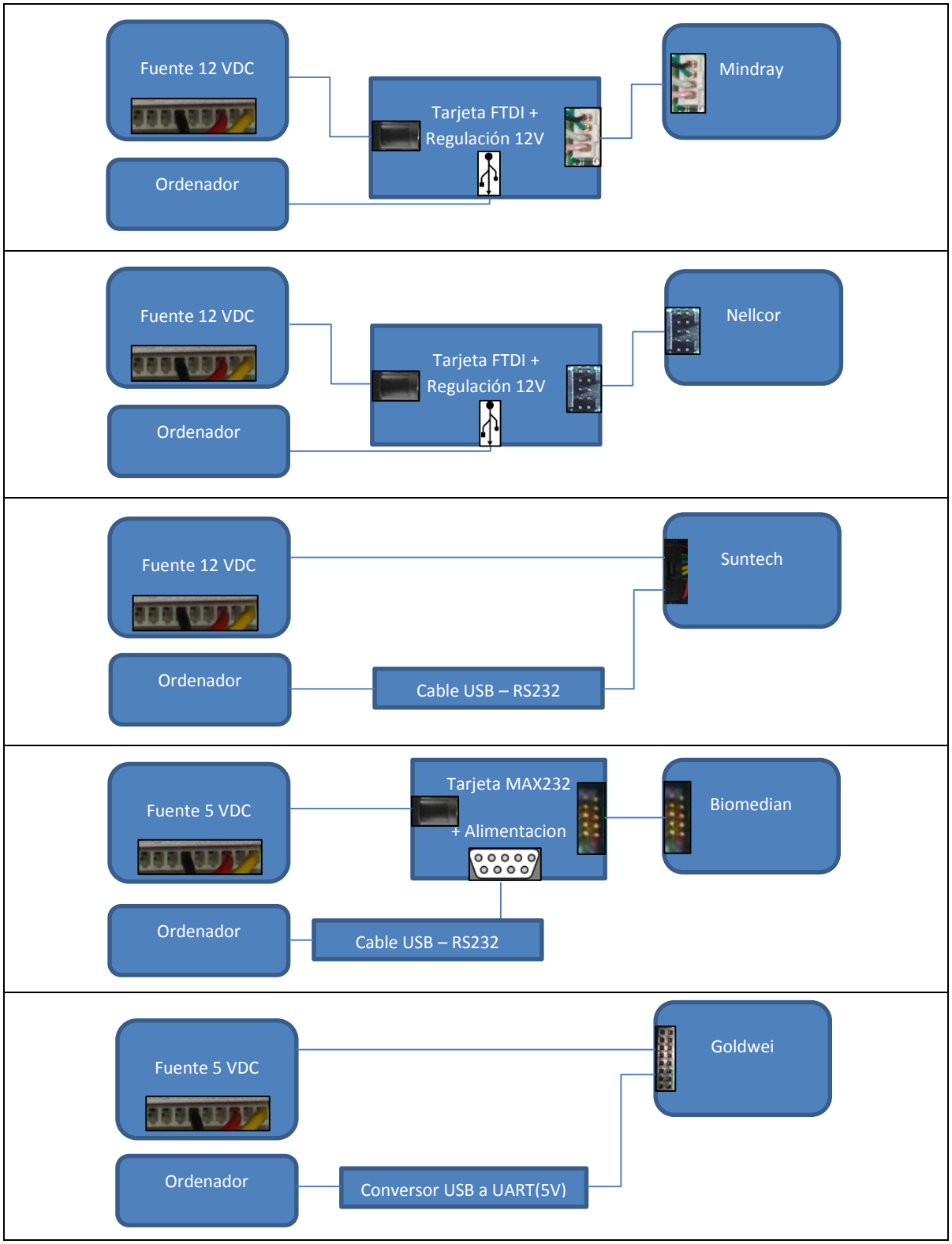

Tabla 7. Esquemas de conexión de las tarjetas OEM.

4.1.1.8 Identificación del protocolo de transmisión de las tarjetas OEM

A pesar de que usan protocolos físicos iguales (UART), cada tarjeta tiene una forma distinta de distribuir la información en la trama de datos. Antes de iniciar el diseño del algoritmo de decodificación es necesario estudiar la estructura de la trama de datos de cada tarjeta. Básicamente todos los protocolos poseen 3 elementos: ID, datos y comprobación de errores.

- Mindray:

	HEAD	**DATA1**	**DATA2**	**DATA3**	**DATA4**	**DATA5**	**DATA6**	**DATA7**	
ID	N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	CKSUM
Tamaño del paquete = 1+N+1									

Tabla 8. Estructura del paquete de la tarjeta Mindray.

El paquete se compone de 1+N+1 bytes (tabla 8), donde N varía según el tipo de comando. ID identifica el tipo de comando. Los bytes de datos se diferencian de los demás en su bit más significativo, que se reemplaza por 1. Es necesario reconstruir cada byte de datos, con su bit más significativo original. Este se encuentra en el byte head, como se muestra en la tabla 9.

1	**MSBD7**	**MSBD6**	**MSBD5**	**MSBD4**	**MSBD3**	**MSBD2**	**MSBD1**
MSB							LSB

Tabla 9. Estructura del byte head.

El ID se encuentra en la documentación del equipo, y cada uno tiene una distribución diferente para los datos, entre los bytes DATA. La comprobación de errores se hace mediante *checksum*, procedimiento en el cual se suman todos los bytes del paquete y se descartan los bits mayores a cierta cantidad.

Checksum = (ID + N(1) + N(2) + N(3) + N(4) + N(5) + N(6) + N(7) + N(8)) & 127

El *checksum* calculado se compara con el *checksum* recibido, si coinciden entonces el paquete es válido, de lo contrario es descartado. Esta característica es muy importante sobre todo en equipos médicos ya que un dato erróneo puede llevar a un mal diagnóstico.

Para transmitir paquetes desde el *host* hacia la tarjeta se usa el mismo protocolo, es muy importante calcular correctamente el *checksum* ya que de lo contrario la tarjeta descartara el paquete.

- Nellcor

El protocolo de comunicación de esta tarjeta es tal vez el más complejo de todos. Posee dos capas lógicas de comunicación, la capa de paquetes, y la capa de datos.

Los campos de los paquetes se distribuyen como se muestra en la tabla 10.

0x55	STX	SIZE	DATA	CHECK	ETX
1	1	1	N	1	1

Tabla 10. Estructura del paquete de la tarjeta Nellcor.

Dónde:

- 0x55 es el primer campo del paquete, es un valor fijo.
- STX corresponde a “inicio de texto”, es un valor fijo
- SIZE determina el número de bytes en el campo DATA. Es útil para determinar la posición del campo CHECK. Es variable y menor a 38.
- DATA contiene uno o más mensajes. Los mensajes no se pueden dividir entre paquetes y cada paquete contiene al menos un mensaje.
- CHECK permite comprobar la integridad de los datos. Se usa CRC para dicha validación, y solo aplica para los bytes en el campo DATA.
- ETX corresponde a “final de texto”, es un valor fijo

El campo DATA contiene uno o más mensajes. La estructura de un mensaje es:

KEY	LEN	Value1 …ValueN
1	1	N

Tabla 11. Estructura del campo DATA.

- KEY es el identificador del mensaje. No puede coincidir con STX ni ETX
- LEN es el número de bytes en el campo Value.
- Value contiene la información del paquete

Los identificadores KEY se encuentran especificados en la documentación del dispositivo, y cada uno hace uso del campo Value de forma distinta. Se usa el mismo protocolo tanto para la recepción como para el envío de datos (de *host* a tarjeta).

Para realizar el cálculo del CRC existen dos alternativas:

- *Look-up table*:

  El fabricante provee una *look up table (*figura 23), en base a la cual y con una sencilla operación iterativa puede realizarse el cálculo del CRC.

```
unsigned char crcTable[256]={
    0, 94,188,226, 97, 63,221,131,194,156,126, 32,163,253, 31, 65,
  157,195, 33,127,252,162, 64, 30, 95,  1,227,189, 62, 96,130,220,
   35,125,159,193, 66, 28,254,160,225,191, 93,  3,128,222, 60, 98,
  190,224,  2, 92,223,129, 99, 61,124, 34,192,158, 29, 67,161,255,
   70, 24,250,164, 39,121,155,197,132,218, 56,102,229,187, 89,  7,
  219,133,103, 57,186,228,  6, 88, 25, 71,165,251,120, 38,196,154,
  101, 59,217,135,  4, 90,184,230,167,249, 27, 69,198,152,122, 36,
  248,166, 68, 26,153,199, 37,123, 58,100,134,216, 91,  5,231,185,
  140,210, 48,110,237,179, 81, 15, 78, 16,242,172, 47,113,147,205,
   17, 79,173,243,112, 46,204,146,211,141,111, 49,178,236, 14, 80,
  175,241, 19, 77,206,144,114, 44,109, 51,209,143, 12, 82,176,238,
   50,108,142,208, 83, 13,239,177,240,174, 76, 18,145,207, 45,115,
  202,148,118, 40,171,245, 23, 73,  8, 86,180,234,105, 55,213,139,
   87,  9,235,181, 54,104,138,212,149,203, 41,119,244,170, 72, 22,
  233,183, 85, 11,136,214, 52,106, 43,117,151,201, 74, 20,246,168,
  116, 42,200,150, 21, 75,169,247,182,232, 10,84, 215,137,107, 53};
```

Figura 23. LUT para cálculo de CRC [27]

Se hace XOR de cada uno de los bytes con una variable que guarda el elemento de la LUT cuyo índice es la respuesta a la operación anterior.

- Polinomio CRC

$$CRC = X^8 + X^5 + X^4 + 1$$

Esta es la forma estándar para realizar comprobación de redundancia cíclica (CRC). En base al polinomio se realiza la operación lógica mostrada en la figura 24.

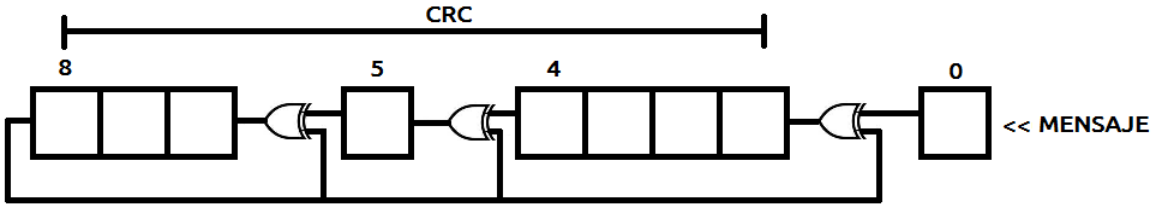

Figura 24. Circuito lógico equivalente al cálculo de CRC

Todas las posiciones del CRC se inicializan en 0. Se desplaza entonces el mensaje bit por bit, realizando las operaciones correspondientes. El cálculo del CRC finaliza cuando el ultimo bit del Mensaje sale de la posición 8 (para que esto suceda, debe desplazarse el registro insertando ceros en la posición cero), y el valor del CRC se ubica entre la posición 8 y la posición 1.

- Suntech

Este protocolo posee dos tipos de estructuras: una para envío y otra para recepción.

Host → Tarjeta

START	CMD	DATA	CKSUM
1	1	N	1

Tabla 12. Estructura del paquete de la tarjeta Suntech, *host* a tarjeta.

La estructura de este paquete se muestra en la tabla 12, y sus campos tienen las siguientes características:

- START es un carácter fijo que marca el inicio del paquete host
- CMD define la operación del modulo
- DATA contiene los parámetros del comando a enviar. Su tamaño varía según el comando
- CKSUM permite comprobar la integridad de los datos.

Tarjeta → Host

START	PACKET	DATA	CKSUM
1	1	N	1

Tabla 13. Estructura del paquete de la tarjeta Suntech, tarjeta a *host*.

La estructura de este paquete se muestra en la tabla 13, y sus campos tienen las siguientes características:

- START es un carácter fijo que marca el inicio del paquete cliente
- PACKET define el tipo de paquete
- DATA contiene los parámetros del comando a enviar. Su tamaño varía según el paquete
- CKSUM permite comprobar la integridad de los datos.

El *checksum* se realiza de la siguiente manera:

CHECKSUM = 0x100 – 256 % (START + PACKET/CMD + DATA)

La suma de todos los bytes en el paquete incluido el *checksum* debe ser igual a cero.

- Biomedian

HEADER	LEN	CODE	DATA	CHECKSUM
2	1	1	N	1

Tabla 14. Estructura del paquete de la tarjeta Biomedian.

La estructura del paquete de la tarjeta Biomedian se muestra en la tabla 14. Posee los siguientes campos:

- HEADER: Posee 2 bytes fijos, que indican el inicio del paquete
- LEN: Establece la longitud de DATA + CODE
- CODE: Identificador del tipo de paquete. Los identificadores se encuentran definidos en la especificación del protocolo del fabricante.
- DATA: Señales y parámetros, de longitud variable. Pueden venir agrupados en bytes o *words* según el parámetro.

WORD	
Byte más significativo	Byte menos significativo

Tabla 15. Bytes en dato tipo *word*.

- CHECKSUM: Permite comprobar la integridad de los datos

El *checksum* se realiza de la siguiente manera:

CHECKSUM = (LEN + CODE + DATA) & 0xFF

Se usa la misma estructura de paquete para comunicación del host a la tarjeta o viceversa.

- Goldwei

START	ID	DATA	CKSUM	END
1	1	N	1	1

Tabla 16. Estructura del paquete de la tarjeta Goldwei.

La estructura del paquete de la tarjeta Goldwei se muestra en la tabla 16, y posee los siguientes campos:

- START: Indica el inicio del paquete, es un valor fijo
- ID: Indica el tipo del paquete. Está definido en la especificación del protocolo del fabricante
- DATA: Datos relacionados al ID. Pueden venir en cualquier formato (desde *unsigned char* hasta *double floating point*). El formato de cada uno de ellos está relacionado también con el ID.
- CKSUM: Permite comprobar la integridad de los datos
- END: Indica el final del paquete, es un valor fijo

El *checksum* se calcula de la siguiente forma:

CHECKSUM = (suma (DATA) & 0XFF) XOR 0XF0

El protocolo de esta tarjeta tiene la particularidad de ser parcialmente legible en ASCII, facilitando la inspección de sus mensajes (sobre todo mensajes de error).

El uso del formato de punto flotante permite alcanzar una mayor precisión en los datos, al igual que evitar dañar señales muy débiles por errores de redondeo. Es posible configurar la tarjeta para usar dos formatos de punto flotante:

- Formato de punto flotante rápido Motorola

MANTISA	SIGNO	EXPONENTE
24 Bits	1 Bit	7 Bits

Tabla 17. Formato de punto flotante rápido, Motorola

Para realizar la conversión de estos valores a un número flotante se realiza la siguiente operación (debe guardarse en un tipo de dato adecuado para evitar sobrescribir otros elementos en memoria):

$$FP = (-1)^{SIGNO} * 2^{EXP-0x40} * \frac{MANTISA}{0x1000000}$$

- Formato de punto flotante IEEE

MANTISA	SIGNO	EXPONENTE
23 Bits	1 Bit	8 Bits

Tabla 18. Formato de punto flotante, IEEE

Para realizar la conversión de estos valores a un número flotante se realiza la siguiente operación (debe guardarse en un tipo de dato adecuado para evitar sobrescribir otros elementos en memoria):

$$FP = (-1)^{SIGNO} * 2^{EXP-0x7F} * \frac{MANTISA}{0x800000}$$

4.1.1.9 Instalación y configuración de las librerías GTK necesarias para el desarrollo de la GUI

Se instalan las dependencias necesarias para compilar el código fuente. Los paquetes se mencionan a continuación, (cabe aclarar que las dependencias que instala la versión para ARM de Ubuntu Linux ya se encuentran portadas para la arquitectura):

- libgtk-3-0
- libgtk-3-dev
- libglade2-dev
- libtool
- fuentes tipográficas de la interfaz

Para su instalación se utilizó la herramienta *apt-get*. Para la edición del archivo UI (encargado de contener la estructura de la interfaz) se probaron varios programas, con diferentes resultados:

- Glade 2

  Diseñador de GUIs basadas en GTK. Totalmente funcional con los objetos de GTK+2, pero desactualizada respecto a GTK+3. Es la versión más extendida de la aplicación, y los archivos de recursos (XML) no son compatibles con los de versiones posteriores (UI), su estructura cambia ligeramente.

- Glade 3

  Diseñador de GUIs basadas en GTK. Posee nuevos objetos en comparación a Glade 2 y es compatible con GTK+2 y GTK+3. Soporta los archivos UI.

- Anjuta IDE

  Entorno de desarrollo para Aplicaciones con GUI basada en GTK. Posee Glade 3 como uno de sus elementos, así como un editor de texto, compilador GCC integrado y *autotools*. Fue seleccionado para el desarrollo debido a que contiene todos los elementos necesarios para el desarrollo en una sola herramienta.

La edición del archivo de recursos se realizó en un ordenador portátil(x86). Anjuta creo un proyecto nuevo con la estructura estándar de *autotools*. Los scripts de configuración se crearon en la raíz del proyecto, los archivos fuente en un subdirectorio llamado *src*, y los recursos visuales en otro subdirectorio llamado *rc.*

Se configuro el compilador para que usara el perfil de *debugging*, para así permitir posteriormente a GDB seguir el estado del programa en caso de fallas.

4.1.1.10 Identificación e implementación de tareas básicas de la librería GTK

La librería GTK, se compone básicamente de un gran conjunto de objetos con diferentes jerarquías. La identificación de los elementos básicos y de la jerarquía es crucial para diseñar la implementación de cualquier aplicación grafica basada en la misma. En la figura 25 se muestra la jerarquía de objetos resumida de la librería GTK+.

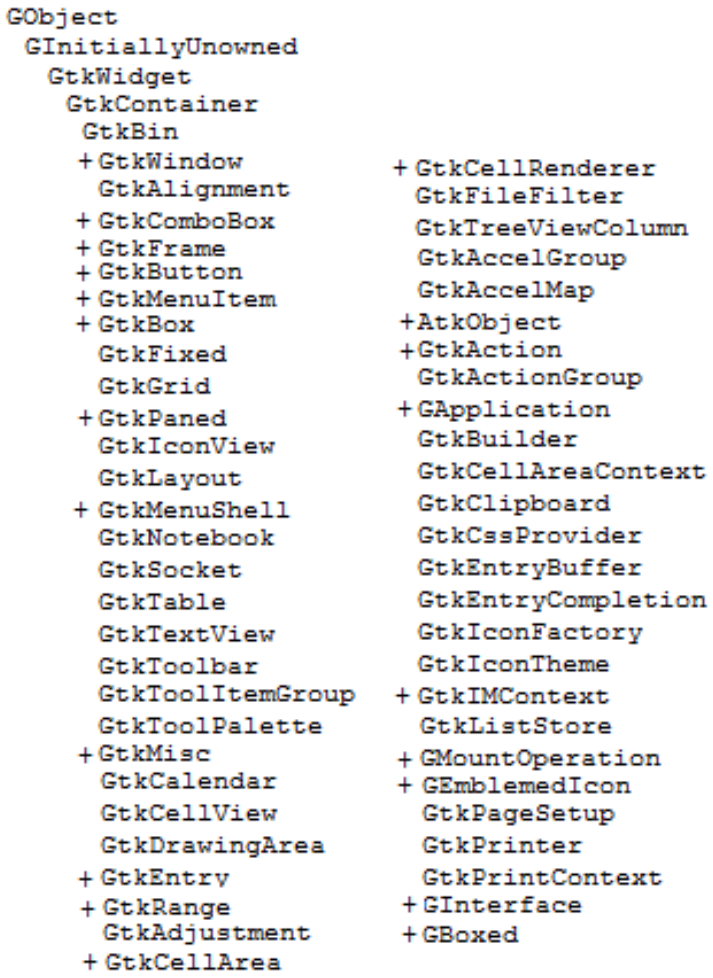

Figura 25. Jerarquía reducida de la librería GTK

Dependiendo de la aplicación se usaran ciertos elementos. Sin embargo hay algunos que son comunes a todas las aplicaciones.

GObject: Todo objeto es un GObject. Las funciones asociadas a interactuar con GObjects poseen compatibilidad con cualquier objeto de GTK o buscan hacer conversión a otro tipo de objeto o dato.

GtkWidget: Cualquier objeto después de ser localizado en la jerarquía de objetos de una aplicación. Es el objeto más utilizado en la librería, y su interacción es más sencilla que con GObject.

GtkWindow : Ventana. Toda aplicación debe tener al menos una ventana principal y eso hace obligatorio su uso.

GtkButton : Botón. De este objeto derivan los diferentes tipos de botones, cada uno hereda las características base de su padre (botón) ademas de poseer características específicas a su función.

GtkGrid: Malla. GTK a diferencia de otras librerías GUI está orientado a ubicar sus widgets en celdas dentro de mallas y no en coordenadas, esto permite a las aplicaciones ser redimensionadas sin que por ello se oculten los objetos.

GtkMisc : Contiene objetos sencillos varios. De este objeto se deriva GtkLabel, GtkArrow y GtkImage.

Las tareas básicas a realizar corresponden a la creación de una aplicación grafica de prueba. Se creó entonces una aplicación donde se mostrara una ventana con un título. El código utilizado fue el siguiente:

```
#include <gtk/gtk.h>
Void main (int   argc,char *argv[])
{
  GtkWidget *window;
  gtk_init (&argc, &argv);
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);
  gtk_widget_show (window);
  gtk_main ();
}
```

Se compilo el programa usando GCC:

```
gcc `pkg-config --cflags gtk+-3.0` -o prueba prueba.c `pkg-config --
libs gtk+-3.0`
```

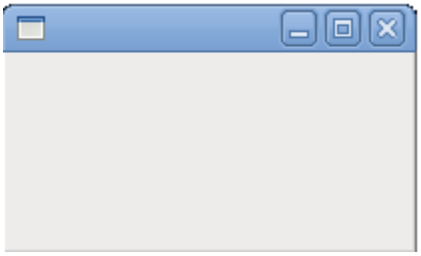

Figura 26. Aplicación básica en GTK

En esta aplicación de demostración se encuentran varios elementos básicos de una aplicación gráfica, por ello se analizara la función de cada línea.

*GtkWidget *window*
Creacion de una variable GtkWidget. De forma similar a como se declaran variables de tipos de datos estándar en C (*int, char, short*) , pueden declararse variables de tipos de datos definidos en una librería (aunque en este caso la variable en realidad es un objeto y está diseñada para interactuar como tal).

gtk_init (&argc, &argv)
GTK siempre necesita ser inicializado, esto se logra con la función anterior. Sus argumentos son las variables que recibe cuando la aplicación es lanzada por consola (aunque no son necesarios). Si los argumentos no son reconocidos, son ignorados por parte de GTK.

window = gtk_window_new (GTK_WINDOW_TOPLEVEL)
Creación de una ventana, como *toplevel*. Es guardada en la variable (GtkWidget) que creamos anteriormente, para así hacer futuras referencias a ella por dicha variable.

g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL)
Conexión de un evento en el GtkWidget *window*, a una función estándar de GTK (gtk_main_quit). Lo que se logra con ello es que al cerrar la ventana, el sistema de GTK también termine. Esto es de vital importancia para evitar fugas de memoria (si GTK+ no concluye apropiadamente puede que la memoria donde se encontraba quede inaccesible hasta reiniciar el ordenador, ya que C no posee un recolector de basura que la libere).

gtk_widget_show (window)
Muestra el GtkWidget *window*, en la pantalla activa.

gtk_main ()
Proceso principal de GTK. Debe ejecutarse para que GTK se inicie correctamente.

4.1.1.11 Diseño del algoritmo de decodificación del protocolo de transmisión de las tarjetas OEM

El diseño del algoritmo de decodificación se realizó en cada tarjeta de forma separada, basándose en las especificaciones de las hojas de datos de cada fabricante. Cada tarjeta tiene un algoritmo de decodificación diferente y específico para realizar solo las tareas objeto de esta práctica (aunque con pequeñas modificaciones es posible agregar cualquier otra funcionalidad necesaria).La primera implementación del algoritmo de decodificación regresa los valores como texto y solo se dedica a esa tarea. De la figura 27 a la 31 se muestran los diagramas de flujo de los algoritmos de decodificación.

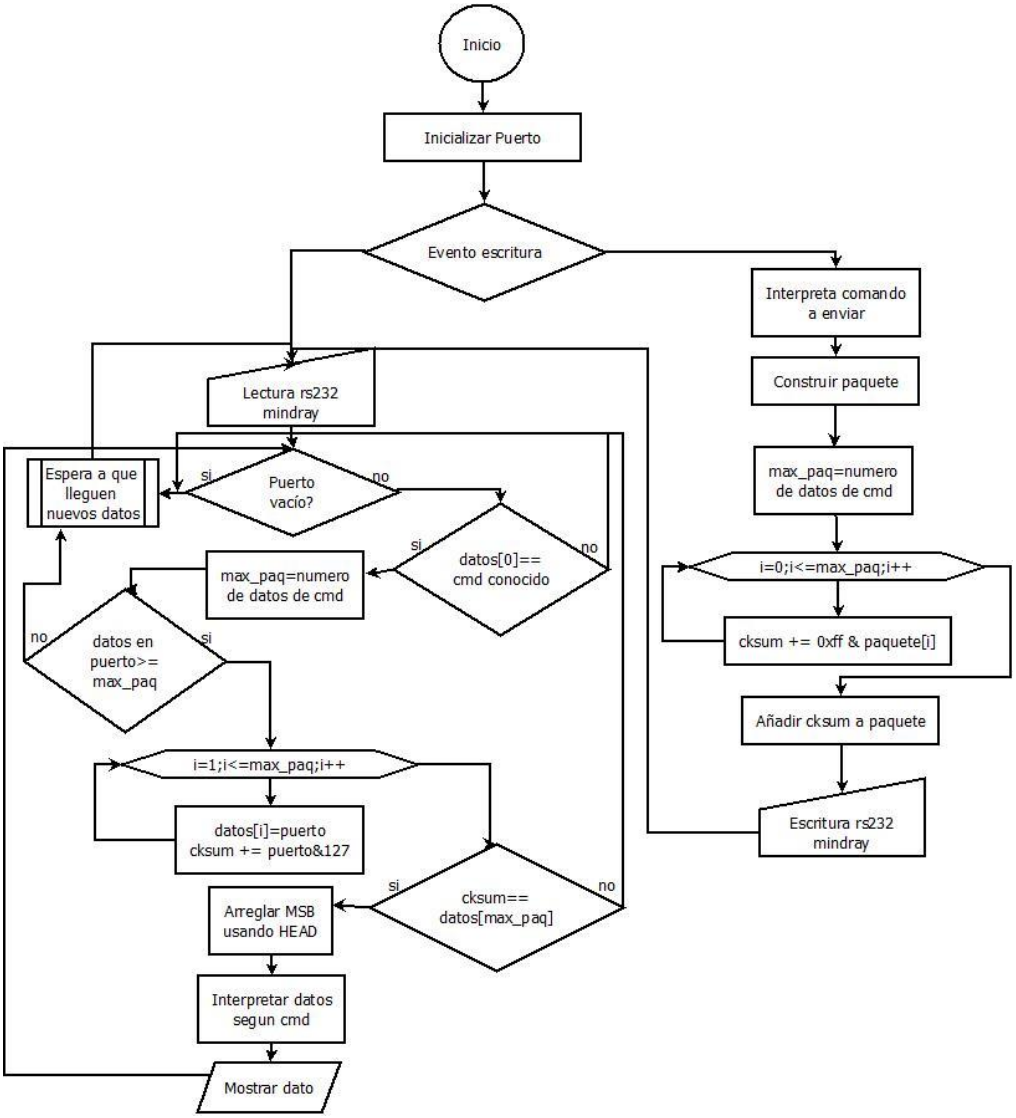

Figura 27. Diagrama de flujo del algoritmo de decodificación de la tarjeta Mindray.

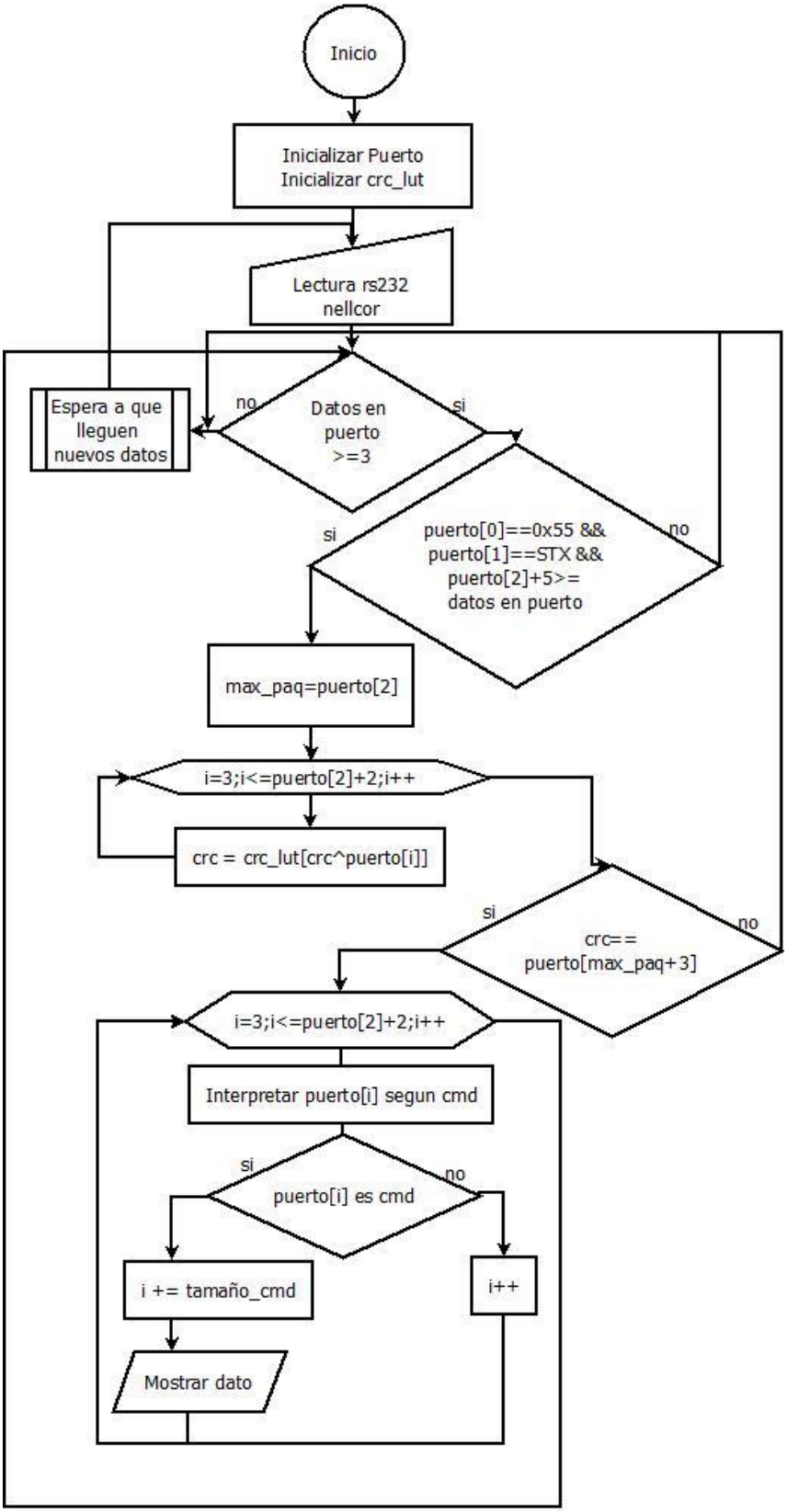

Figura 28. Diagrama de flujo del algoritmo de decodificación de la tarjeta Nellcor

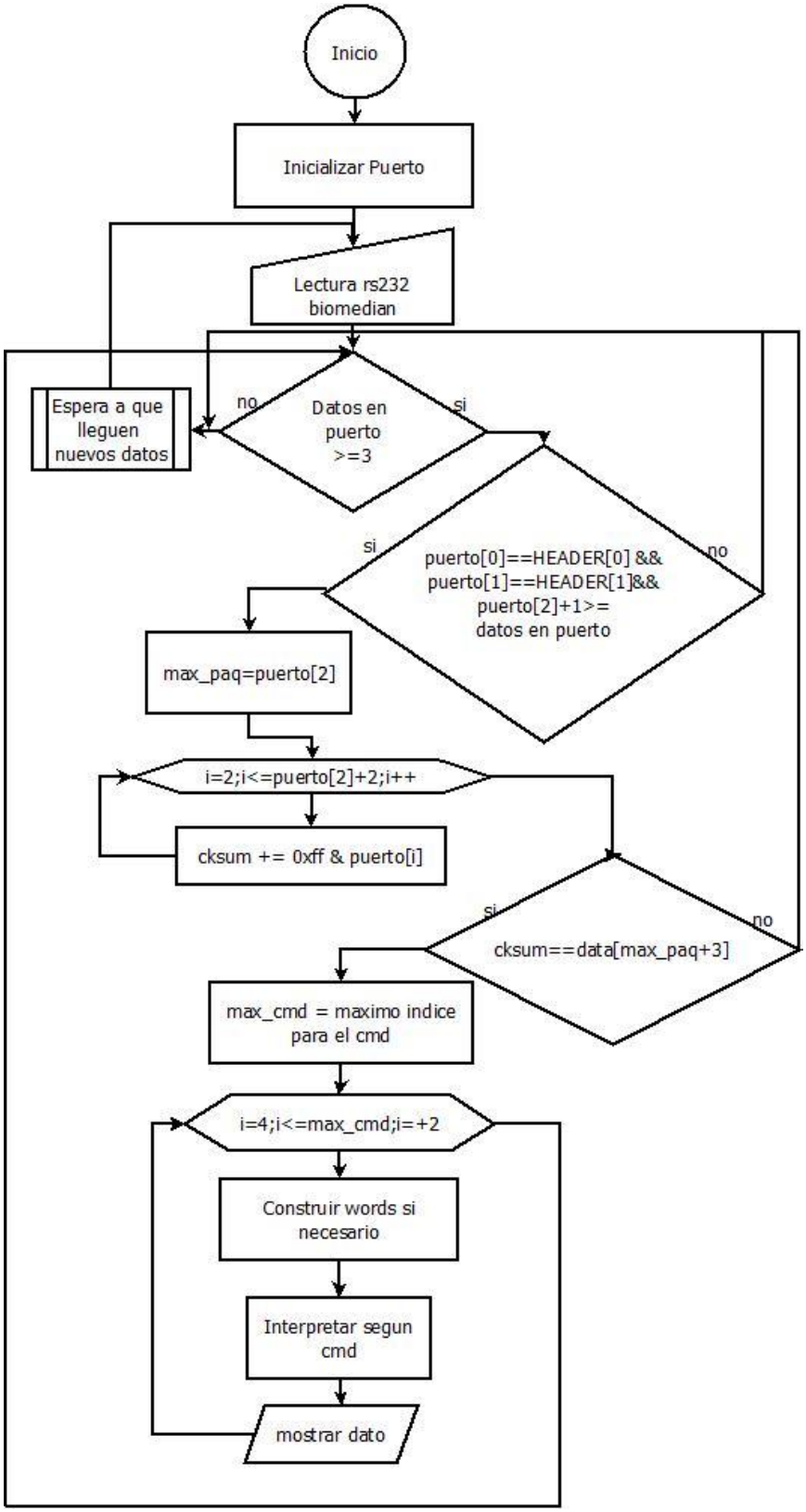

Figura 29. Diagrama de flujo del algoritmo de decodificación de la tarjeta Biomedian

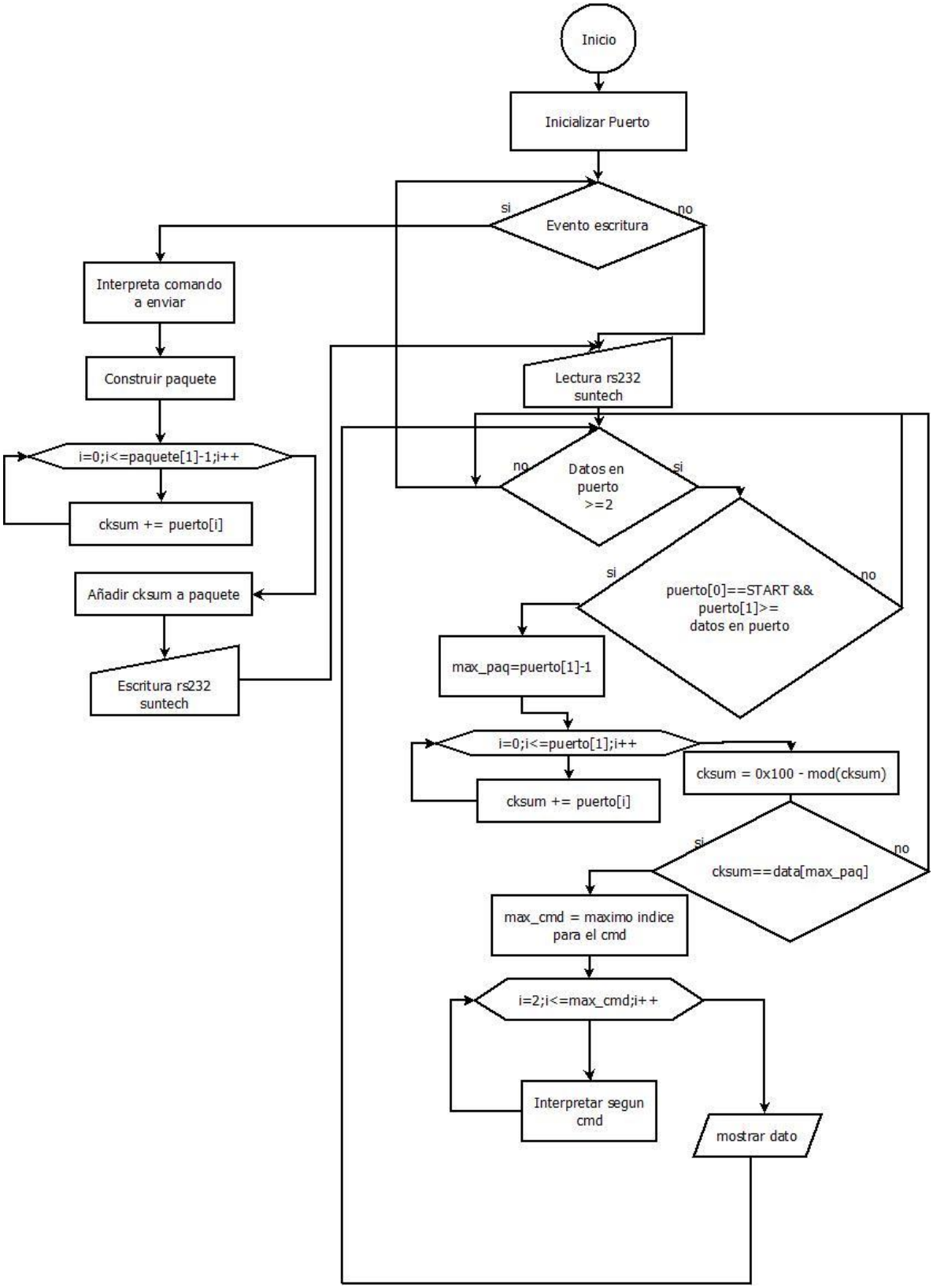

Figura 30. Diagrama de flujo del algoritmo de decodificación de la tarjeta Suntech

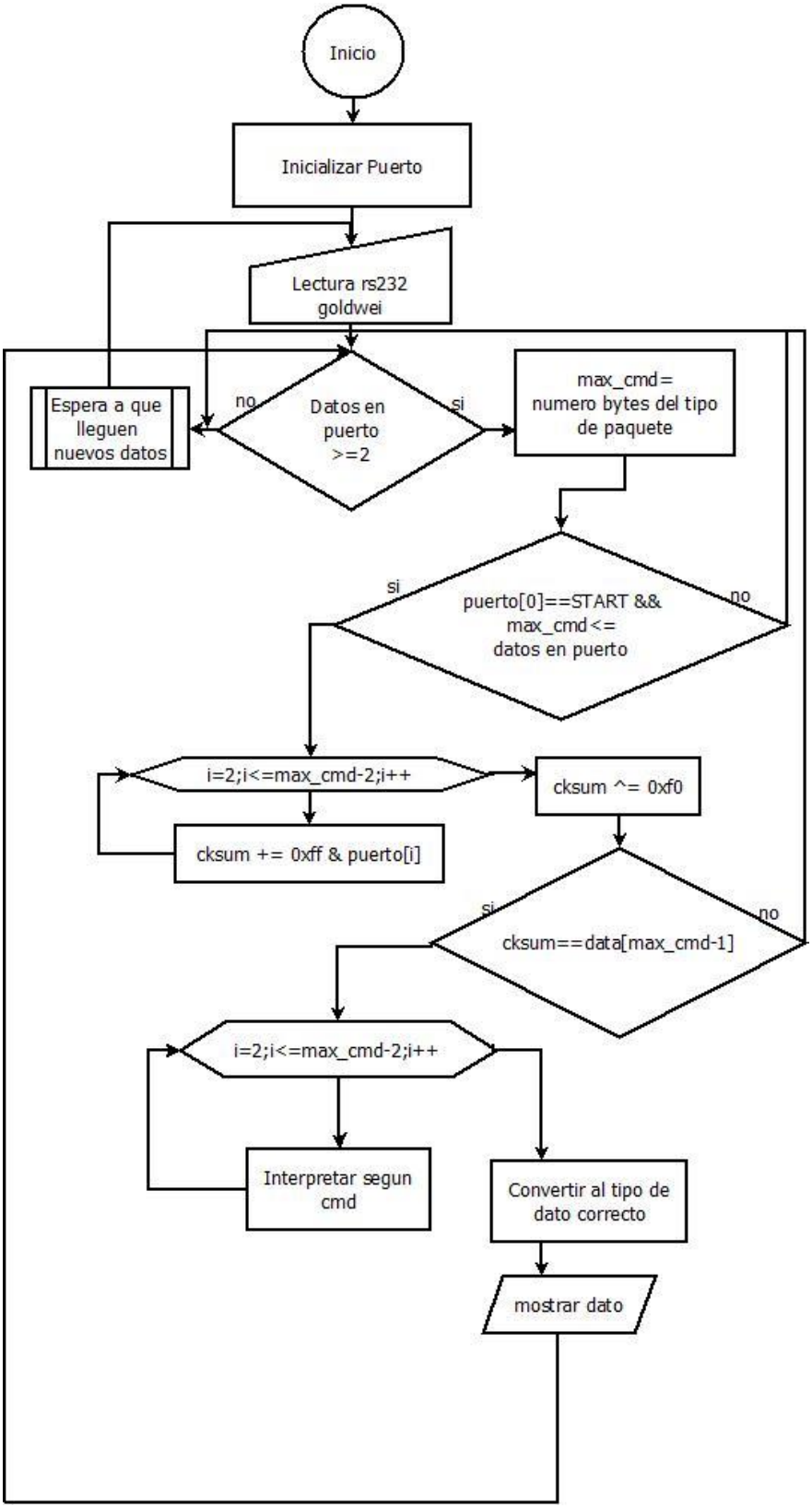

Figura 31. Diagrama de flujo del algoritmo de decodificación de la tarjeta Goldwei

Ciertos elementos en los diagramas de flujo no se especifican explícitamente debido a que son propiedad intelectual de la FCV, en especial los identificadores y la interpretación de los paquetes procesados según el comando. Se pudo establecer comunicación con las tarjetas satisfactoriamente, las cuales retornaban un valor correspondiente en su documentación al de “sensor desconectado”, debido a que aún no se tenía una señal en la entrada de sus sensores.

4.1.1.12 Identificación del flujo de ejecución de una aplicación grafica basada en eventos

Para implementar una interfaz gráfica es necesario comprender que su funcionamiento difiere del de una aplicación lineal, ya que debe estar constantemente actualizando y revisando todos sus componentes, y además de ello debe interactuar con el código propio de la aplicación. Básicamente la mayoría de librerías graficas usan un paradigma de programación llamado “Programación dirigida por eventos”, donde la estructura y la ejecución de la aplicación se define por eventos que ocurran en el sistema (figura 32), definidos por el usuario o por el sistema. Difiere de la programación secuencial en que el flujo del programa no es definido por el programador sino por la misma aplicación, el programador solo podrá requerir la ejecución de porciones de código en respuesta a eventos de la GUI.

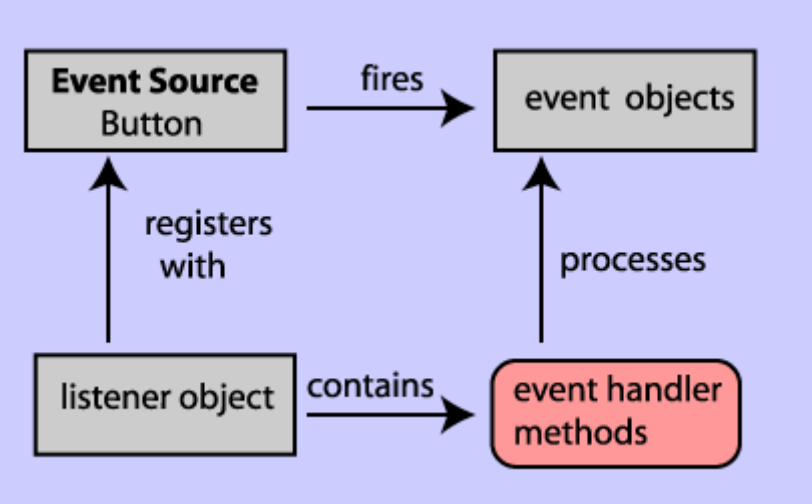

Figura 32. Funcionamiento de aplicación grafica basada en eventos [28]

Fuente de eventos: Objeto con la capacidad de generar eventos. El tipo de evento generado depende del objeto.

Objeto escucha: contiene los métodos a ejecutar cuando se genera un evento. Son definidos por el programador

En la figura 31 se ilustra como un objeto escucha se registra con una fuente de eventos, y tiene la función de procesar los eventos, mientras que la fuente de

eventos solo se encarga de disparar dichos eventos. En algunos lenguajes de programación estos eventos se consideran objetos en sí mismos.

El funcionamiento de la librería GTK+ está basado en el paradigma de la programación dirigida por eventos, y el flujo de ejecución de una aplicación basada en esta librería contiene los siguientes elementos:

gtk_init():
Inicialización de GTK. Su ejecución es necesaria para correr cualquier aplicación basada en GTK+.

gtk_builder_new ():
Crea un nuevo objeto tipo *builder*. El objeto tipo *builder* se asocia posteriormente a un archivo UI (*User Interface*) cuya estructura es similar a un XML y contiene el árbol de objetos de la aplicación.

gtk_builder_add_from_file ():
Lee un archivo UI y lo almacena en un objeto *builder* previamente creado.

gtk_css_provider_load_from_path():
Lee un archivo CSS (Cascading Style Sheet) y configura la apariencia de los objetos de la interfaz (*skin*).

Topwindow:
En el archivo UI debe crearse una ventana principal, la cual se mostrara inmediatamente al iniciar el programa, con los *widgets* contenidos en ella mediante la función gtk_widget_show_all(topwindow).

gtk_main ():
Lazo principal (*main loop*) de la aplicación GTK+. Después de la ejecución de este método el control del flujo de ejecución pasa a ser manejado por GTK+, dirigido por sus eventos y *handlers*. La única forma de regresar a la forma de ejecución normal es saliendo del método lo cual inmediatamente cierra cualquier ventana de GTK+

Cuando el usuario no opera la GUI, GTK se queda ejecutando el lazo principal, esperando alguna entrada. Si el usuario realiza una acción el lazo sale del estado de espera a enviar un evento a GTK+. La librería dirige entonces el evento a uno o más *Widgets*, los cuales al recibirlo emiten una o varias señales, que notifican al programa que algo ha sucedido, e invocan funciones conectadas a estas señales (esto puede hacerse manualmente o automáticamente si los métodos se referencian en el archivo UI). Después de ejecutar los métodos asociados GTK+ regresa al lazo principal esperando por alguna otra acción. El diagrama de flujo de la figura 33 ilustra el procedimiento descrito anteriormente.

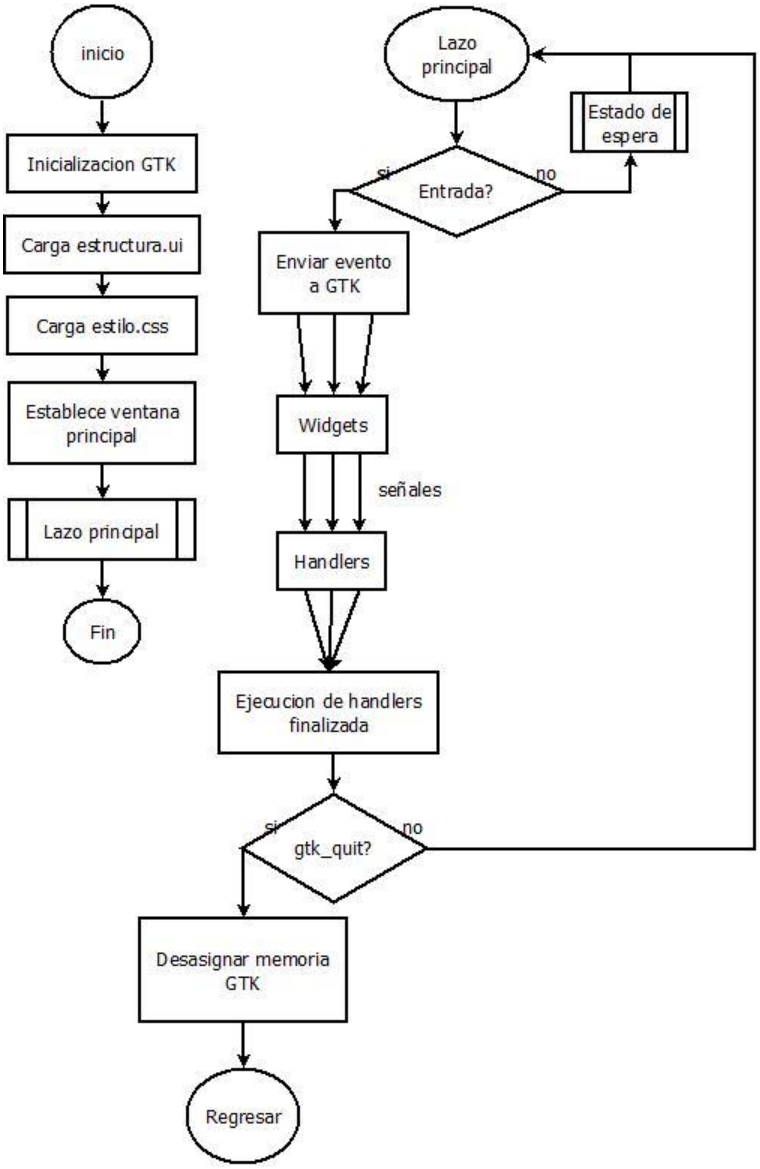

Figura 33. Diagrama de flujo de aplicación básica GTK

4.1.1.13 Implementación de la interfaz gráfica usando GTK

Usando los recursos gráficos suministrados por el diseñador industrial, se inicia entonces la implementación de la interfaz gráfica usando GTK. Los recursos fueron suministrados en forma de imágenes o archivos PDF, con sus medidas y colores (especificados en formato RGB). Muchos de los elementos necesarios fueron ligeramente modificados usando la herramienta GIMP, ya sea para leer sus valores de color, crear imágenes con canal alfa (transparencia), convertir entre formatos o ajustar su tamaño.

Debido a que muchas de las características visuales de la interfaz no son estándar en la librería GTK, se hizo necesario implementar estas características una por una buscando la mejor manera de ajustar los objetos base de GTK a la aplicación. En su mayoría se usaron imágenes con objetos especializados en detección de eventos, los cuales actuaban sobre las imágenes para emular el comportamiento de un botón, de forma similar al comportamiento de botones web. Se sobrepusieron varios *Widgets* gracias a las capacidades extendidas de las mallas (*grids*), (no se integraron como hijos entre ellos en la jerarquía ya que esto no permitía su correcto funcionamiento).

La implementación de la interfaz gráfica se hizo en 3 fases:

- Primera implementación: No fue utilizado ningún IDE, y el archivo de recursos fue editado usando Glade 3. Se compilo de forma manual, sin usar *autotools*, y no se conectaron eventos a ningún botón. La apariencia de esta implementación está basada en la versión 1.0 de la interfaz, por tanto algunas características difieren de versiones posteriores.

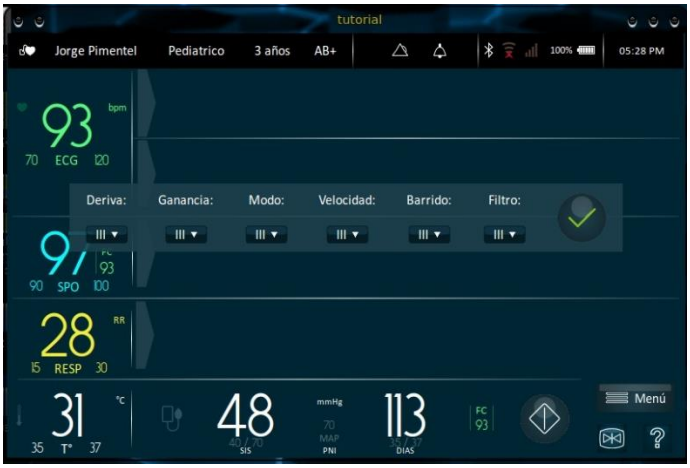

Figura 34. Primera implementación de la interfaz grafica

La aplicación se dividió en ventanas, una principal y dos secundarias correspondientes a menús (desde la perspectiva del usuario). En la figura 33 se pueden ver dos de las ventanas implementadas inicialmente.

- Segunda Implementación: Se hizo la transición al entorno de desarrollo Anjuta IDE, donde se realizaron modificaciones a la interfaz gráfica actualizando sus características a una versión más reciente del diseño de la interfaz gráfica.

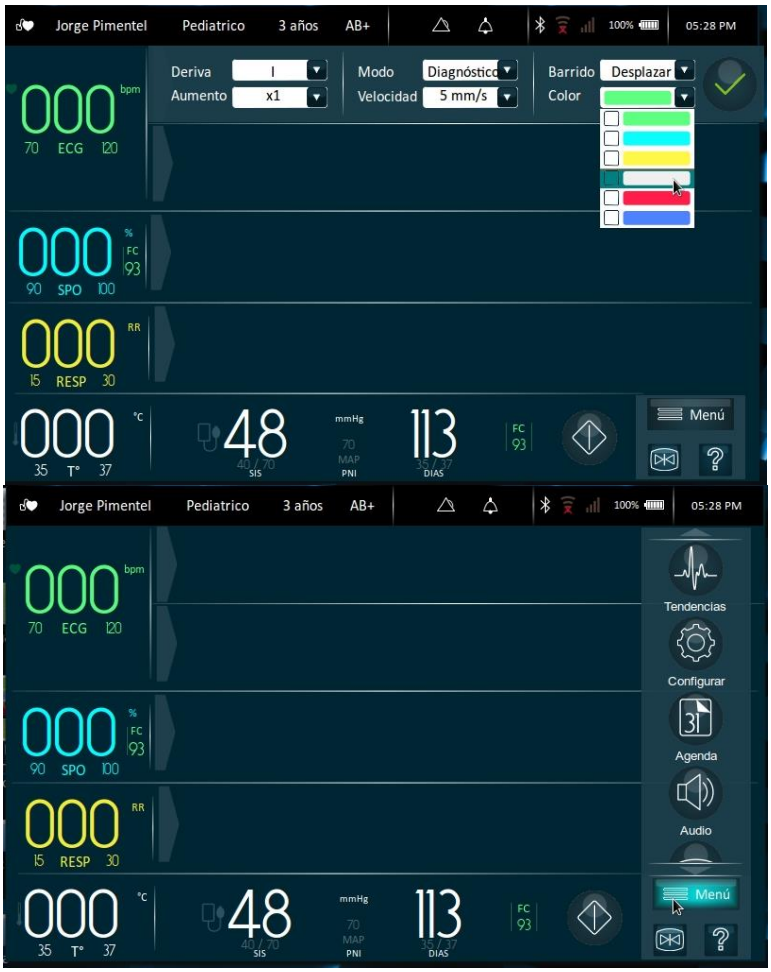

Figura 35. Segunda implementación de la interfaz grafica

Se retiraron los bordes de ventana y se agregaron algunos eventos relacionados con la interacción entre los menús y los elementos que los desplegaban. Se crearon elementos adicionales (Sub-menús desplegables) y se añadió la capacidad de deslizamiento al menú lateral.

- Tercera Implementación: Usando la librería Cairo API, se creó el código encargado de graficar, asociado con los colores especificados en el diseño de la

interfaz, y con los eventos de GTK. Se simularon señales en el programa para comprobar el correcto funcionamiento de la interfaz, con una señal rampa.

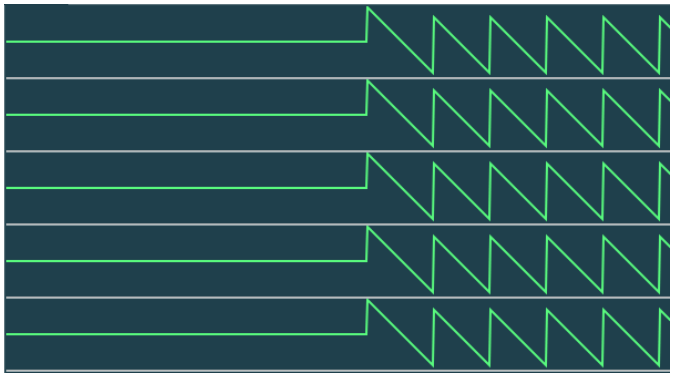

Figura 36. Simulación de señales en interfaz grafica

Fue necesario para ello crear *handlers* que usaran Cairo API para graficar una serie de puntos en el GtkWidget. Los *handlers* debían ser llamados periódicamente para calcular e insertar los nuevos valores de las señales. Para ello se usaron funciones cuya ejecución fue programada a intervalos regulares en GTK, por funciones en la librería encargadas de esta tarea en específico:

- *g_timeout_add()* , que ejecuta funciones en intervalos de ms
- *g_idle_add(),* que ejecuta funciones en cualquier intervalo libre del lazo principal

Se descartó el uso de g_idle_add() por su funcionamiento irregular , alto consumo de tiempo de procesamiento y se realizaron pruebas con los intervalos de g_timeout_add(), en búsqueda del valor optimo donde no hubieran diferencias de velocidad entre datos. Cabe aclarar que los valores en ms no son ejecutados de forma exacta, ya que GTK es quien decide cuándo puede seguir con la ejecución de esta tarea programada, siendo así es necesario buscar el equilibrio de tiempo entre la aplicación gráfica y los intervalos de graficación.

Posteriormente se realizó la integración del algoritmo de la tarjeta Mindray dentro de la aplicación. El comportamiento del algoritmo en comparación a las señales simuladas presenta varias diferencias. Mientras que las señales simuladas no requerían de métodos de sincronización de sus datos (ya que no eran más que el resultado de una operación matemática), para las señales reales se requiere que sean obtenidas, procesadas y guardadas en espera de ser graficadas, todo esto en el menor tiempo posible y sin afectar el funcionamiento del resto de la aplicación.

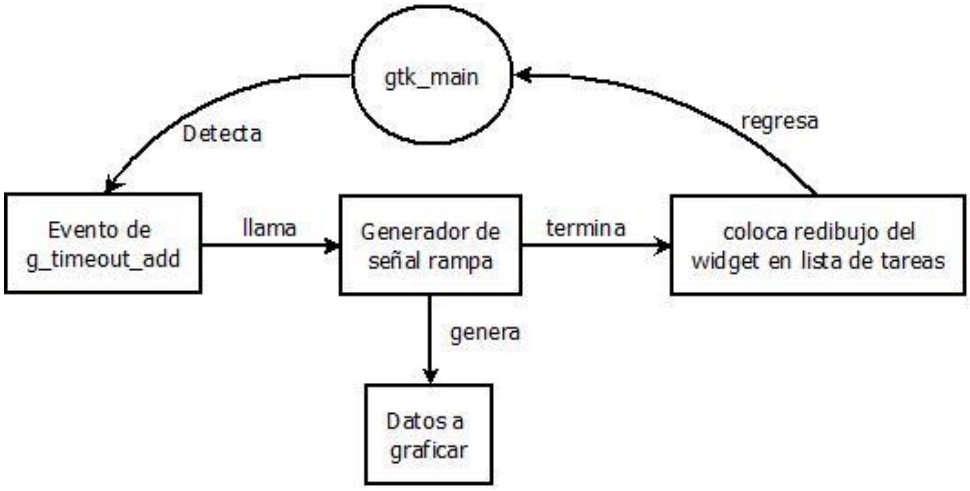

Figura 37. Diagrama de flujo de aplicación con señales simuladas

En la figura 37 se muestra el orden de ejecución de la aplicación simulando las señales. Nótese que la señal se genera inmediatamente antes de ser graficada y por ello no requiere ser almacenada, caso contrario a una señal real donde el dato llega asíncrono al proceso de graficado y cuyo procesamiento puede retrasar la respuesta de la gráfica. Para evitar estos fenómenos se incluyeron dos registros FIFO donde se introducen datos periódicamente desde el puerto y se extraen a medida que la aplicación los solicite (se busca también que estas solicitudes sean periódicas y las pausas entre ellas poco perceptibles para el usuario). Las operaciones realizadas en un FIFO se llaman *pop* y *queque*, las cuales extraen e introducen datos, respectivamente. Se usaron FIFOs circulares, los cuales tienen la característica de ser más eficientes a comparación de los lineales, debido a que estos últimos desplazan todos sus datos en cada iteración mientras que los circulares solo desplazan sus índices. El algoritmo resultante se muestra en la figura 39.

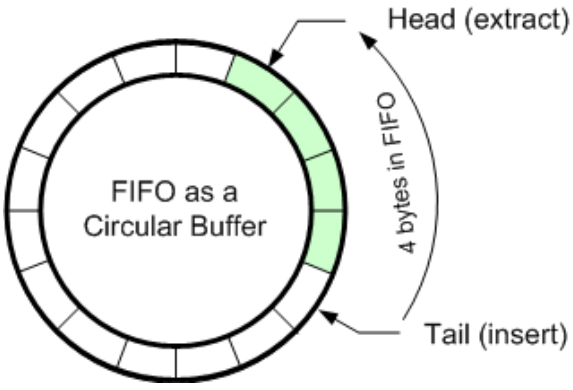

Figura 38. FIFO circular[29]

En la figura 38 se muestra la función de los FIFOs en la decodificación de los datos. Gracias a ellos se pudo dividir el proceso en tres tareas más sencillas: lectura de datos en bruto, procesamiento de los datos, y graficado, con frecuencias ajustables manualmente, lo cual al finalizar el desarrollo permitirá su ejecución optima y ajuste a las capacidades del dispositivo final, el sistema embebido.

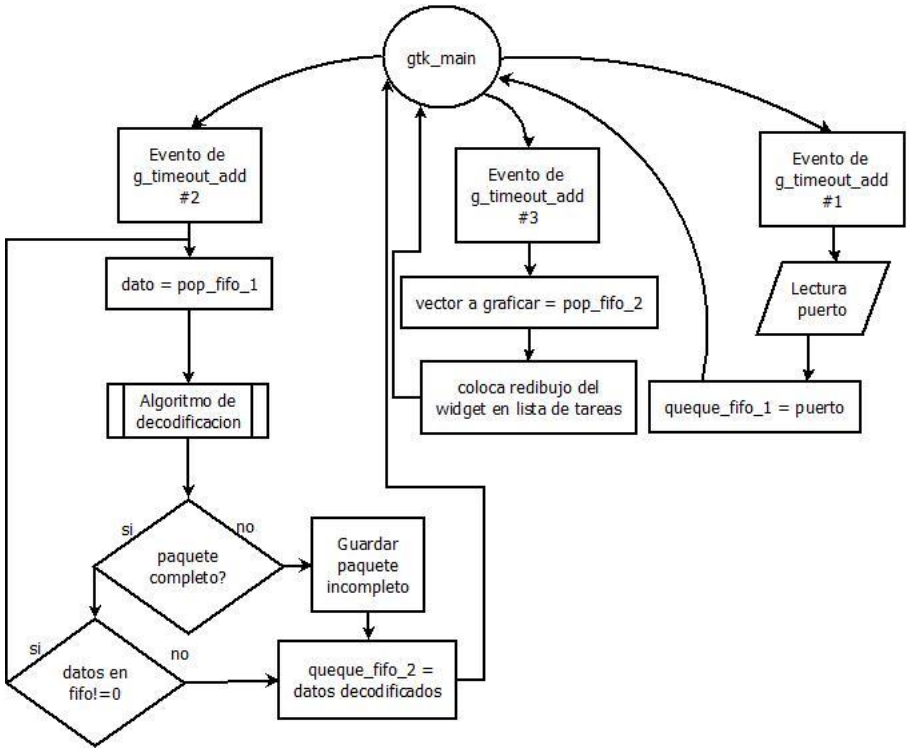

Figura 39. Diagrama de flujo de aplicación con señales reales

Un detalle importante a mencionar es que durante las pruebas, la necesidad de los FIFO se hizo evidente al intentar extraer dato por dato del puerto (a diferencia de extraer grupos de datos con longitud fija), lo cual hizo decaer el rendimiento de la aplicación significativamente. Al mismo tiempo se intentó procesar los datos de llegada (Desde las OEM), lo cual conllevo a pérdidas de datos (que se mostraban en las gráficas como discontinuidades) y retrasos en su graficado (realizar ambos procedimientos de forma simultanea toma un tiempo considerable). Después de realizar ajustes en los intervalos de lectura, procesamiento y graficado estos inconvenientes dejaron de suceder.

Se incluyeron gráficas y parámetros adicionales a los existentes en la especificación de la interfaz gráfica. Se copiaron y modificaron las estructuras de los parámetros ya implementados, para así mostrar homogeneidad en la interfaz.

4.1.1.14 Identificación de funciones y *bindings* a funciones de propósito general, parte de Glib

Con el objetivo de mejorar la estabilidad de la aplicación, y en base a la documentación del proyecto GTK, se busca sustituir funciones de la librería estándar C utilizadas en el programa por alternativas que interactúen de manera más segura con el lazo principal de GTK.

El lazo principal (*main loop*) de GTK es la base de cualquier aplicación basada en esta librería, por tanto cualquier interacción incorrecta entre este y el código generara inestabilidad a la aplicación. Por ello se recomienda en lo posible usar los tipos y funciones estándar como reemplazo de las librerías estándar de C. En la tabla 19 se muestran algunas de las funciones estándar de C relevantes para la aplicación y su *binding* en GTK. Muchas de ellas tienen la misma sintaxis.

GTK	Estandar
gint, gboolean, gdouble, gchar	Int,bool,double,char
g_print	Printf
g_thread	pthread
g_free	free
g_strlcpy	strcpy
g_ascii_dtostr	itoa

Tabla 19. Ejemplo de funciones y tipos estándar, con su reemplazo en GTK

Para permitir la interacción de variables estándar de C con los *bindings* de GTK e incluso para usar los resultados de alguna de sus funciones como argumento de otra es necesario hacer la conversión de tipos respectiva (casting). Para ello se recomienda usar los macros definidos por GTK para cada tipo propio de la librería en vez de hacer la conversión manualmente, ya que así pueden detectarse errores de conversión en tiempo de ejecución.

Los inconvenientes de algunas funciones estándar con GTK residen principalmente en el manejo de los *threads y* en la prioridad que se le da a funciones externas a GTK dentro de estos *threads*. Un ejemplo de ello es la función *printf*, básica en cualquier aplicación que permite a los programadores detectar errores en el programa en tiempo de ejecución (mediante la impresión de parámetros del programa). A pesar de que ambas funciones se ejecutan correctamente, *printf* “acumula” impresiones cuando es lanzada por un *handler* de GTK, mientras que g_print imprime uno a uno los valores. Los errores pueden ir mas allá de esto, como pasa con *pthread*, que puede corromper el espacio de memoria de los hilos de GTK.

4.1.1.15 Configuración de drivers del sistema

A pesar de que el sistema es básicamente funcional, para asegurar el mejor rendimiento de la aplicación es necesario que los módulos encargados de manejar

la GPU y los puertos seriales emulados (FTDI – USB) funcionen correctamente. Explorando en las propiedades del sistema y los drivers instalados se detecta que ninguno de los dos módulos se encuentra instalado, por ello se buscan alternativas para su instalación.

En muchos de los sitios consultados la solución del problema involucraba compilar el driver desde los archivos fuentes, y relacionarlo con el núcleo del sistema (para lo cual hay que recompilarlo), procedimiento tedioso y riesgoso para el sistema. Se decide entonces buscar una imagen preinstalada de Ubuntu que posea dichos drivers (que en los ordenadores de escritorio es estándar y no requiere instalación). Finalmente se encuentra una imagen con dichas características. Se requiere entonces instalar y volver a implementar todas las características del sistema hechas hasta el momento, aun asi, y gracias a que todo el procedimiento fue documentado paso a paso, fue posible replicarlo en la nueva instalación.

Los drivers de FTDI son libres y funcionaron después de instalar la aplicación Minicom (usando *apt-get install*) encargada de gestionar el puerto serial y comprobar que detectaba los dispositivos FTDI-USB conectados. Por otro lado los drivers de la GPU son privativos y necesitan ser activados antes de ser utilizados. Para ello se accedió a la configuración del sistema, donde se instaló bajo la categoría de “*hardware adicional*”.

Posterior a su activación los efectos como transparencias fueron visibles y los movimientos de ventanas en el entorno Fluxbox se suavizaron, mostrando una evidente mejora en el rendimiento del sistema.

4.1.1.16 Implementación de la interfaz gráfica en la tarjeta de desarrollo *Pandaboard*

Se continua el desarrollo de la interfaz iniciado previamente en el ordenador(x86) portátil bajo Ubuntu Linux, en la tarjeta de desarrollo *Pandaboard*. Inicialmente se planeó realizar compilación cruzada, sin embargo, esto hace el proceso más complejo y además, la plataforma tiene suficiente potencia para realizar esta tarea sin inconvenientes. Para facilitar la actualización del código, la edición de los recursos, y evitar instalar programas innecesarios en la tarjeta de desarrollo, se establece una conexión TFTP usando el protocolo SSH. Para ello se instala el cliente y servidor de SSH, en la tarjeta y en el ordenador (permitiendo comunicación bidireccional). Así puede accederse al sistema de archivos de la tarjeta de desarrollo desde el explorador Nautilus en el ordenador portátil.

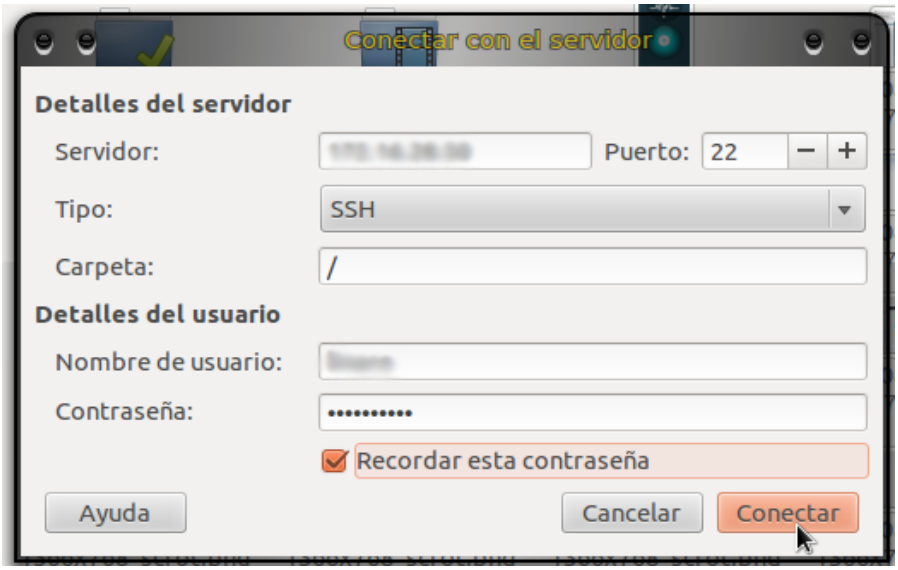

Figura 40. Ventana de conexión TFTP de Nautilus

Desde Nautilus, se hace *login* en la tarjeta usando su IP, y se accede al sistema de archivos según los permisos de la cuenta con la que se haga *login*. En este caso la cuenta posee permisos de administrador por lo cual se puede acceder directamente a la raíz (*/*).

Desde Anjuta IDE, se prepara el proyecto para ser movido a la tarjeta, para lo cual se genera desde el IDE un bzip, incluyendo el código fuente y los scripts de configuración de las *autotools* (herramientas de compilación de GNU). Es necesario mover por separado los recursos (estos no son incluidos en el bzip). Después de crear un directorio con todos los archivos necesarios para la compilación de la aplicación, se usa *autotools* para compilar (debe ejecutarse en consola desde el directorio donde están las *autotools* del proyecto:

make && make install

Sin embargo, puede que se haya olvidado instalar alguna dependencia, lo cual resultara en un mensaje de error diciendo que falta algún archivo de cabecera (.h), y se soluciona instalando la dependencia respectiva así como se hizo en el ordenador portátil(x86) en la sección 4.1.1.9.

Después de compilar correctamente, se ejecuta la aplicación. Debe ejecutarse con permisos de superusuario para que así pueda acceder a los puertos seriales. La ventaja de usar *autotools* es que la aplicación queda instalada en el directorio */bin/* del usuario, permitiendo su ejecución solo escribiendo el nombre en consola.

Usando la conexión establecida por TFTP se editan los archivos en la tarjeta de desarrollo desde Anjuta IDE, continuando con la integración de los algoritmos de decodificación en la aplicación. Siguiendo el modelo de la implementación del algoritmo de decodificación de la tarjeta Mindray, se integran también los algoritmos restantes y se compila la aplicación. El cambio de plataforma (de x86 a ARM) y la inclusión de los módulos OEM restantes, hacen que algunos parámetros configurados (como el periodo de ejecución de las funciones llamadas por *g_timeout_add()* y el número de datos leídos del puerto) necesiten ser reajustados. La decodificación pasa a realizarse en anillo (ver figura 41) para así, mientras se realiza la decodificación de la trama de datos de una tarjeta, las demás puedan acumular datos. A riesgo de hacer ilegible y poco eficiente la implementación de los algoritmos se optó por usar una característica poco recomendada: la instrucción *goto.* La información acerca de su uso en C advierte que su riesgo yace en su uso extendido por todo el código, que puede llevar a un programa inestable o poco legible. Sin embargo solo se usó para una porción muy pequeña.

Al cambiar de plataforma (de x86 a ARM) también se hicieron evidentes errores que no sucedían en el ordenador portátil. Algunas variables, eran muy pequeñas en relación a los datos que se les asignaban y por ello, sobrescribían algunas veces zonas adyacentes de memoria. La gran capacidad de memoria disponible en el ordenador (en comparación a la tarjeta de desarrollo), permitió muchas veces la ejecución normal del programa. El error en tiempo de ejecución se detectó gracias a los mensajes que cualquier programa compilado en modo *debug*, son impresas por GCC, (*stack smashing*), resultado de escribir en espacios de memoria destinados para otras tareas. Fue posible detectar las líneas de código problemáticas usando el GNU *Debugger* (GDB).

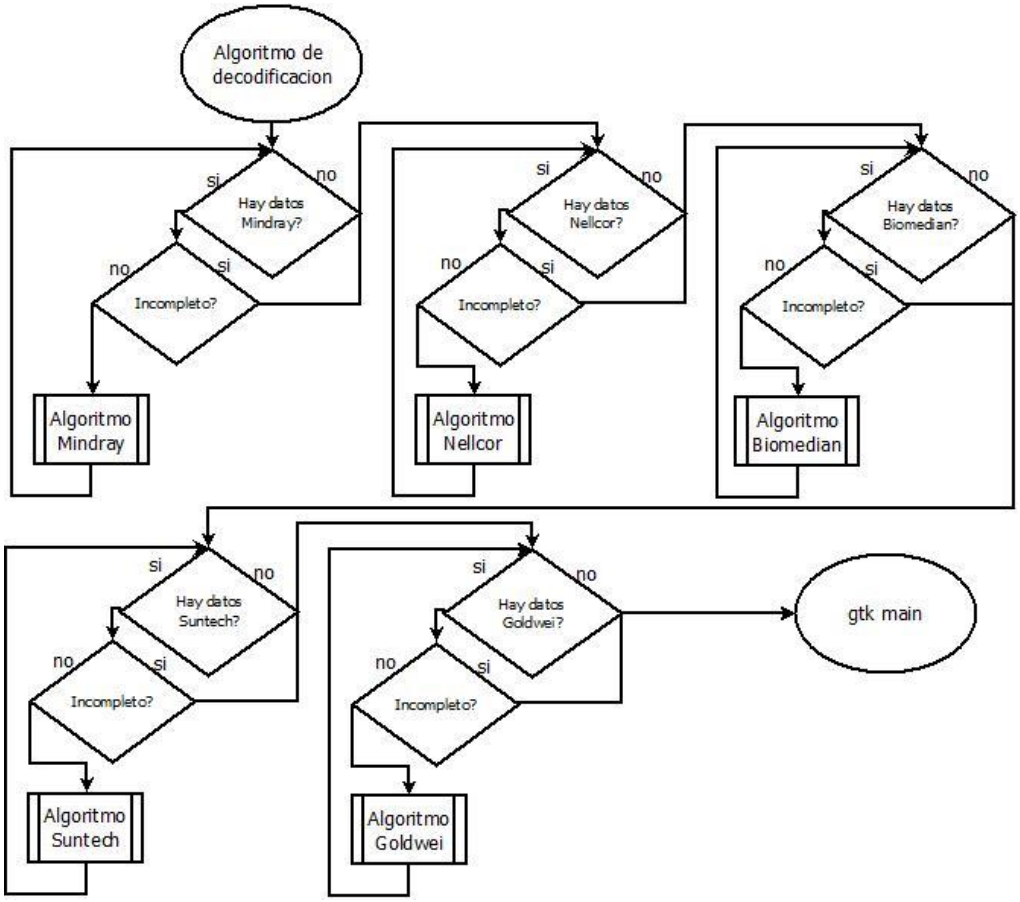

Figura 41. Implementación final del algoritmo de decodificación

4.1.1.17 Simulación de parámetros y verificación del funcionamiento de la GUI

Para comprobar la respuesta de la GUI a un flujo de datos similar al de un paciente se usaron los siguientes simuladores médicos:

Figura 42. Simulador Simcube

SimCube: Simulador de ECG, RESP, y PNI. Permite cambiar solo entre configuraciones definidas por el fabricante. Su simulación de PNI es diferencial por lo cual se debe colocar otro origen de presión, como un objeto solido rodeado con un brazalete neumático. Su precisión le permite ser utilizado para calibración de dispositivos médicos. Fue utilizado principalmente por su función de simular RESP y PNI ya que se posee otro simulador de ECG un poco más avanzado.

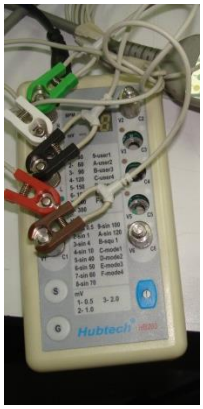

Figura 43. Simulador Hubtech

HubTech: Simulador de electrocardiografía (ECG). Permite enviar diferentes ondas, con características estáticas, variables o aleatorias, e incluso señales no cardiacas. Su ganancia es ajustable.

Figura 44. Simulador Nellcor

Nellcor: Simulador de pulsioximetría (SPO2), todos sus parámetros pueden variar solo entre dos valores, entre ellos frecuencia cardiaca, frecuencia de la onda, porcentaje de oxidación y nivel de la onda.

Figura 45. Esfigmomanómetro

Esfigmomanómetro: Originalmente utilizado para tomar PNI, se conectó a un sensor de presión, con el objetivo de ser utilizado para medir PI. Debido a que solo posee un canal, fue necesario simular los dos canales de PI de la GUI por separado. Solo permite cambiar la presión.

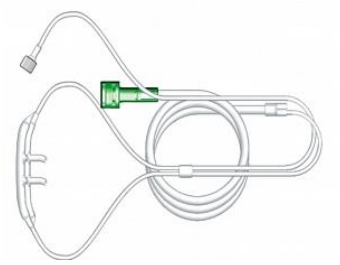

Figura 46. Cánula nasal [30]

Cánula nasal: No es un simulador. Aun así permitió la medición del parámetro CO2 de una persona por parte de la tarjeta OEM. La única variación controlable por parte del usuario fue la amplitud de la onda. La longitud de este dispositivo puede generar retrasos en la lectura de los valores por parte del dispositivo.

Figura 47. Resistencia 36°C

Resistencia: Resistencia ajustada por metrología a un valor interpretado en los sensores médicos como 36 grados Celsius. Está encerrada en un Jack de audio.

Gracias a los datos simulados se corrigieron detalles de las gráficas, se hizo la corrección del algoritmo *checksum* de algunas tarjetas (debido a datos que se perdían), se hicieron ajustes de ganancia, pruebas en selección de derivas y aún más importante, se probó la velocidad de respuesta del equipo a cambios en la señal.

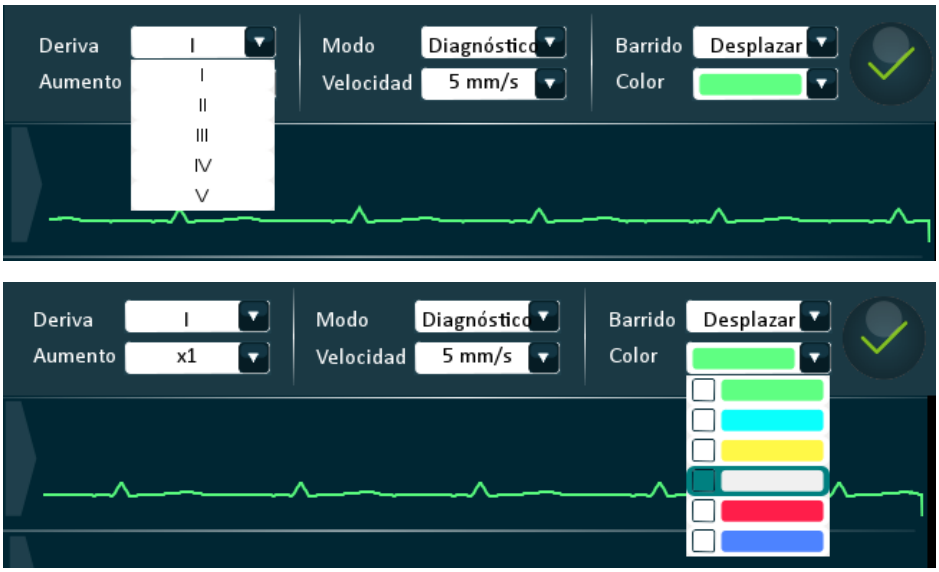

Figura 48. Menús secundarios de la GUI

Opciones disponibles en el menú desplegable de las gráficas. El ejemplo de la figura 48 corresponde a ECG, y puede seleccionarse cualquiera de las derivas especificadas y su color. Cuando es ejecutado por consola, se puede ver la verificación de las tarjetas conectadas, como en la figura 49.

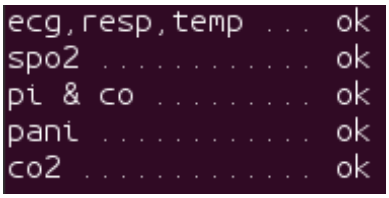

Figura 49. Verificación de las tarjetas por parte de la aplicación

Aplicación principal editada para agregar parámetros adicionales. Está configurada para mostrar dos derivas diferentes de ECG.

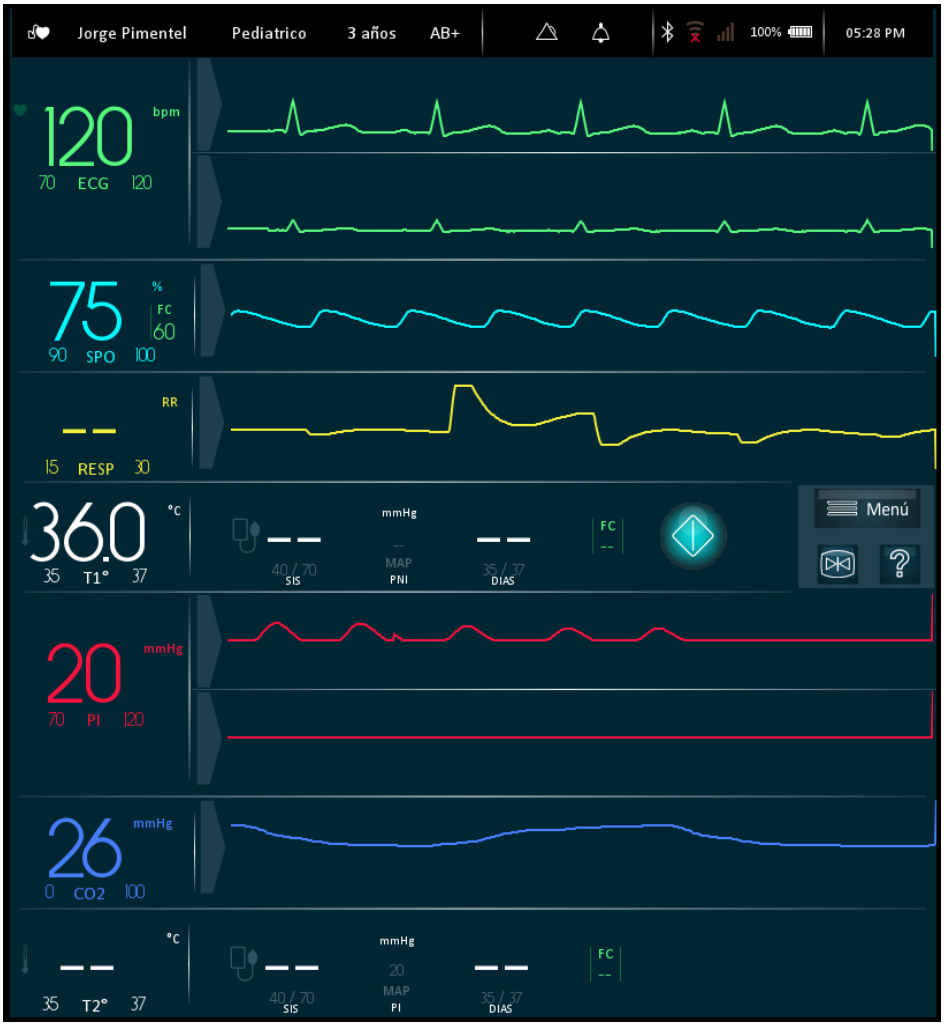

Figura 50. Aplicación final

5. APORTES AL CONOCIMIENTO

El departamento de Bioingeniería de la FCV, desde sus inicios ha impulsado el desarrollo de monitores de signos vitales. Inicialmente se implementaron usando ordenadores convencionales y una aplicación basada en Labview. El uso de este tipo de hardware conlleva varias desventajas entre las que se encuentran un tamaño excesivo e inestabilidad del sistema debido a temperaturas altas (este tipo de ordenadores no están diseñados para su funcionamiento continuo), lo cual reduce el tiempo de vida de los equipos y dificulta su mantenimiento. Esto sumado a la aplicación, que debido a la plataforma en la que está basada tiene un consumo alto de recursos, provocan no solo el desgaste de los componentes del equipo sino también su mal funcionamiento en un tiempo prolongado, exactamente una de las características que no debe tener un monitor de signos vitales.

Debido a esto, la FCV se ha encaminado en el uso de tecnologías más confiables para su próxima generación de monitores de signos vitales. Siendo conscientes de los requerimientos de estos equipos se seleccionaron dos posibles tecnologías que podrían ser de utilidad para su desarrollo: las FPGAs y los sistemas embebidos basados en ARM. Se inició entonces el desarrollo de un monitor de signos vitales basado en FPGAs, el cual se encuentra actualmente en etapas finales de desarrollo.

Buscando cubrir algunas necesidades adicionales de monitores de signos vitales en el sector de gama alta, se incursiono en los sistemas embebidos usando la tarjeta de desarrollo *Pandaboard*, buscando comprobar las capacidades del equipo para la tarea, retomando un enfoque mejorado de sus sistemas con ordenadores ya que a pesar de que se ejecuta un sistema operativo, tanto este como el código desarrollado para el mismo son extensamente utilizados actualmente en muchos campos, demostrando su fiabilidad y eficiencia respecto a sistemas anteriores.

Este desarrollo fue investigativo, ya que ningún miembro del equipo de Bioingeniería había sido asignado a la tarea de realizar una implementación semejante. El uso de sistemas embebidos ARM, pone a la FCV a la par del mercado actual de equipos médicos que en su mayoría también usan esta tecnología (esto hace que los componentes para implementarla sean más económicos). Se deja un sistema operativo preconfigurado sobre el cual otro ingeniero puede continuar trabajando e incluso puede portar a otros dispositivos basados en Linux.

De ser necesario continuar el desarrollo en esta tarjeta, muchas de sus capacidades también fueron probadas y documentadas. La interfaz gráfica puede trabajarse como base para el desarrollo del monitor de signos vitales orientado a necesidades de cuidado crítico.

Otro aporte importante es el algoritmo (*driver*) de decodificación de varias de las tarjetas OEM disponibles en la FCV, lo cual permite validar su funcionamiento, ahorrara tiempo de desarrollo y facilitara su inclusión en futuros desarrollos.

6 CONCLUSIONES

La versatilidad en su configuración tanto de software como de hardware, y la estabilidad frente al procesamiento de grandes flujos de datos del sistema operativo GNU/Linux, hacen que sea un candidato ideal para ejecutarse en un sistema embebido con aplicaciones médicas.

El uso de librerías software de comunicación con las tarjetas OEM crea una capa de abstracción que facilita su implementación en aplicaciones finales.

El soporte de varias arquitecturas por parte del sistema GNU/Linux y sus componentes permite cambiar de dispositivo en el transcurso del desarrollo sin que esto afecte su funcionamiento.

La portabilidad y amplio uso del lenguaje C permiten interactuar de forma más eficiente y directa, con librerías y componentes del sistema operativo GNU/Linux.

Es posible que librerías graficas libres como GTK no posean elementos específicos aplicables a ciertas tareas (como el caso de una GUI para un equipo médico), sin embargo sus componentes base generalmente si pueden ser ajustados a ellas si se posee un extenso conocimiento de la estructura de la librería.

Las razones por las que el procesador ARM se usa extensivamente en el mercado actual son apreciables: el desempeño del sistema desarrollado es más estable, rápido y maneja un mayor volumen de datos que el existente (PC convencional), con un menor consumo.

La implementación de la interfaz gráfica de usuario fue exitosa, y su funcionamiento fue validado por el equipo de Bioingeniería de la FCV.

La práctica desarrollada en la UEN Bioingeniería de la FCV permitió participar de forma activa en la industria de desarrollo de equipos electrónicos, así como su familiarización con los procesos y conductos regulares en una empresa, lo cual facilitara su integración en el mercado laboral.

El uso de tarjetas OEM reduce significativamente el tiempo y coste de desarrollo de un equipo final.

Los registros FIFO circulares son esenciales para obtener y procesar grandes volúmenes de datos.

Al igual que en una librería gráfica, la clave para lograr que una aplicación sea considerada por el usuario como funcional y concurrente es distribuir en pequeños intervalos de tiempo las tareas a realizar en ella.

El desarrollo de módulos hardware con tareas específicas con sus respectivos drivers, facilitan su integración en un proyecto basado en FPGAs

BIBLIOGRAFIA

[1] Organigrama FCV. [En línea] <http://www.fcv.org/corp/index.php?option=com_content&view=article&id=47&Itemid=57>. Consultado el 20 de diciembre de 2012.

[2] Arm architecture reference. [en línea]<https://silver.arm.com/download/download.tm?pv=1111932>. Consultado el 20 de diciembre de 2012.

[3] Arm processor architecture [en línea] <http://www.arm.com/products/processors/instruction-set-architectures/index.php.> Consultado el 20 de diciembre de 2012.

[4] Intel vs ARM [en línea] <http://www.techwatch.com.au/viewtopic.php?f=8&p=37150.> Consultado el 21 de diciembre de 2012.

[5] ARM SoC Block Diagram [en línea] <http://upload.wikimedia.org/wikipedia/commons/8/85/ARMSoCBlockDiagram.svg> Consultado el 21 de Diciembre de 2012.

[6] History of Linux [en línea] <http://digital-domain.net/lug/unix-linux-history.html>. Consultado el 21 de Diciembre de 2012.

[7] In Linux Everything is a file [en línea] <http://www.linux.org/article/view/in-linux-everything-is-a-file>. Consultado el 21 de Diciembre de 2012.

[8] Linux in Government [en línea] <http://www.linuxjournal.com/article/8449>. Consultado el 21 de Diciembre de 2012.

[9] What is GTK? [en línea] <http://developer.gnome.org/gtk-faq/stable/x81.html> Consultado el 21 de diciembre de 2012

[10] Features [en línea] <http://www.gtk.org/images/features/twf.png>. Consultado el 21 de Diciembre de 2012

[11] Pandaboard.org, Panda es B manual [en linea] <http://pandaboard.org/sites/default/files/board_reference/pandaboard-es-b/panda-es-b-manual.pdf>. Consultado el 15 de Agosto de 2012.

[12] OMAP 3 Processors. [en línea] <http://www.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?contentId=14649&navigationId=12643&templateId=6123>. Consultado el 23 de Diciembre de 2012.

[13] OMAP-DM Processors [en línea] <http://www.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=12801&contentId=41248>. Consultado el 23 de Diciembre de 2012.

[14] OMAP Mobile Processors [en línea] <http://www.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=11999&contentId=4683>. Consultado el 23 de Diciembre de 2012

[15] SGX IP Core Family [en línea] <http://www.imgtec.com/powervr/sgx.asp>. Consultado el 23 de Diciembre de 2012.

[16] Logic Element [en línea] <http://wwwdsa.uqac.ca/~daudet/Cours/Vlsi/DOCUMENTS/repertoire435/Livre-MJS-Smith/Book/CH05/CH05-6.gif>. Consultado el 24 de diciembre de 2012.

[17] The LDD/PDD Model [en línea] <http://www.symlab.org/main/documentation/reference/s3/pdk/GUID-EBF4F1F1-F76B-455B-B8EE-B7965CF0717E.html>. 24 de Diciembre de 2012.

[18] Logical Device Drivers [en línea] <http://www.freescale.com/infocenter/index.jsp?topic=%2Fcom.freescale.doc.mcu.processorexpert.usersmanual%2Fhtml%2FLDDcomponents.html>. Consultado el 24 de Diciembre de 2012.

[19] Virtual Device Driver [en línea] <http://www.solarwinds.com/it-management-glossary/what-is-virtual-device-driver.aspx>. Consultado el 24 de Diciembre de 2012

[20] OEM Developer's Manual [doc]. Pag 32, Figura C. Consultado el 20 de diciembre de 2012.

[21] GCC [en línea] <http://upload.wikimedia.org/wikibooks/en/0/0b/Gcc.JPG>. Consultado el 21 de Diciembre de 2012

[22] CAIRO API tutorial [en línea] <http://cairographics.org/tutorial/> . Consultado el 20 de agosto de 2012.

[23] Fluxbox [en línea] <https://wiki.archlinux.org/index.php/Fluxbox>. Consultado el 1 de Septiembre de 2012.

[24] Bootloader [en línea] <http://searchenterpriselinux.techtarget.com/definition/boot-loader>. Consultado el 25 de Diciembre de 2012.

[25] TTY [en línea] <http://www.linusakesson.net/programming/tty/>. Consultado el 20 de Diciembre de 2012.

[26] Word configuration [en línea] <http://www.blaess.fr/christophe/2012/06/04/gpio-pandaboard-et-temps-reel-5-le-multiplexage-des-gpio/>. Consultado el 5 de Septiembre de 2012.

[27] Host Interface Specification [pdf]. Consultado el 10 de Septiembre de 2012.

[28] Event based programming [en línea] <http://www.willamette.edu/~gorr/classes/cs231/lectures/chapter5/events.gif>. Consultado el 10 de Septiembre de 2012.

[29] FIFO as circular buffer [en línea]. <http://wiki.linuxencaja.net/images/e/e7/Buffer_circular.png>. Consultado el 15 de Septiembre de 2012.

[30] Nasal cannula [en línea] <https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcTmo.jpg>. Consultado el 25 de Septiembre de 2012.

[31] OMAP44x Chip Diagram [en línea] http://www.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=12843&contentId=53243#chipDiagram> Consultado el 27 de Diciembre de 2012.

ANEXOS

Como tareas adicionales, paralelo al desarrollo del objetivo principal de esta práctica se desarrollaron:

- Modulo hardware en Verilog HDL encargado de sonidos y alarmas

Usando los datos adquiridos de monitores de signos comerciales, se diseñó la primera versión de un módulo de alarmas con sonidos similares a ellos, buscando su fácil identificación por parte del usuario del dispositivo. Esta primera versión realiza la síntesis de los sonidos usando una onda cuadrada con frecuencia variable.

Figura 51. Versión inicial del módulo de alarmas

En el proceso de validación de las capacidades del módulo, se decide cambiar la síntesis en base a onda cuadrada por la reproducción de muestras de sonidos grabadas en una memoria flash, esto para mejorar la calidad de los sonidos y reducir a su vez el número de elementos lógicos requeridos por el modulo.

Figura 52. Versión final del módulo de alarmas

Para ello se tomaron los sonidos obtenidos inicialmente, y mediante un software de audio se replicó su frecuencia fundamental, considerando los intervalos de tiempo de cada sonido. Para ello se usó el software Audacity. Las señales se basaron en ondas Seno excepto por 3 sonidos que se buscaron en la red y se incluyeron directamente.

El sonido final fue guardado en formato WAV, y se buscó que tuviera un tamaño reducido (menor a 2MB), para lo cual fue necesario reducir un poco la calidad de los sonidos. Finalmente se generó un archivo binario con todos los sonidos.

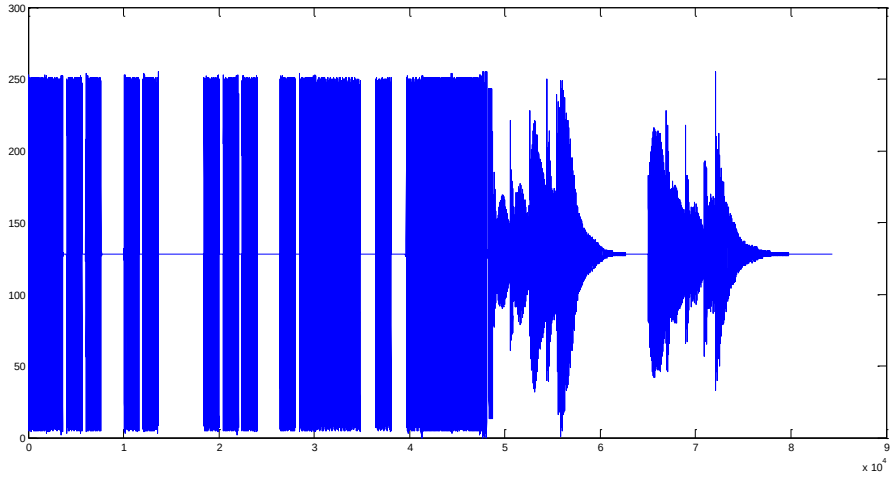

Figura 53. Muestras de sonido del módulo de alarmas

Este archivo binario se copió a la memoria flash del dispositivo, usando el *flash programmer* del procesador Nios II.

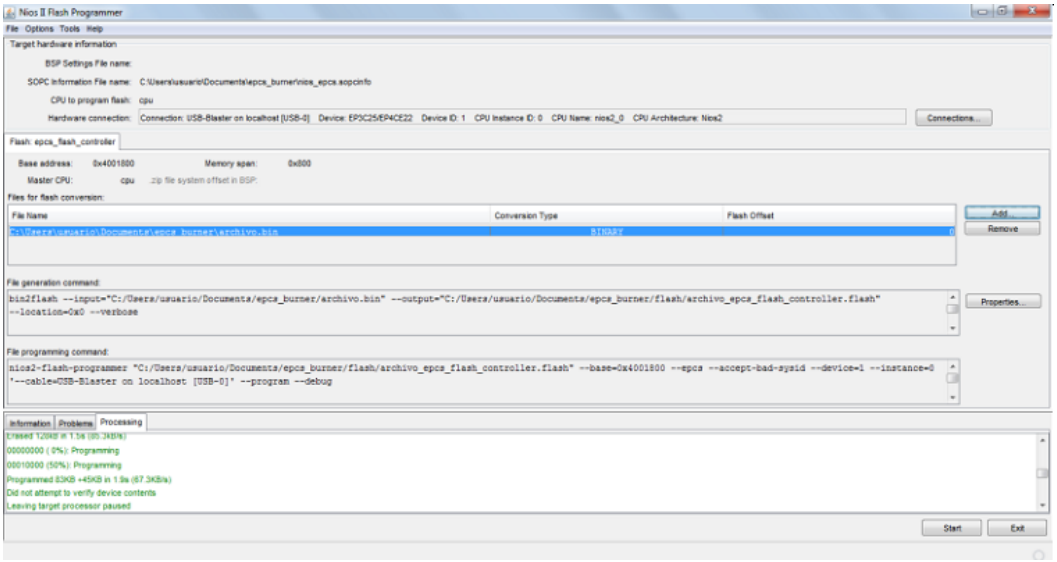

Figura 54. Nios II flash programmer

El modulo se editó para seccionar la memoria *flash*, leyendo para cada comando una sección de la memoria diferente, de forma repetitiva o única, según el comando que se le haya enviado.

- Driver en lenguaje C para procesador Nios II del módulo de sonidos y alarmas

Se diseñó un driver en lenguaje C para el procesador de Nios II que envía comandos y configura el módulo de alarmas. Para ello se usa el puerto *cmd_clk*, *cmd* y *data*, los cuales se encuentran conectados a GPIOs del procesador dedicadas a manejar el modulo. *Data* puede contener uno o más parámetros de configuración según el *cmd* enviado, y los datos son recibidos por el modulo en el flanco positivo de *cmd_clk*. Las funciones del driver se muestran en la figura 55.

```
74  void alarm_cmd_rdy();
75  void write_to_ports(short data,char cmd);
76  void alarm_volume_set(char type,short alarm,char vol);
77  void alarm_state_set(char alarm,char value,char dead);
78  void alarm_custom_config(short freq,short time_on_ms,short time_off_ms);
79  void alarm_wave_set(char alarm,char wave);
80  void alarm_type_set(char type);
81  void tone_freq_set(short freq);
82  void tone_vol_set(short vol);
83  void tone_wave_set(char wave);
84  void beep_saturation_perc(short perc,char value);
85  void beep_vol(short vol);
86  void h_beat();
87  void beep_bt();
88  void beep_bt_state(char data);
89  void beep_bt_vol(char vol);
90  void beep_enable(char enable);
91  void on_sound();
92  void off_sound();
93  void alarm_init();
94  #endif /*ALARM_DRIVER_H_*/
95
```

Figura 55. Funciones del driver para el módulo de alarmas

- Driver en lenguaje C del RTC(Real Time Clock) y Acelerometro (*onboard*) para procesador Nios II

  Se diseñó un driver en lenguaje C para el procesador Nios II usando protocolos I2C y SPI, para comunicarse con el RTC y acelerómetro.

  El driver del RTC tiene la capacidad de configurar y obtener datos de fecha y hora, de convertirlos y regresarlos como una estructura de fecha al programa principal, en formato 12 o 24 H. Sus funciones y estructuras se muestran en la figura 56.

```
typedef struct calendario_struct{
    unsigned char  year;
    unsigned char  month;
    unsigned char  day;
    unsigned char  date;
    unsigned char  hour;
    unsigned char  minutes;
    unsigned char  second;
    unsigned char  am_pm;
    unsigned char  fmt;
}FECHA;
void DEMO_ACCELEROMETER(void);
void rtc_time_get(FECHA* yymmddhhmmss);
//Funcion : obtener los valores de fecha del rtc, en una estructura de tipo FECHA
//Retorno : 1 si la lectura fue satisfactoria, de lo contrario 0
int rtc_time_set(FECHA* yymmddhhmmss);
//Funcion : establecer los valores de fecha del rtc, usando una estructura de tipo FECHA
//Retorno : 1 si la escritura fue satisfactoria , de lo contrario 0
int rtc_format_set(unsigned char format);
//Funcion : establecer el formato de hora del rtc, puede ser F24H o F12H
//Retorno : 1 si se establecio el formato correctamente, de lo contrario 0
int rtc_ram_write(unsigned char szBuf[],unsigned char from_dir,unsigned char to_dir);
//Funcion : escribir valores en la ram del rtc, usando un vector
//Retorno : 1 si la escritura fue satisfactoria , de lo contrario 0
int rtc_ram_read(unsigned char szBuf[],unsigned char from_dir,unsigned char to_dir);
//Funcion : leer valores desde la ram del rtc, usando un vector
//Retorno : 1 si la lectura fue satisfactoria , de lo contrario 0
int bcd2dec(char num);
//Funcion complementaria, convierte de bcd a decimal, regresa el valor en decimal
int dec2bcd(char num);
//Funcion complementaria, convierte de decimal a bcd, regresa el valor en bcd
int sizeofstring(unsigned char num[]);
//Funcion complementaria, halla el tamaño de un string terminado en NULL ('/0')
int rtc_osf_get();
int rtc_osf_clr();
void rtc_clk_halt(unsigned char value);
#endif /*RTC_DRIVER_H_*/
```

Figura 56. Funciones del driver para el RTC

El driver del acelerómetro obtiene la aceleración ejercida sobre la tarjeta en sus tres ejes y permite regresar su valor tanto en forma de aceleración lineal (x , y ,z) como en forma de aceleración angular (pitch, roll). Sus funciones y estructuras se muestran en la figura 57.

```
#ifndef ACEL_DRIVER_H_
#define ACEL_DRIVER_H_
#include "system.h"
#include "math.h"
#define PI 3.141592
//#define WRITE_ALARM_DATA(x) IOWR_ALTERA_AVALON_PIO_DATA(ALARM_DATA_BASE,x)
typedef struct acel_data_struct{
    int  x;
    int  y;
    int  z;
    float pitch;
    float roll;
}ACEL;
float rad2deg(float num);
//Funcion complementaria, convierte de radianes a grados, regresa el valor en grados
void acel_init();
int acel_get(ACEL* xyzpr);
#endif /*ACEL_DRIVER_H_*/
```

Figura 57. Funciones del driver para el Acelerómetro

GLOSARIO

AAMI	Association for the advancement of medical instrumentation
ADC	Analog – Digital Converter
API	Application Programming Interface
ARM	Advanced Risc Microcontrollers
ASIC	Application Specific Integrated Circuit
CO2	Dioxido de carbono (Capnografia)
CRC	Cyclic redundancy check
DAC	Digital – Analog Converter
DSP	Digitial Signal Processor
ECG	Electrocardiografia
FCV	Fundacion Cardiovascular
FIFO	First input first output
FPGA	Field programable gate array
FTDI	Future technology devices international
GCC	Gnu compiler collection
GDB	Gnu debugger
GIB	Gibibyte
GNU	GNU is not Unix (acrónimo recursivo)
GPIO	General purpose input output
GPU	Graphics processing unit
GTK	Gimp toolkit
GUI	Graphical user interface
HDL	Hardware description language
I2C	Inter integrated circuit
IDE	Integrated development environment
IEEE	Institute of electrical and electronic engineers
LDD	Logical device driver
LUT	Look-up table
MUX	Multiplexor
OEM	Original equipment manufacturer
OMAP	Open multimedia applications platform
PDD	Physical device driver
PI	Presion invasiva
PLL	Phase-locked loop
PNI	Presion no invasiva
POST	Power-on self test
RGB	Red Green blue
RISC	Reduced instruction set computer
ROM	Read only memory
RS232	Recommended standard 232
RTC	Real-time clock

SDRAM	Sychronous dynamic Access memory
SOC	System on a chip
SOPC	System on a programable chip
SPI	Serial peripheral interface
SPO2	Pulsioximetria
SSH	Secure shell
TFTP	Trivial file transfer protocol
TTY	Teletipo
UART	Universal asynchronous receiver- transmitter
UEN	Unidad estratégica de negocios
USB	Universal serial bus
VDD	Virtual device driver