UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE ADQUISICIÓN DE DATOS Y GUI DE UN PDA PARA MANTENIMIENTO DE EQUIPOS MÉDICOS UTILIZANDO UN SISTEMA LINUX EMBEBIDO Y DSPICS.

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

JUAN CARLOS SESSAREGO DÍAZ

PROMOCIÓN

2008 - I

LIMA – PERÚ
2014

DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE ADQUISICIÓN DE DATOS Y GUI DE UN PDA PARA MANTENIMIENTO DE EQUIPOS MÉDICOS UTILIZANDO UN SISTEMA LINUX EMBEBIDO Y DSPICS.

A Mónica quien sin su apoyo, comprensión y consejos no hubiese sido posible realizar la presente tesis.

SUMARIO

En el presente trabajo de tesis se desarrolla el sistema de adquisición de datos de un asistente digital personal para el mantenimiento de equipos médicos en hospitales como su interfaz gráfica de usuario. El equipo tomará datos desde instrumentos de medición conectados a este a través del sistema de adquisición diseñado.

La interfaz gráfica de usuario será visible desde un LCD 7 pulgadas en la que se mostrarán los programas gráficos de todos los instrumentos de medición, ordenes de trabajo de mantenimiento, aplicaciones, utilitarios, etc.

Para poder diseñar dicho sistema se usará como herramientas de desarrollo principales el sistema embebido chino mini2440 y microcontroladores dspic de microchip.

Los temas en esta tesis involucrarán la programación de controladores de dispositivos en linux, manejo de los recursos del sistema embebido, programación en C de los dspics, manejo de la plataforma de aplicación integral Qtopia la cual proporcionará el entorno gráfico de desarrollo y el uso del gestor de base de datos sqlite para enviar y recoger la información de mantenimiento desde y hacia el sistema embebido.

ÍNDICE

PRÓLOGO

En la actualidad el avance de la tecnología se está dando rápidamente proporcionándonos herramientas para poder afrontar cualquier problema al desarrollar un proyecto. La presente tesis está dirigida a aquellas personas interesadas en el manejo de sistemas Linux Embebido, Interfaz Gráfica de Usuario (GUI) y microcontroladores los cuales serán empleados para elaborar un Asistente Digital Personal (PDA) el cual tendrá como propósito mejorar la calidad del mantenimiento de equipos en hospitales.

La tesis será dividida en seis capítulos:

Capítulo 1: "Generalidades", se presentará la problemática, propuesta de solución y el alcance del trabajo.

Capítulo 2: "Marco teórico", se presentarán todos los conceptos teóricos necesarios utilizados en la tesis, estos abarcan conceptos relacionados al manejo de sistemas linux embebido, microntroladores, Interfaz Gráfica de Usuario, Norma RS485 etc.

Capítulo 3: "Diseño del sistema", se presentará el sistema de adquisición de datos e Interfaz Gráfica de Usuario del Asistente Digital Personal para el mantenimiento del los equipos médicos, además se presentará las características del sistema embebido escogido para el sistema diseñado.

Capítulo 4: "Protocolo de comunicación", se explicará el protocolo de comunicación elaborado para la transferencia de información desde los instrumentos de medición al sistema embebido y viceversa.

Capítulo 5: "Diseño de Aplicaciones Gráficas", se explicará los programas y librerías gráficas usadas para el diseño de la Interfaz Gráfica de Usuario, además del uso del gestor de base de datos para la construcción de las ordenes de trabajo de mantenimiento en el sitema linux embebido.

Capítulo 6: "Comunicación del Sistema Embebido e Instrumentos", se analizará el software involucrado en la transferencia de la información basada en el protocolo de comunicaciones presentando todos los programas relacionados al sistema de adqusición de datos tanto del sistema linux embebido como de los instrumentos.

Capítulo 7: “Evaluación de Costos”, se explicará el costo de producción de nuestro sistema propuesto. Además se darán ciertas alternativas para poder reducir el costo de producción.

Agradezco a la Empresa Tumimed SRL por el apoyo brindado durante el desarrollo de la presente tesis.

CAPÍTULO I
GENERALIDADES

1.1 Problemática

En la actualidad no todos los equipos en hospitales públicos tiene protocolos de mantenimiento que son formularios donde se especifican los pasos a seguir por parte de los técnicos para comprobar el correcto funcionamiento de un equipo. Para la mayoría de los equipos se utilizan Ordenes de Trabajo de Mantenimiento (OTM) que a diferencia de los protocolos de mantenimiento son formularios con una serie de ítems relacionados más a un aspecto administrativo que al propio mantenimiento. Dichas OTM especifican el nombre, modelo, marca, serie, ubicación, etiqueta patrimonial, fecha de inicio, fecha de término, tipo de OTM, ejecutor, mano de obra, etc.

El ítem número 6 de la OTM (Datos generales de la ejecución) es el campo donde los técnicos en un máximo de cinco líneas describen todas las actividades que han realizado en el mantenimiento del equipo médico.

Al no existir una estandarización en el desarrollo de los pasos a seguir para el mantenimiento de cada equipo médico en el ítem número 6 de la OTM genera que el mantenimiento dependa en gran medida de la experiencia del técnico como del ingeniero supervisor lo que hace que este se vuelva bastante complejo y difícil para ciertos equipos médicos.

Muchas veces las consideraciones del técnico no coinciden con la del ingeniero supervisor tomándose más tiempo de lo previsto en el mantenimiento generando un retraso en la toma de decisiones.

Como las OTMs son formularios que se llenan a mano por parte de los técnicos, estas posteriormente deberán ser ingresadas por las secretarias a la base de datos del hospital que lleva el registro de todos los equipos de tal forma que mientras no se haya actualizado dicha base de datos no podrá utilizarse el equipo generando una pérdida de tiempo innecesaria. El mismo trámite en que la OTM pasa del ingeniero supervisor, al técnico y por último a las secretarias o que las propias OTM no son legibles generan una

serie de retrasos.

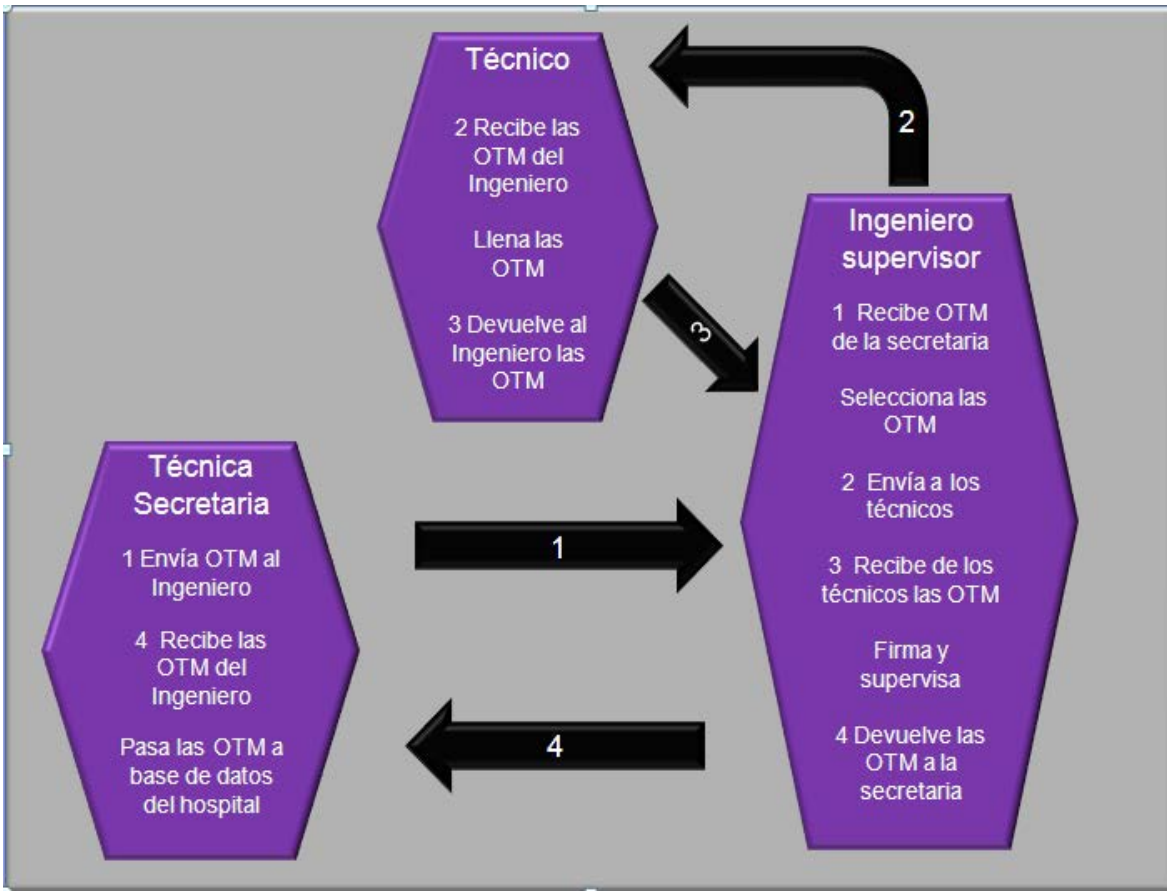


Figura 1.1 Ciclo de la OTM (Fuente Propia)

1.2 Propuesta de solución

Para poder solucionar la problemática mencionada se diseñará e implementará las herramientas de un equipo PDA en base a un sistema Linux embebido que a través de su GUI permita la generación de OTM personalizadas para cada equipo médico las cuales tendrán todos los pasos a seguir recomendados por el fabricante del equipo médico de tal forma que se logre una estandarización en el mantenimiento por equipo.

El equipo PDA tendrá la capacidad de conectarle simultáneamente hasta cuatro instrumentos de medición (elaborados en base a microcontroladores) utilizados por los técnicos para poder llenar ciertos campos de las OTM personalizadas, además recibirá la información de los instrumentos de medición a través de una interfaz paralela controlado por un driver.

El equipo PDA proporcionará una serie de utilidades como archivos de ayuda para los equipos médicos, cámara, calendario, videos, fotos, etc. lo que permitirá hacer más fácil y rápido el mantenimiento.

Una vez que el técnico llene las OTM personalizadas usando el teclado y mouse del equipo

PDA la información del mantenimiento será almacenada en una base de datos siendo posteriormente descargada a la PC del hospital vía USB para su actualización con lo que se logrará agilizar el proceso de mantenimiento.

Nota:

No es motivo de la presente tesis la elaboración de la electrónica necesaria para la implementación de la parte de sensado de los instrumentos de medición, esto quedará a cargo de un grupo de ingenieros , más bien se centra en que todos los instrumentos tendrán un microcontrolador el cual será encargado de enviar la información a través del bus 485 por lo que esta tesis pone énfasis en la transferencia de la información, sin embargo muy pocos instrumentos solo constarán de un microcontrolador el cual recibirá información desde instrumentos de medición ya elaborados los cuales enviarán los datos sensados vía rs232 u otro protocolo.

Tampoco es motivo de la presente tesis los programas necesarios en la PC del hospital que deberán actualizar su base de datos cuando reciban la base de datos sqlite generado por el equipo PDA una vez que los técnicos hayan llenado las OTMs

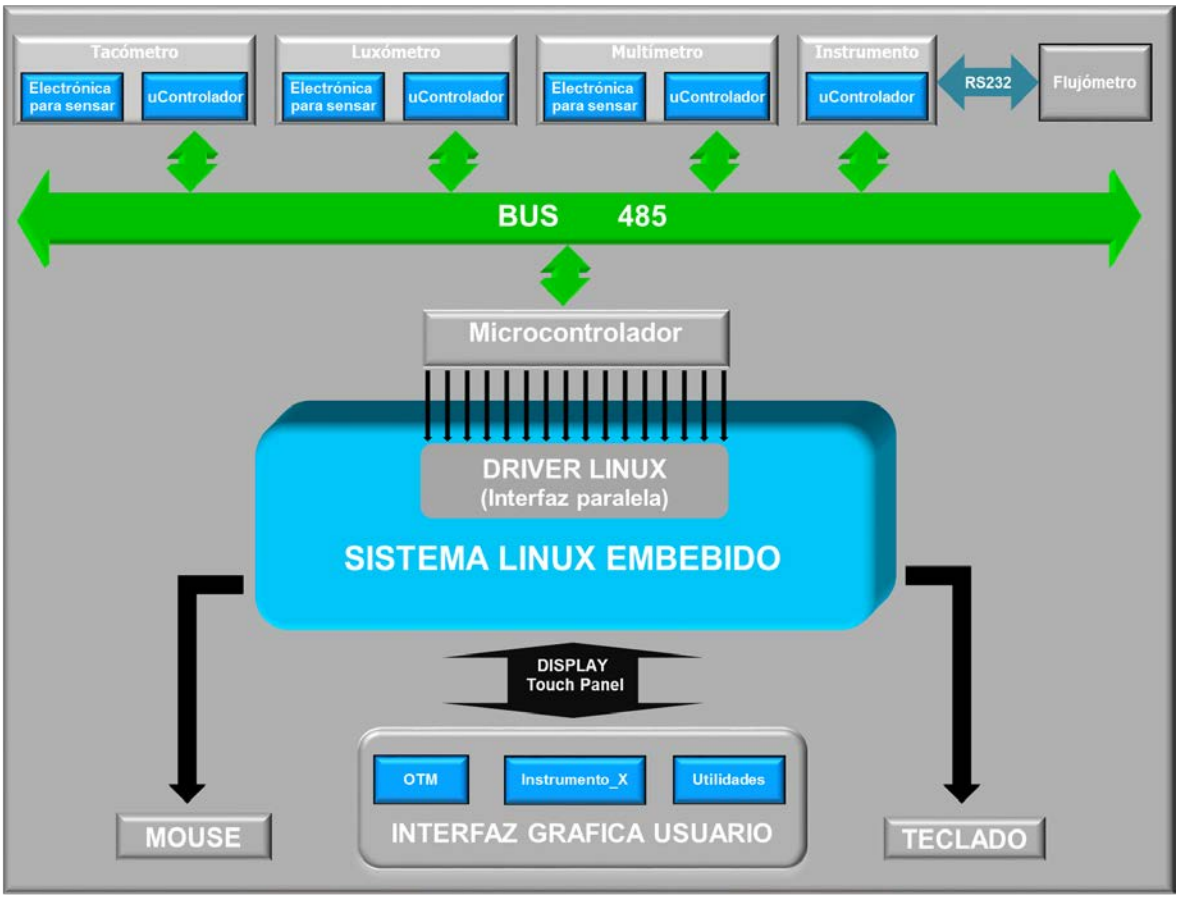


Figura 1.2 Diagrama general del Sistema (Fuente Propia)

personalizadas, esto quedará a cargo de otro grupo de ingenieros.

1.3 Objetivos

Desarrollar el sistema de adquisición de datos de un PDA para el mantenimiento de equipos médicos el cual consiste en la implementación a nivel de hardware y software de una interfaz paralela de 16 bits para la tarjeta Linux embebida mini2440 usando el bus de datos del sistema embebido la que se comunicará paralelamente a un dspic concentrador al cual se conectarán a través del estándar RS-485 hasta un máximo de cuatro instrumentos de medición que tendrán un dspic esclavo cuyo objetivo será almacenar la información sensada de los equipos médicos y enviarlas al dspic concentrador el cual enviará dicha información a la mini2440 a través de la interfaz paralela.

Modificar, compilar, instalar y personalizar la plataforma Qtopia en nuestro sistema Linux embebido mini2440 para poder correr aplicaciones gráficas usando las librerías Qt.

Desarrollar un programa modelo usando las librerías Qt que se tomará como referencia para poder elaborar cualquier orden de trabajo de mantenimiento personalizada para equipos médicos las cuales serán descargadas hacia el PDA desde una PC principal del hospital y una vez que el técnico llene las ordenes de mantenimiento serán enviadas a la PC principal para actualizar la base de datos.

Desarrollar un programa modelo usando las librerías Qt que se tomará como referencia para visualizar los datos numéricos, tendencias y curvas en el tiempo si fuese el caso de los instrumentos de medición conectados al PDA tales como voltímetros, tacómetros, flujómetro , luxómetro , manómetro, etc.

ALCANCES

- Presentar los fundamentos para poder diseñar drivers para sistemas embebidos sobre Linux para poder agregar un hardware particular a nuestras necesidades.
- Comprender el manejo y uso de Qtopia en su versión PDA (Asistente Digital Personal) como plataforma para ejecución de aplicaciones Qt.
- Introducirnos al diseño de aplicaciones gráficas usando las librerías Qt.
- Manejo de los recursos proporcionados por el sistema operativo Linux tales como memoria compartida, señales, etc. para comunicar y sincronizar procesos.
- Manejo de los recursos proporcionados por dspic como timers, interrupciones, pll , etc. además del uso del estándar RS-485.

CAPÍTULO II
MARCO TEÓRICO

2.1 Sistemas Linux Embebido

Los sistemas embebidos son dispositivos de propósito específico que han sido diseñados para realizar un conjunto reducido de operaciones. En contraste, una computadora de propósito general como las computadoras personales pueden realizar una gran cantidad de tareas dependiendo de los programas que ejecuten.
Las características de un sistema embebido son:

- Costo reducido.
- Bajo consumo eléctrico.
- La mayoría poseen restricciones de tiempo-real.
- Existen múltiples arquitecturas de procesadores (MIPS, ARM, PowerPC, etc.) cuyas características varían según la aplicación específica del sistema embebido (procesamiento de imágenes, transmisión de datos, etc.)
- Poseen recursos limitados de memoria RAM, ROM u otros dispositivos de Entrada / Salida (E/S) en comparación con una computadora tipo PC.
- Un sistema embebido es diseñado desde dos perspectivas, hardware y software, teniendo en cuenta su aplicación específica.

Este tipo de dispositivos son utilizados en diferentes áreas; frecuentemente en equipamiento de redes de datos como routers, firewalls, switchs; en sistemas de comunicaciones como teléfonos celulares, PDAs, cámaras digitales; o para propósitos hogareños como reproductores de MP3, sistemas de entretenimiento, etc.

Esta gran variedad de dispositivos realizan un conjunto de operaciones limitadas en función de las características que posee el software embebido que ha sido desarrollado por el fabricante. Específicamente en el área de redes, estos dispositivos son vistos como una caja negra que ofrecen funcionalidades acotadas, restringiendo al usuario la posibilidad de incrementar o personalizar la capacidad del mismo.

La gran mayoría de los sistemas embebidos poseen algún tipo de procesador integrado,

también denominados SOC (System On Chip). A diferencia de aquellos procesadores cuya función es solo la de procesamiento, denominados autónomos (Stand-Alone), los procesadores SOC poseen integrados controladores DRAM, UART, PCI, Ethernet, etc.

La arquitectura de procesadores autónomos **(1)** como se muestra en la figura **2.1** son utilizados en computadoras de propósito general y son diferentes de los procesadores embebidos.

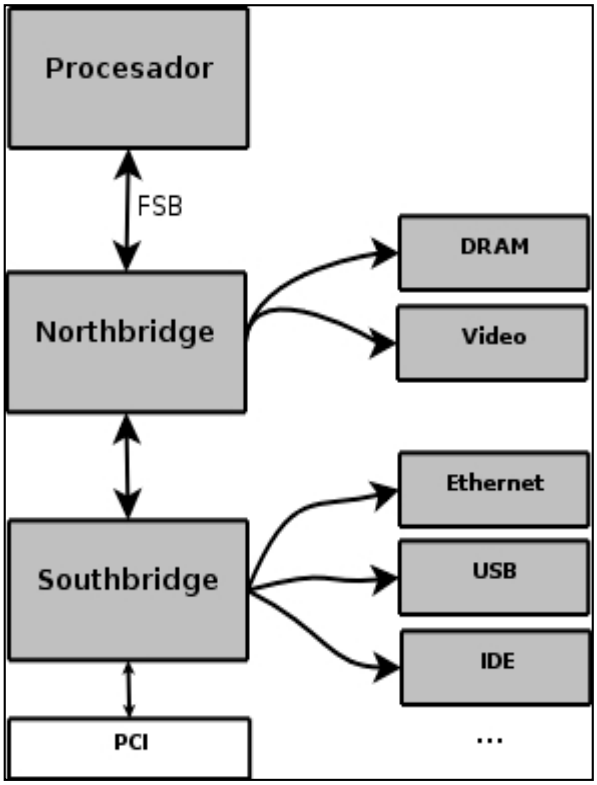


Figura 2.1 Arquitectura Procesadores autónomos (Fuente (1))

Estos procesadores poseen componentes extras (chipsets) para satisfacer los requerimientos de conexión y permitir dispositivos periféricos externos como el sistema de memoria principal, (DRAM), ROM, flash, sistema de buses PCI, puertos seriales, interfaces IDE, etc. En la figura **2.2** observamos la anatomía general de un sistema embebido, en donde el procesador posee integrado una UART (Universal Asynchronous Receiver-Transmitter) para una interfaz serial y un controlador ethernet, además de memorias (RAM y NAND) y un bus para la conexión de una memoria compact flash.

Existen cientos de marcas en el mercado que fabrican soluciones SOCs de arquitectura RISC (Reduced Instruction Set Computer). Algunos ejemplos de estas arquitecturas soportadas por Linux son:

PowerPC, utilizado en sistemas embebidos para redes y telecomunicaciones.

ARM, generalmente utilizados en teléfonos celulares y equipamiento de redes.

MIPS, arquitectura con implementaciones de 32 y 64 bits utilizado en una gran cantidad de productos populares, televisores Sony de alta definición, access points inalámbricos Linksys y la popular consola de video juegos Sony Play Station 2.

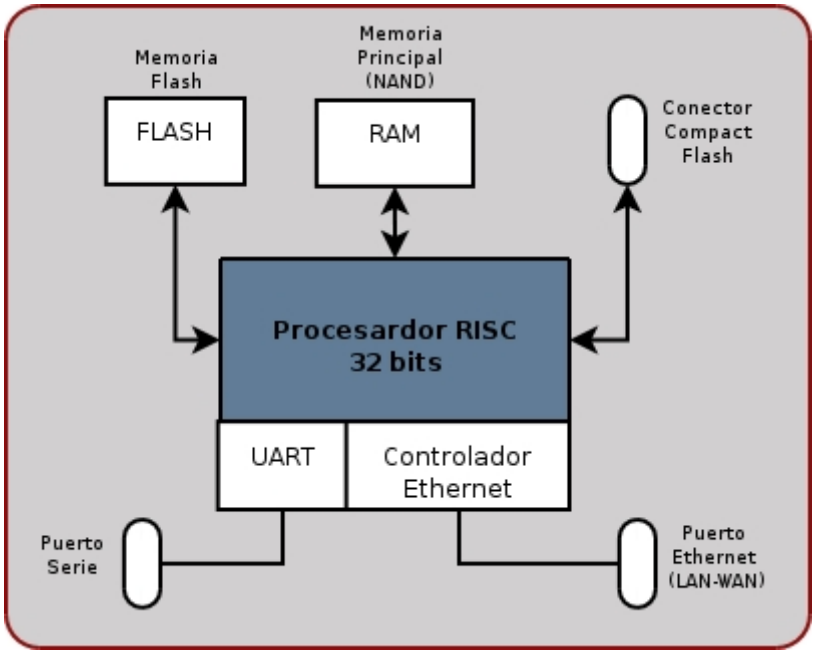


Figura 2.2 Diagrama general Sistema Embebido (Fuente (1))

Una de las ventajas de Linux como sistema operativo embebido es el rápido soporte de nuevos chipsets, sin embargo, en el mercado de sistemas embebidos Linux, estas tres arquitecturas son las más importantes y utilizadas, prueba de esto es la encuesta realizada por el portal de Linux embebido Linuxdevices.com como se muestra en la figura **2.3**.

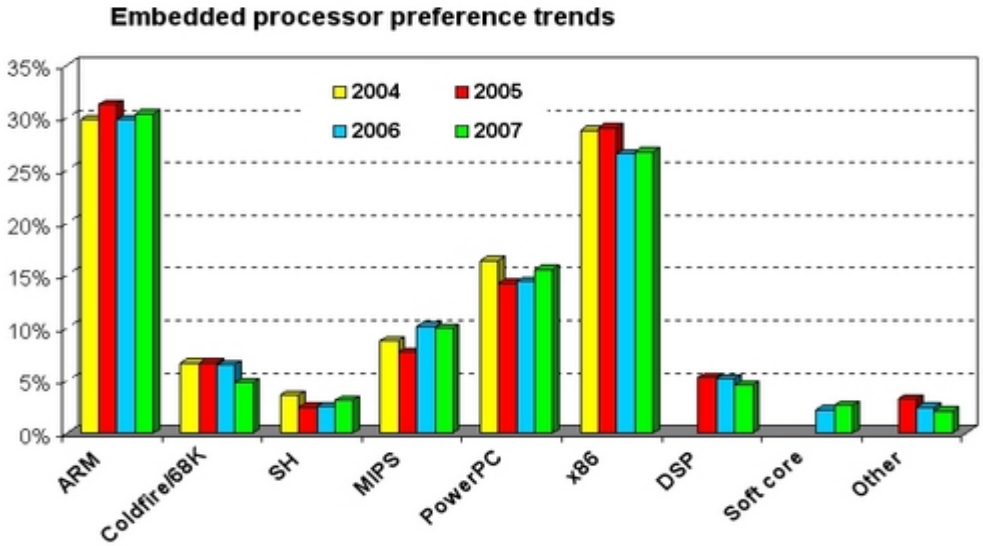


Figura 2.3 Estadísticas de arquitecturas para sistemas embebidos (Fuente (1))

Un sistema Linux embebido simplemente hace referencia a un sistema embebido basado en el kernel de Linux. Es importante destacar que no existe un kernel específico para sistemas

embebidos, es decir, no necesitamos crear un kernel especial para sistemas embebidos. A menudo se utilizan las versiones oficiales del kernel de Linux para construir un sistema, por supuesto, es necesario configurar al mismo para dar soporte especial al hardware de un determinado equipo. Fundamentalmente, un kernel utilizado en un sistema embebido difiere de un kernel usado en una computadora de escritorio o servidor en la configuración del mismo al momento de compilarlo.

Linux embebido tiene algunas ventajas en relación a otros sistemas operativos embebidos, como pueden ser el Código abierto, pequeño (Windows CE ocupa 21 MB comparado con los 2 MB para Linux embebido), puede no tener costos por derechos, maduro, estable (Más de 20 años de edad y utilizado en muchos dispositivos) y con respaldo.

2.2 Controladores en Linux

2.2.1 Definición

Un driver o controlador posibilita que el sistema operativo de una computadora pueda entenderse con un equipamiento periférico, como es el caso de una impresora, una placa de video, un mouse, un módem, etc. Para poder interactuar con el periférico, el sistema operativo debe realizar una abstracción del hardware brindando una forma de manipularlo mediante una interface, esto es, algún mecanismo que permita controlar su funcionamiento, un conjunto de instrucciones que indican de qué manera debería comunicarse con tal o cual dispositivo.

2.2.2 Módulo Cargable

Un módulo es una pieza de código que puede ser enlazado dinámicamente al kernel en cualquier momento después de arrancar el sistema. La estructura básica de un módulo puede ser vista en el siguiente ejemplo:

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void){
    printk(KERN_ALERT "Hello, world\n");
    return 0;
}
static void hello_exit(void){
printk(KERN_ALERT "Goodbye, cruel world\n");}
module_init(hello_init);
```

module_exit(hello_exit);

Este módulo define dos funciones, una puede ser invocada cuando el módulo es cargado dentro del kernel (hello_init) y la otra cuando el módulo es removido (hello_exit).

Module_init y module_exit son macros especiales del kernel para indicar el rol de esas funciones. Otra macro especial es MODULE_LICENSE la que se usa para decirle al núcleo que este módulo tiene una licencia libre.

La función printk es definida en el kernel de Linux y hecha disponible a los módulos, su comportamiento es similar a la función printf de la librería estándar C. El kernel de Linux necesita su propia función de impresión porque esta corre por sí misma, sin ayuda de una librería C. El módulo puede llamar a printk porque después de que insmod lo ha cargado, el módulo es enlazado al kernel y puede acceder a los símbolos públicos de este (funciones, variables). El string KERN_ALERT es la prioridad del mensaje. Se ha especificado una alta prioridad de mensaje en el módulo.

Podemos testear el módulo con las utilidades insmod y rmmod. Solamente el superusuario podrá cargar y descartar módulos.

root# insmod ./hello.ko

Hello, world

root# rmmod hello

Goodbye cruel World

De acuerdo al mecanismo usado por tu sistema para enviar los mensajes las salidas pueden ser diferentes. Los mensajes anteriores fueron tomados desde una consola de texto, si se está ejecutando insmod y rmmod bajo un emulador terminal corriendo bajo el sistema de ventanas no aparecerá nada en la pantalla. Los mensajes irán a uno de los archivos de registro del sistema como /var/log/message.

Antes de seguir adelante es necesario conocer la diferencia entre módulos kernel y una aplicación. Mientras la mayoría de las aplicaciones ejecutan una tarea desde un inicio a fin, un módulo kernel es registrado para poder proporcionar un requerimiento futuro y su función de inicialización termina inmediatamente. En otras palabras la tarea de la función de inicialización es preparar para la invocación más tarde de las funciones del módulo. La función exit del módulo se invoca justo antes de que el módulo sea descargado.

Este tipo de enfoque es similar a la programación orientada a eventos, pero no todas las

aplicaciones son orientadas a eventos mientras los módulos sí.

Una diferencia entre aplicaciones orientadas a eventos y módulos kernel es la función exit la cual es la encargada de limpiar todo lo que la función init realizó.

La habilidad de descargar un módulo es una de las características de modularización más apreciadas porque permite realizar distintas versiones de un driver sin necesidad de reiniciar el sistema.

Como programador se conoce que una aplicación puede llamar a una función la cual es implementada en librerías como por ejemplo printf definida en libc. Un módulo es enlazado al kernel y solo se pueden llamar a funciones exportadas por el kernel, allí no hay librerías para enlazar. La función printk es la versión del printf dentro del kernel y exportada a los módulos.

Un módulo corre en el espacio kernel, mientras que una aplicación en el espacio usuario. Todos los procesadores actuales tiene por lo menos dos niveles de protección, y algunos como la familia x86 más de uno; cuando distintos niveles existen el más alto y bajo son usados. Bajo Unix, el kernel se ejecuta en el nivel más alto (también llamado modo supervisor) donde todo es permitido, mientras las aplicaciones se ejecutan en el modo más bajo (llamado modo usuario), donde el procesador regula el acceso directo al hardware y el acceso no autorizado a memoria.

Usualmente se refiere a los modos de ejecución como espacio kernel y espacio usuario. Estos términos no solamente abarcan los diferentes niveles de privilegio inherentes a los dos modos, sino que cada uno tiene su propio espacio de direcciones.

Unix transfiere la ejecución desde el espacio de usuario al espacio kernel cuando la aplicación emite una llamada al sistema o es suspendida por una interrupción hardware.

El fin de un módulo es extender la funcionalidad del kernel; código modularizado corre en el espacio kernel. Algunas funciones en el módulo son ejecutadas como parte de una llamada al sistema, y otras en el manejador de interrupciones.

Para compilar y cargar un módulo kernel debemos tener ciertas consideraciones como asegurarse que se tiene la versión adecuada del compilador , utilidades de módulo ,etc . El archivo Documentation/Changes en el directorio de documentación del kernel se encuentra las versiones de las herramientas requeridas, intentar construir el kernel (y sus módulos) con versiones de herramientas erróneas puede conducir a un sin fin de problemas. Tenemos que considerar que una versión de compilar nueva puede generar

tantos problemas como una versión antigua.

Para poder construir un módulo cargable debemos de tener el árbol de kernel en nuestro sistema de archivos.

Para compilar el ejemplo trivial escrito arriba debemos de crear un makefile colocando obj-m := hello.o .

La asignación de arriba indica que hay un módulo que será construido a partir del código objeto hello.o. El módulo resultante es llamado hello.ko después de haber sido construido desde el archivo objeto.

Si tenemos un módulo llamado module.ko el cual es generado desde dos archivos fuente (llamados, file1.c y file2.c), deberíamos generar un makefile como:

obj-m := module.o

module-objs := file1.o file2.o

Para que el makefile de arriba funcione debemos de llamar al comando make desde el directorio donde se encuentra nuestro árbol fuente del kernel. Si nuestro árbol se encuentra en el directorio ~/kernel-2.6 debemos colocar:

make -C ~/kernel-2.6 M=`pwd` modules.

Este comando empieza cambiando su directorio al proporcionado por la opción -C (directorio fuente del kernel) , allí encontrará el top-level makefile del kernel. La opción M= causa que makefile retroceda al directorio fuente de módulo tratando de construir los módulos especificados en la variable obj-m.

La utilidad insmod nos permite cargar el código de módulo dentro del kernel en ejecución por la resolución de todos los símbolos desde la tabla de símbolos exportables del kernel. La tabla contiene las direcciones de los ítems globales del kernel como funciones o variables las cuales son necesarias para la implementación de drivers modularizados. Cuando un módulo es cargado cualquier símbolo exportado por el módulo llega ser parte de la tabla de símbolo del kernel.

La utilidad modprobe es parecida a la insmod pero se diferencia en que este toma la precaución de cargar los módulos del cual dependa el módulo que está siendo cargado.

Los módulos pueden ser removidos del núcleo usando la utilidad rmmod.

Debemos de considerar que la remoción de un módulo puede fallar si es que el kernel cree que todavía está siendo usado. Se puede configurar el kernel para forzar la remoción de un módulo incluso cuando ellos parecen estar ocupados. De todas formas si hay algún problema en remover un módulo bastará simplemente con reiniciar el sistema.

El programa lsmod produce una lista de los módulos actualmente cargados en el kernel, lsmod trabaja leyendo el archivo virtual /proc/modules. Información de los módulos actualmente cargados también puede ser encontrado en /sys/module.

2.2.3 Controladores para Dispositivos de Caracteres

Esta clase de drivers **(2)** es disponible para la mayoría de dispositivos de hardware simples, basados en la transferencia de flujo de bytes desde y hacia programas usuarios. Char devices son accedidos a través de archivos de dispositivo o nodos desde el árbol del sistema de archivos los cuales son convencionalmente localizados en el directorio /dev. Estos archivos especiales para char drivers son identificados por un "c" en la primera columna de la salida de ls -l.

Si emitimos el comando ls -l desde el directorio /dev observaremos dos números separados por coma en la entrada de los archivos de dispositivos antes de la fecha de la última modificación.

```
rw-rw-rw-  1 root root    1,  3  Apr 11 2002   null
crw-------  1 root root   10,  1  Apr 11 2002   psaux
crw-------  1 root root    4,  1  Oct 28 03:04  tty1
crw-rw-rw-  1 root tty     4, 64  Apr 11 2002   ttys0
crw-rw----  1 root uucp    4, 65  Apr 11 2002   ttyS1
crw--w----  1 vcsa tty     7,  1  Apr 11 2002   vcs1
crw--w----  1 vcsa tty     7,129  Apr 11 2002   vcsa1
crw-rw-rw-  1 root root    1,  5  Apr 11 2002   zero
```

Tradicionalmente el número mayor identifica el driver asociado con el dispositivo. Por ejemplo, /dev/null y /dev/zero ambos son manejados por el driver 1, mientras que la consola virtual y el terminal serial son manejados por el driver 4; similarmente, ambos dispositivos vcs1 y vsca1 son manejados por el driver 7. Modernos kernel permiten múltiples drivers compartan números mayores pero la mayoría de los dispositivos manejan el principio de un número major un driver.

El número menor es usado por el kernel para determinar exactamente cuál es el dispositivo que está siendo referido.

Dentro del kernel, el tipo dev_t definido en linux/types.h es usado para almacenar los números de dispositivos. Para versiones del kernel 2.6 dev_t es de 32 bits , 12 bits para el número major y 20 para el número menor. Sin embargo nunca se debería asumir respecto a la organización de los números de los dispositivos, para obtenerlo el número major y

menor desde un dev_t deberíamos utilizar las siguientes macros encontradas en linux/kdev_t.h :

MAJOR(dev_t dev);

MINOR(dev_t dev);

Para colocar los números mayor y menor dentro de un dev_t deberíamos utilizar:

MKDEV(int major , int minor);

Algunos números mayores de ciertos dispositivos son asignados estáticamente. Una lista de esos dispositivos pueden ser encontrados en Documentation/devices.txt dentro del árbol del kernel.

La mayoría de las operaciones fundamentales de un driver involucra tres estructuras de kernel importantes llamadas file_operations, file, inode.

Nosotros hemos reservado algunos números de dispositivos para nuestro uso pero todavía no hemos conectado nuestras operaciones de driver a esos números, la estructura file_operation es como nuestro char driver establece esta conexión. La estructura, definida en linux/fs.h es una colección de punteros a funciones. Cada campo en la estructura debe apuntar a la función en el driver que implementa una operación específica o ser nulo para una operación no soportada.

Las funciones más comunes que se presentarán en la estructura file_operation son:

struct module *owner;

Este campo es usado para prevenir que el módulo sea descargado mientras está en operación.

loff_t (*llseek) (struct file *, loff_t, int);

El método lseek es usado para cambiar la posición actual en un file y la nueva posición es retornada como un valor positivo.

ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);

Usado para recibir data desde el dispositivo.

ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);

Envía data al dispositivo.

int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);

La llamada al sistema ioctl ofrece una forma de emitir comandos específicos del dispositivo.

int (*open) (struct inode *, struct file *);

Apertura un dispositivo, es siempre la primera operación ejecutada en un archivo de

dispositivo.

int (*release) (struct inode *, struct file *);

Esta operación es invocada cuando la estructura file es liberada.

Como se puede observar en la lista de métodos de file_operation notaremos que un número de parámetros incluye el string __user. Esta notación indica que el puntero hace referencia a una dirección del espacio usuario la cual no puede ser desreferenciada directamente.

La estructura file , definida en linux/fs, es la segunda más importante. Esta estructura representa un open file. Es creada por el kernel en el método open y es pasada a cualquier función que opere en el archivo hasta el último close. Después de que todas las instancias de el file son cerrados el kernel libera la estructura.

Los campos más importantes de la estructura file son :

mode_t f_mode;

El file mode identifica si el archivo se puede leer, escribir o ambos por medio de los bits FMODE_READ y FMODE_WRITE.

loff_t f_pos;

Indica la posición actual de lectura o escritura. El driver puede leer este valor si necesita conocer la posición actual en el file pero normalmente no debería cambiar este.

void *private_data;

Este campo es usado para asignar data, tiene que ser liberado en el método release antes de que la estructura file sea destruido por el kernel.

Otra estructura de gran importancia es la inode usada por el kernel internamente. Este difiere de la estructura file la cual representa un archivo descriptor. Puede haber múltiples estructuras file representando a múltiples descriptores abiertos pero todos ellos apuntan a una simple estructura inode.

Dos campos de esta estructura son de interés para escribir un código driver:

dev_t i_rdev;

Este campo contiene información de los números del dispositivo.

struct cdev * i_cdev;

Esta es una estructura interna del kernel usada para representar dispositivos de carácter.

Como mencionamos el kernel usa estructuras del tipo cdev para representar internamente char device. Antes de que el kernel invoque las operaciones de tu dispositivo deberás asignar y registrar una o más de esas estructuras.

El modo clásico para registrar un char device driver es usando la función:

int register_chrdev(unsigned int major, const char *name, struct file_operations *fops);

Aquí major es el número mayor de interés, name es el nombre del driver el cual aparecerá en /proc/devices, y fops es la estructura file_operations por defecto.

Una llamada a register_chrdev registrará números menores 0-255 para el major dado y configurará una estructura cdev por defecto.

Si usamos la función register_chrdev la apropiada función para remover tu dispositivo desde el sistema es:

int unregister_chrdev (unsigned int major, const char *name);

Aquí major y name deben ser los mismos que se pasaron en register_chrdev o la llamada fallará.

Ahora veremos algunos detalles de los métodos más comunes implementados en drivers:

El Método Open:

Este método es proveído por un driver para hacer cualquier inicialización preparándolo para operaciones posteriores. En la mayoría de los drivers open debería ejecutar las siguientes tareas:

Inicializar el dispositivo si este ha sido aperturado por primera vez.

Actualizar el puntero fop si es necesario.

Asignar y llenar cualquier estructura de datos a ser colocado en fprivate_data.

El Método Release:

El objetivo de este método es revertir lo hecho en el open, sus tareas serían:

Desasignar cualquier cosa que el open asignó en private_data.

Una cosa a tener en consideración es que no siempre la llamada al sistema close causará que se invoque el método release.

Solamente la llamada que actualmente libera la estructura de datos del dispositivo invocará al método release.

El kernel conserva un contador de la cantidad de veces que una estructura file está siendo utilizada. La llamada al sistema close ejecuta el método release solamente si el contador para la estructura file cae a cero, lo cual sucede cuando la estructura es destruida. Esta relación entre el método release y la llamada al sistema close garantiza que tu driver observe solamente una llamada release para cada open.

Los Métodos read y write:

Ambos ejecutan tareas similares, la cual es, copiar data desde y hacia el código de

aplicación. Sus prototipos son similares:

ssize_t read(struct file *filp, char __user *buff,size_t count, loff_t *offp);

ssize_t write(struct file *filp, const char __user *buff,size_t count, loff_t *offp);

Para ambos métodos, filp es un puntero a file y count la cantidad de datos requeridos para transferir. El argumento buff apunta a un buffer del usuario para almacenar el dato a ser escrita o el buffer vacío donde el nuevo dato debiera ser colocado. Finalmente, offp indica la posición del file a la que el usuario esta accediendo.

Debemos de recalcar que buff es un puntero del espacio usuario, por lo que no debiera ser desreferenciado directamente por el código kernel.

Uno de los problemas que pude ser causado al tratar de desreferenciar directamente un puntero es que puede tratarse de un puntero erróneo, malicioso dado desde el espacio usuario comprometiendo la seguridad del sistema debido a que podría por ejemplo sobrescribir cualquier parte de la memoria del sistema.

Por lo tanto si deseamos hacer una transferencia de arreglo de bytes desde y hacia el espacio de usuario el kernel proporciona las siguientes funciones.

unsigned long copy_to_user(void __user *to,const void *from,unsigned long count);

unsigned long copy_from_user(void *to, const void __user *from,unsigned long count);

El rol de estas funciones no sólo se limita a copiar datos desde y hacia el espacio usuario, ellos también chequean si el puntero del espacio usuario es válido. Si el puntero es inválido, ninguna copia es ejecutada, si una dirección inválida es alcanzada durante la copia, parte de ese dato es copiado. En ambos casos el valor de retorno es la cantidad de memoria a ser copiada.

La tarea del método read es copiar dato desde el dispositivo usando copy_to_user mientras que el método write tiene que copiar data desde el espacio usuario al dispositivo usando copy_from_user.

La cantidad de data transferida debiera generalmente actualizar la posición en *offp para representar la posición actual después de que se completó una llamada al sistema satisfactoriamente. El kernel luego propaga el file position a la estructura file.

2.2.4 Operaciones Avanzadas

La mayoría de los dispositivos pueden ejecutar operaciones más allá de una simple transferencia de datos, el espacio usuario a menudo tiene que ser capaz de requerir por ejemplo, un cambio en el baud rate, información de errores o la autodestrucción. Esas operaciones son usualmente soportadas vía el método ioctl , la cual implementa la llamada

al sistema del mismo nombre. La llamada al ioctl realmente es una interface común para controlar un dispositivo.

En el espacio usuario la llamada al sistema ioctl tiene el siguiente prototipo:

int ioctl (int fd, unsigned long cmd, ...);

El prototipo se destaca de las llamadas al sistema de unix debido a los puntos, los cuales usualmente indican que la función tiene un número de argumentos variables.

En un sistema real, sin embargo, no puede tener actualmente un número de argumentos variables. Las llamadas al sistema tiene que tener un prototipo bien definido, por lo tanto los puntos no significan un número de argumentos variables sino un argumento opcional, tradicionalmente identificado como un char *argp.

Los puntos están allí simplemente para evitar el chequeo de tipos en tiempo de compilación. Algunos comandos no toman ningún argumento, algunos toman un valor entero, y otros toman un puntero a un dato.

Usando un puntero es el modo de pasar datos arbitrarios a la llamada del ioctl. La naturaleza no estructurada de ioctl no es muy bien vista por los desarrolladores del kernel, sin embargo el ioctl es a menudo la opción más sencilla y directa para operaciones en dispositivos reales.

El método driver del ioctl tiene un prototipo distinto de la versión del espacio usuario:

int (*ioctl) (struct inode *inode, struct file *filp,unsigned int cmd, unsigned long arg);

Los punteros inode y filp son los mismos del método open, el argumento cmd es pasada por el usuario sin cambios y el argumento opcional arg es pasado como un unsigned long independientemente si fue dado por el usuario como un puntero o entero.

Si el programa invocante no pasa un tercer argumento, el valor de arg recibido por el driver no está definido. Debido a que el tipo de chequeo está deshabilitado en el argumento extra el compilador no puede emitir una advertencia si se ha pasado un argumento inválido al ioctl y cualquier error asociado sería difícil de detectar.

Como podremos imaginarlos la mayoría de las implementaciones de ioctl consisten en una sentencia switch grande la cual selecciona el correcto comportamiento acorde al argumento cmd.

Diferentes comandos tiene diferentes valores numéricos, los cuales son usualmente nombres simbólicos dados para simplificar código. El nombre simbólico es asignado por una definición de preprocesador. Drivers actuales usualmente declaran tales símbolos en archivos de cabecera. Programas de usuario tiene por supuesto que incluir los archivos de

cabecera para acceder a esos símbolos.

El primer instinto de muchos programadores para obtener números correspondientes a los comandos que se van a utilizar en la llamada del ioctl es elegir estos empezando de 0,1 y así sucesivamente. Existen buenas razones para no hacerlo de esa manera. El número de comando del ioctl debería ser único a través del sistema para prevenir errores por emitir un comando correcto a un dispositivo equivocado.

Si cada número del ioctl es único, la aplicación obtendrá un error EINVAL.

Para ayudar a los programadores a crear códigos de comandos ioctl únicos esos códigos han sido divididos en distintos campos.

Para elegir números ioctl para tu driver acorde a la convención del kernel linux, deberíamos chequear include/asm/ioctl.h y Documentation/ioctl-number.txt.

El archivo de cabecera define los campos de bits : type (número mágico), número ordinal, dirección de transferencia, y tamaño del argumento.

El ioctl-number.txt lista los números mágicos usados a través del kernel, tú deberías ser capaz de elegir tu propio número mágico y evitar superposiciones.

Los campos de bits usados tienen los siguientes significados

type:

El número mágico, elegir un número después de consultar ioctl-number.txt y úsalo a través del driver. Este campo es de ocho bits (_IOC_TYPEBITS).

number:

Número ordinal secuencial, es de ocho bits (_IOC_NRBITS) de ancho.

direction:

Dirección de datos transferidos, los posibles valores son _IOC_NONE (ningún dato transferido) , _IOC_READ , IOC_WRITE y _IOC_READ | IOC_WRITE (datos transferidos en ambas direcciones).

_IOC_READ significa leer desde el dispositivo, para el driver escribir al espacio usuario.

size:

Tamaño de la data involucrada, el ancho es dependiente de la arquitectura, pero usualmente es 13 o 14 bits. Tú puedes encontrar el valor para tu arquitectura específica en _IOC_SIZEBITS.

El archivo de cabecera <asm/ioctl.h> incluido en <linux/ioctl.h> define macros que te ayudan a configurar los números de comandos como sigue:

_IO(type,nr) para comandos que no tiene argumentos, _IOR(type,nr datatype) para leer

datos desde el driver ,_IOW(type,nr,datatype) para escribir data , _IOWR(type,nr,datatype) para transferencia bidireccional.

Los campos type y number son pasados como argumentos, y el campo size es conducido para aplicar sizeof al argumento datatype.

El archivo de cabecera también provee macros usadas en tu driver para decodificar los números.: _IOC_DIR(nr) , _IOC_TYPE(nr) _IOC_NR(nr) y _IOC_SIZE(nr).

El estándar POSIX establece que si un comando ioctl inapropiado ha sido emitido, -ENOTTY debería ser retornado.

Este código de error es interpretado por la librería de C como un ioctl inapropiado para el dispositivo.

A través de la llamada al sistema ioctl, unos pocos comandos son reconocidos por el kernel , esos comandos cuando son aplicados a tu dispositivo , son decodificados antes que tu propio archivo de operaciones sea llamado, así si eliges el mismo número para uno de tus comandos de ioctl nunca verás cualquier requerimiento para este y la aplicación podría hacer algo inesperado por conflictos entre números ioctl.

Algunos de los comandos predefinidos usuales son:

FIOASYNC:

Configuración de una notificación asíncrona.

FIONBIO:

“File IOctl Non_Blocking I/O”. Par poder establecer operaciones no bloqueantes.

2.2.5 Manejo de Interrupciones

Una interrupción es una simple señal que el hardware envía cuando este desea la atención del procesador. Un driver necesita registrar la interrupción y manejarla apropiadamente cuando llegue.

Los manejadores de interrupciones corren concurrentemente con otro código, de modo que debemos tener en cuenta los posibles problemas de concurrencia. Uno de los recursos preciados y escasos son las líneas de interrupción. El kernel conserva un registro de dichas líneas.

Un módulo debe requerir un canal de interrupción antes de usarlo y liberarlo cuando finalice.

Las siguientes funciones declaradas en <linux/interrupt.h> implementan la interface de registro de interrupción:

int request_irq (unsigned int irq, irqreturn_t (*handler)(int, void *, struct pt_regs *),

unsigned long flags, const char *dev_name, void *dev_id);

void free_irq (unsigned int irq, void *dev_id);

El valor de retorno de request_irq es cero en caso de éxito o un código de error negativo.

No es raro que la función retorne –EBUSY para indicar que otro driver está usando la línea de interrupción requerida.

Los argumentos de las funciones son:

unsigned int irq:

El número de la interrupción solicitada.

irqreturn_t (*handler)(int, void *, struct pt_regs *):

El puntero a una función manejadora instalada.

unsigned long flags:

Máscara de bits de opciones relacionadas al manejo de la interrupción.

const char *dev_name:

Este string es usado en /proc/interrupts para mostrar el propietario de la interrupción.

void *dev_id:

Puntero usado para líneas e interrupciones compartidas. Si la interrupción es no compartida, dev_id puede configurarse a NULL.

SA_INTERRUPT:

Indica un manejador de interrupción rápido. Los manejadores rápidos son ejecutados con interrupciones deshabilitas en el procesador actual.

SA_SHIRQ:

Indica que la interrupción puede ser compartida entre dispositivos.

El manejador de interrupciones puede instalarse en la inicialización del driver o cuando el dispositivo es aperturado por primera vez.

Instalar el manejador de interrupciones desde la función de inicialización no es una buena idea especialmente si tu driver no comparte interrupciones.

Si un módulo requiere una interrupción en la inicialización, esto previene que cualquier otro driver use la interrupción, incluso si el dispositivo que la solicitó nunca la usara.

El lugar más apropiado de llamar a request_irq es cuando el dispositivo es aperturado, y para la llamada de free_irq es la última vez que el dispositivo es cerrado.

El reporte de las interrupciones puede ser mostrado en /proc/interupts.

2.2.6 Manejo de Concurrencia

Linux es un kernel reentrante **(3)** esto significa que más de un proceso pueda estar

ejecutándose en modo kernel al mismo tiempo por supuesto en sistemas monoprocesador, sólo un proceso puede progresar, pero el resto puede estar bloqueado en modo kernel esperando por la CPU o que se completa alguna operación de E/S.

Si ocurre una interrupción de hardware, un kernel entrante es capaz de suspender el proceso actual, incluso si este proceso esta ejecutándose en modo kernel. Esta capacidad es muy importante, ya que mejora el rendimiento de los controladores de dispositivos.

Un camino de control del kernel (Kernel Control Path, KCP a partir de ahora) denota una secuencia de instrucciones ejecutada por el kernel para manejar una llamada al sistema, una excepción o una interrupción. En el caso más simple, la CPU ejecuta el KCP secuencialmente, desde la primera instrucción hasta la última. Sin embargo, cuando uno de los siguientes eventos ocurre, la CPU intercala KCPs:

● Un proceso ejecutándose en modo usuario invoca una llamada al sistema, y el KCP correspondiente verifica que el requerimiento realizado no puede satisfacerse inmediatamente. Entonces invoca al scheduler para elegir un nuevo proceso. Se realiza el context switch y un nuevo proceso comienza a ejecutarse. Éste podría hacer una llamada al sistema y tendríamos 2 procesos en modo kernel.

● La CPU detecta una excepción (por ejemplo, el acceso a una página no presente en RAM) mientras está ejecutando un KCP. Éste es suspendido y la CPU comienza la ejecución del manejador de la excepción.

● Una interrupción de hardware ocurre mientras la CPU se encuentra ejecutando un KCP (con las interrupciones activadas). Similar al anterior.

● Una función diferible comienza a ejecutarse, mientras la CPU se encuentra ejecutando un KCP.

La Implementación de un kernel reentrante requiere el uso de mecanismos de sincronización. Si un KCP es suspendido mientras modifica una estructura de datos del kernel, ningún KCP debería poder actuar sobre la misma estructura hasta que se alcance un estado consistente.

SMP (Symmetric Multi Processing) y kernel preemtion son escenarios que generan múltiples hilos de ejecución. Esos hilos pueden simultáneamente operar en estructuras compartidas del kernel. Debido a esto el acceso a tales estructuras tiene que ser serializado.

Un área de código la cual accede a recursos compartidos es llamada sección crítica. Spinlocks es un mecanismo básico usado para proteger secciones críticas del kernel.

Un spinlock asegura que solo un hilo de ejecución ingresa a la sección crítica a la vez. Cualquier otro hilo que desee ingresar a la sección crítica tendrá que permanecer girando en un lazo hasta que el primer hilo salga.

El uso básico de spinlocks es como sigue:

```
#include <linux/spinlock.h>
spinlock_t mylock = SPIN_LOCK_UNLOCKED; /* Initialize */
/* Acquire the spinlock. This is inexpensive if there
* is no one inside the critical section. In the face of
* contention, spinlock() has to busy-wait.
*/
spin_lock(&mylock);
/* ... Critical Section code ... */
spin_unlock(&mylock); /* Release the lock */
```

Para ilustrar el uso de protección por concurrencia, analizaremos cuatro casos:

Caso 1: Process Context, UP Machine, No Preemption:

Este simple caso no necesita de bloqueo.

Case 2: Process and Interrupt Contexts, UP Machine, No Preemption

En este caso sólo necesitaremos bloquear las interrupciones para proteger el acceso a la región crítica.

Para ver porque asume que A y B son hilos en el contexto de proceso, y C un hilo en contexto de interrupción, todos intentando ingresar a la misma sección crítica, como muestra la Figura **2.4** .

A causa de que el hilo C esta ejecutándose en un contexto de interrupción y siempre se ejecuta hasta el final antes de ceder al hilo A o B, este no necesita preocuparse por protección. El hilo A por su parte no tiene por qué preocuparse por el hilo C y viceversa porque el kernel es no preemptible.

Así el hilo A e hilo B necesitan protegerse solamente de la posibilidad de que el hilo C ingrese a la sección crítica mientras ellos estén dentro de la misma sección. Ellos lograrán este por deshabilitar las interrupciones.

```
Point A:
local_irq_disable(); /* Disable Interrupts in local CPU */
/* ... Critical Section ... */
local_irq_enable(); /* Enable Interrupts in local CPU */
```

Sin embargo si las interrupciones han sido deshabilitadas cuando se alcanzó el punto A ,local_irq_enable() puede generar el efecto secundario de rehabilitar interrupciones y no de restablecer el estado de las interrupciones. Esto se puede resolver de la siguiente manera.

```
unsigned long flags;
Point A:
local_irq_save(flags); /* Disable Interrupts */
/* ... Critical Section ... */
local_irq_restore(flags); /* Restore state to what
it was at Point A */
```

Esto funciona correctamente independiente del estado de las interrupciones en el punto A.

Case 3: Process and Interrupt Contexts, UP Machine, Preemption:

Si preemption está habilitado no bastará con deshabilitar las interrupciones. El hilo A y el hilo B necesitan protegerse entre sí mismos además del protegerse contra hilo C. La solución aparentemente es deshabilitar kernel preemption antes de empezar la sección crítica y rehabilitarlo al final de esta, en adición de habilitar y deshabilitar interrupciones.

Para esto el hilo A e hilo B usa una variante de spinlock:

```
unsigned long flags;
Point A:
/* Save interrupt state.
* Disable interrupts - this implicitly disables preemption */
spin_lock_irqsave(&mylock, flags);
/* ... Critical Section ... */
/* Restore interrupt state to what it was at Point A */
spin_unlock_irqrestore(&mylock, flags);
```

Case 4: Process and Interrupt Contexts, SMP Machine, Preemption:

En este caso el spinlock no solo hace la habilitación y rehabilitación de las interrupciones y preemption . La funcionalidad de bloque ha sido realmente compilada. En presencia de SPM la lógica de bloque es compilada.

```
unsigned long flags;
Point A:
/*
- Save interrupt state on the local CPU
```

```
- Disable interrupts on the local CPU. This implicitly disables
preemption.
- Lock the section to regulate access by other CPUs
*/
spin_lock_irqsave(&mylock, flags);
/* ... Critical Section ... */
/*
- Restore interrupt state and preemption to what it
was at Point A for the local CPU
- Release the lock
*/
spin_unlock_irqrestore(&mylock, flags);
```

En sistemas SPM , solamente las interrupciones locales CPU son deshabilitadas cuando adquirimos un spinlock.

Así un hilo en contexto de proceso puede estar corriendo en un CPU (por ejemplo hilo A) , mientras que un manejador de interrupciones esta ejecutándose en otro CPU (hilo C). Un manejador en un procesador no local necesita esperar hasta que el código del contexto de proceso en el procesador local (hilo A) salga de la sección crítica.

El código del contexto de interrupción llama a spin_lock()/spin_unlock():

```
spin_lock(&mylock);
/* ... Critical Section ... */
spin_unlock(&mylock);
```

2.3 Interfaz de usuario

La creación de las interfaces de usuario ha sido un área del desarrollo de software que ha evolucionado dramáticamente a partir de la década de los setentas. La interfaz de usuario es el vínculo entre el usuario y el programa de computadora. Una interfaz es un conjunto de comandos o menús a través de los cuales el usuario se comunica con el programa.

Esta es una de las partes más importantes de cualquier programa ya que determina que tan fácilmente es posible que el programa haga lo que el usuario quiere hacer. Un programa muy poderoso con una interfaz pobremente elaborada tiene poco valor para un usuario no experto.

La elaboración de una interfaz de usuario, bien diseñada, exige una gran dedicación pues generalmente las interfaces son grandes, complejas y difíciles de implementar, depurar y

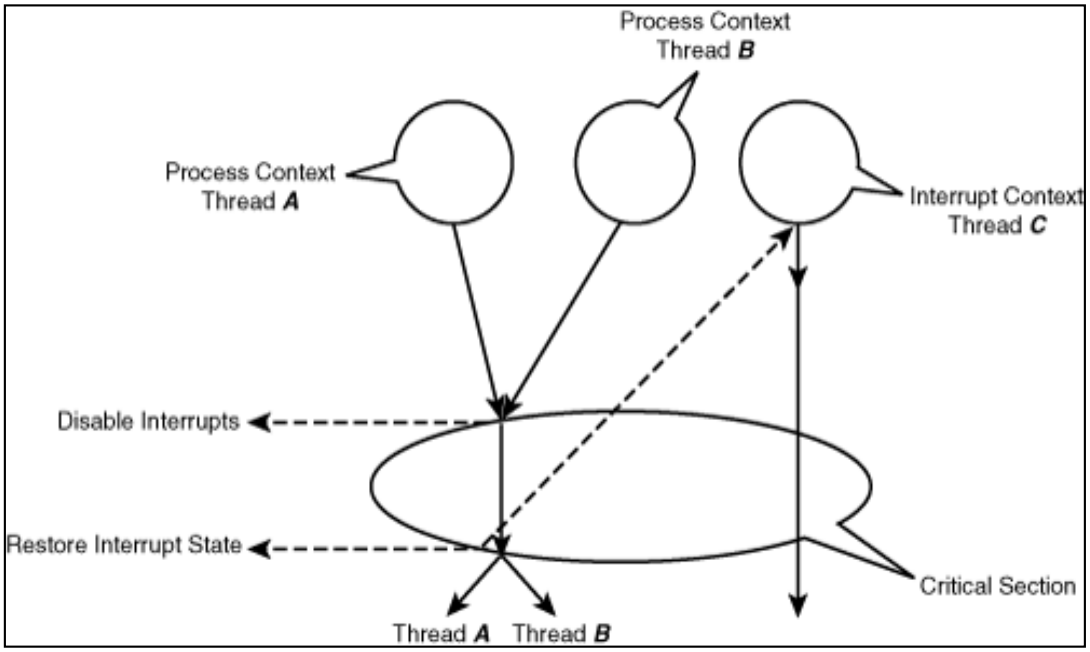


Figura 2.4 Manejo de concurrencia de tres Procesos (Fuente (3))

modificar. Hoy en día las interfaces de manipulación directa (también llamadas interfaces gráficas de usuario, GUI por sus siglas en inglés) son prácticamente universales. Las interfaces que utilizan ventanas, íconos y menús se han convertido en estándar en los materiales computacionales.

La interfaz representa el punto de encuentro entre el usuario y la computadora. En esta interacción, el usuario juzga la utilidad de la interfaz; el hardware y el software se convierten en simples herramientas sobre los cuales fue construida la interfaz. La definición de interfaz en sí misma es un tanto arbitraria, aunque esto depende de la naturaleza de la tarea que se tiene enfrente.

Existen muchos tipos de software para la creación de interfaces de usuario .El sistema de ventanas permite la división de la pantalla en diferentes regiones rectangulares, llamadas "ventanas". El sistema de ventanas XWindows para Unix divide la funcionalidad de la ventana en dos capas: el sistema de ventanas, el cual es la interfaz funcional, y el administrador de ventanas. El sistema de ventanas provee de procedimientos que permiten a la aplicación el dibujar figuras en la pantalla y sirve como medio de entrada de las acciones del usuario. El administrador de ventanas le permite al usuario final el mover las ventanas por la pantalla, y es el responsable de desplegar las líneas de título, bordes e íconos alrededor de las ventanas.

La parte central de un sistema de ventanas es el conjunto de herramientas (toolkit) , el cual contiene los objetos gráficos (widgets) más empleados tales como menús, botones, barras

de scroll, y campos para entrada de texto. El toolkit generalmente se conecta a los programas de aplicación a través de una serie de procedimientos definidos por el programador. La función de estos procedimientos es el decidir la forma en que se comportarán los objetos gráficos. Ejemplos de toolkits son el Motif/X11 y los widgets definidos en los toolboxes para Windows y Macintosh. Se ha investigado mucho sobre los toolkits, produciendo algunos como Garnet, Amulet, InterViews, Tcl/Tk, Java y las bibliotecas de objetos gráficos de QT.

Los toolkits virtuales le permiten a una aplicación el que esta sea escrita una sola vez y que se pueda ejecutar en diferentes sistemas operativos, como Windows, MacOS o Unix. Otro nombre para estos toolkits es de "sistemas de desarrollo multiplataforma".

2.3.1 Definición.

La interfaz de usuario (IU) **(4)** es uno de los componentes más importantes de cualquier sistema computacional, pues funciona como el vínculo entre el humano y la máquina. La interfaz de usuario es un conjunto de protocolos y técnicas para el intercambio de información entre una aplicación computacional y el usuario. La IU es responsable de solicitar comandos al usuario, y de desplegar los resultados de la aplicación de una manera comprensible. La IU no es responsable de los cálculos de la aplicación, ni del almacenamiento, recuperación y transmisión de la información.

El éxito de un programa frecuentemente se debe a qué tan rápido puede aprender el usuario a emplear el software, de igual importancia es el que el usuario alcance sus objetivos con el programa de la manera más sencilla posible.

Es importante señalar que dentro del proceso de creación de la IU existen cuatro diferentes tipos de personas involucradas. La primera persona, y probablemente la más importante, es el usuario final o simplemente usuario. El usuario es quien va a utilizar el programa final. La segunda persona es aquella que crea la interfaz de usuario. Esta persona es conocida como diseñador o arquitecto de la interfaz de usuario. Trabajando muy cercanamente con el diseñador estará el programador de la aplicación, este será el encargado de la escritura del software del resto de la aplicación. Muy frecuentemente el diseñador utilizará herramientas especiales para la creación del software de la IU (como los toolkits2.1) , y estas herramientas son elaboradas por el creador de herramientas .

Dificultades para la creación de interfaces de usuario

El software es juzgado en base a muchos factores, los cuales incluyen la robustez del programa, funcionalidad, velocidad, y facilidad de uso. La elaboración de esta es un

proceso complicado, a continuación se mencionan algunas razones sobre esto:

Desconocimiento del usuario

Es difícil saber el grado de conocimientos de cómputo del usuario final, lo cual, frecuentemente, hace que las interfaces de usuario desarrolladas no sean las apropiadas. Se da el caso de que el diseñador implemente la IU pensando en que la van a usar programadores avanzados, como el propio diseñador, por lo que cuando el producto final es usado por el usuario es posible que se presenten una gran cantidad de problemas de usabilidad.

Deben manejar múltiples eventos.

El software de la interfaz generalmente debe de estar organizado para manejar múltiples eventos, los cuales se suelen presentar de manera concurrente. Los usuarios desean tener la posibilidad de abortar acciones, y también de deshacerlas. Así mismo, debe ser posible realizar todas estas acciones por medio de distintos métodos de entrada (con el teclado o el mouse, p. ej.).

El programa en que se haya implementado la IU debe de estar estructurado de tal manera que pueda aceptar los eventos de entrada en cualquier momento, aun y cuando esté ejecutando otros comandos. Consecuentemente, cualquier operación que tome mucho tiempo, como la impresión, búsqueda, e incluso el repintado de la pantalla, deben de ser ejecutados en un proceso separado. Una motivación extra para el empleo de procesos múltiples es el hecho de que el usuario pueda estar trabajando con distintas ventanas pertenecientes a una misma aplicación. Cada una de esas ventanas necesitará recordar y manejar todos los eventos producidos por el usuario, o por la misma aplicación.

Existen requerimientos de tiempo real para la de los eventos de entrada.

La creación de IUs tiene las dificultades de la programación en tiempo real. La mayoría de las interfaces gráficas y con manipulación directa tendrán objetos que sean animados o que sea posible moverlos con el mouse. Para que esto sea atractivo para el usuario, los objetos deben de ser redibujados entre 30 y 60 veces por segundo, sin pausas notables.

La próxima generación de interfaces de usuario incluiría nuevas tecnologías como video, reconocimiento de voz, animación de simulaciones, y otros tipos de multimedia que contarán con las restricciones de la programación de tiempo real.

Deben de ser programadas de "adentro hacia afuera".

Las IU modernas son escritas de adentro hacia afuera. En vez de organizar el código para que la aplicación tenga el control, la aplicación debe más bien estar dividida en muchas

subrutinas que son llamadas cuando el usuario realiza alguna acción. Se requiere de una programación con un cuidado extremo en la organización y modularización del software de la IU.

El software debe ser especialmente robusto.

Por supuesto que todo el software debe de ser robusto, pero en el caso de la IU este tema es de capital importancia ya que se deberán manejar una gran cantidad de eventos de entrada-salida, ocurriendo de manera concurrente. A diferencia de los programas no interactivos, sin incluir al software para IU, en donde al ocurrir un error se requiere el uso de un debugger para identificar el error y corregirlo, en el caso del software de interfaz de usuario si un error ocurriese se deberá presentar un mensaje de error, explicando que fue lo que sucedió y dar la oportunidad hacer las correcciones pertinentes.

Dificultad para modularizar el programa.

El escribir un programa de manera modular facilita el mantenimiento del mismo. La bibliografía especializada recomienda que la porción correspondiente a la IU esté separada del resto del software, para facilitar que esta sea fácilmente alterada (para el diseño iterativo). Desafortunadamente, la separación de estas dos partes es muy difícil, prácticamente imposible, ya que los cambios en la IU requieren inevitablemente cambios en el resto del software. Incluso con la utilización de herramientas para la creación de interfaces de usuario gráficas (Graphical User Interfaces - GUI) el problema de modularidad se hace más difícil por la gran cantidad de funciones call-back. Generalmente, cada widget en la pantalla requiere que el programador escriba al menos un procedimiento de aplicación a ser llamado cuando el operador lo activa. Cada tipo de widget tendrá su propia secuencia de funciones call-back. Una puede contener miles de widgets, por lo que habrá al menos la misma cantidad de funciones call-back.

2.3.2 Herramientas de Diseño Gráfico

Existen herramientas que facilitan la creación de interfaces gráficas de usuario. Estas son conocidas como toolkits. Un toolkit consiste en una biblioteca de widgets que pueden ser llamados por los programas de aplicación. Típicamente los widgets usados en los toolkits son objetos como barras de scroll, botones, cuadros de texto, etc. En la figura **2.5** se muestran algunos widgets del toolkit de QT, del cual se hablará en detalle más adelante en esta tesis.

Los toolkits facilitan enormemente la creación de IUs; sin embargo, aun utilizando el toolkit, es necesario conocimientos de programación para la elaboración de la interfaz

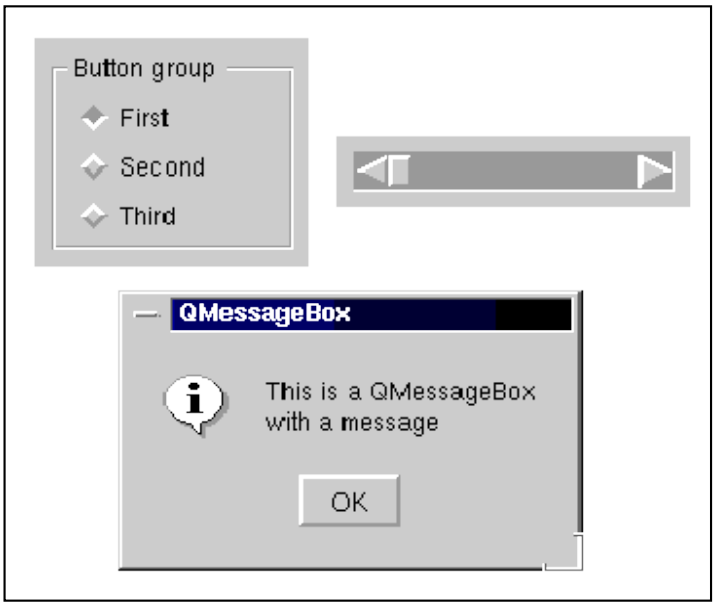


Figure 2.5 Widgets del toolkit QT. (Fuente (4))

puesto que los widgets sólo tienen una interfaz procedural.

El uso de toolkits tiene grandes beneficios, como los que se mencionan a continuación:

Independencia de la IU:

Separa la parte correspondiente al diseño de la IU de las complejidades de la programación. Esta separación le permite a los diseñadores el realizar prototipos rápida y fácilmente, hacer revisiones en minutos, y de esta manera agilizar los procedimientos de prueba. La programación necesaria para el sistema se empieza a escribir una vez que el diseño de la IU se ha estabilizado.

Código más confiable:

El código generado por los toolkits es frecuentemente más confiable que el código creado por un implementador humano, por lo que requiere una menor cantidad de debugging. Esto se debe a que el código generado automáticamente tiene un alto nivel de especificación.

Elaboración rápida de prototipos:

Con un toolkit las IUs pueden ser construidas y modificadas rápidamente. Gran parte del trabajo de diseño y construcción de un programa se debe a la creación de la IU. Al utilizar un toolkit para su elaboración, el tiempo de fabricación se reduce de una manera considerable. El tiempo ganado puede ser utilizado para hacer una mayor cantidad de pruebas de evaluación (probar, revisar, probar, revisar...), y tener al final un producto apropiado para el usuario final.

Interfaces de usuario más fácil de crear y de mantener:

Esto es porque las especificaciones de la interfaz pueden ser representadas, validadas y

evaluadas más fácilmente gracias al empleo de los widgets incluidos en el toolkit, la cantidad de código a escribir es mucho menor.

Facilidad para hacer modificaciones:

El empleo de un toolkit permite que la incorporación de cambios a la interfaz después de haber hecho las pruebas correspondientes sea más sencilla.

Un aspecto muy importante a considerar es la utilización de herramientas que produzcan programas multiplataforma, lo que posibilita el que la interfaz pueda ser usada en distintos ambientes operativos como windows o Unix, sin hacer modificación alguna a la IU. Ejemplos de toolkits multiplataforma son Tk y QT.

Así pues, podemos concluir que el uso de un toolkit provee al diseñador de la IU de gran control y flexibilidad.

2.4 Microcontroladores

2.4.1 Introducción.

Los microcontroladores son computadores digitales integrados en un chip que cuentan con una unidad de procesamiento central (CPU), una memoria para almacenar el programa, una memoria para almacenar datos y puertos de entrada salida. A diferencia de los microprocesadores de propósito general, como los que se usan en los computadores PC, los microcontroladores son unidades autosuficientes y más económicas.

El funcionamiento de los microcontroladores está determinado por el programa almacenado en su memoria. Este puede escribirse en distintos lenguajes de programación.

Además, la mayoría de los microcontroladores actuales pueden reprogramarse repetidas veces.

Los microcontroladores son ampliamente utilizados como el cerebro de una gran variedad de sistemas embebidos que controlan máquinas, componentes de sistemas complejos, como aplicaciones industriales de automatización y robótica, equipos médicos, sistemas aeroespaciales, e incluso dispositivos de la vida diaria como automóviles, hornos de microondas, teléfonos y televisores.

Frecuentemente se emplea la notación μC para referirse a estos.

Las principales características de los μC son:

• Unidad de Procesamiento Central (CPU)

• Memoria de Programa

• Memoria de Datos

• Generador del Reloj

• Interfaz de Entrada/Salida:

– Puertos seriales (UARTs, Universal AsynchronousReceiver/Transmitter),

– I2C (Inter-Integrated Circuit),

– Interfaces de Periférico Seriales (SPIs, Serial Peripheral Interfaces),

– Red de Área de Controladores (CAN,Controller Area Network),

– USB (Universal Serial Bus).

• Otras opciones:

– Conversores Análogo-Digitales (A/D, analog-to-digital)

– Moduladores por Ancho de Pulso (PWM, Pulse-Width Modulation)

– Módulo Timers , Captura , Comparación.

– DMA, PLL, etc.

La alta integración de subsistemas que componen un μC reduce el número de chips, la cantidad de pistas y espacio que se requeriría en un circuito impreso si se implementase un sistema equivalente usando chips separados. A través de los pines del chip asociados a las interfaces de entrada/salida el μC puede interactuar con otros circuitos externos enviándoles señales de comando o recibiendo estímulos correspondientes a variables externas.

Por lo general varios pines de datos son bidireccionales, es decir pueden configurarse como entradas o salidas.

Cuando son entradas, pueden adquirir datos interpretando el valor de voltaje como un valor lógico 0 o 1, mientras que cuando son salidas pueden entregar una señal binaria de voltaje cuya magnitud dependerá del valor lógico 0 o 1.

Monitoreando el valor de las entradas, el microcontrolador puede responder a eventos externos y realizar una cierta acción, como variar las señales de salida de acuerdo al valor en las entradas.

Para responder a eventos externos, los μCs cuentan con un recurso conocido como interrupciones. Las interrupciones son señales que detienen la ejecución normal del programa para ejecutar alguna subrutina de respuesta al evento. Una vez ejecutada la subrutina de interrupción la ejecución del programa continúa en el punto en que se encontraba antes de generarse la interrupción. Un ejemplo típico es el de un botón pulsador conectado a un pin de entrada. Una vez pulsado, se genera una señal de interrupción que iniciará la ejecución de la subrutina de interrupción, que por ejemplo podría activar un pin de salida para encender un led.

No todas las interrupciones necesariamente están asociadas al cambio del estado de los pines de entrada. También hay interrupciones que pueden estar asociadas al valor de una entrada AD, o al cumplimiento de un periodo de tiempo fijado por un timer o temporizador.

Estas características dependerán del modelo de μC empleado.

2.5 Norma Rs-485

2.5.1 Bus RS485

El bus 485 **(5)** podemos resumirlo como un sistema de interconexión para transmisión de datos a grandes distancias y apto para operar en ámbitos eléctricamente ruidosos. Su conexión es muy sencilla: a partir del puerto serie (COM1) de cualquier ordenador utilizando tan sólo dos circuitos integrados muy económicos y fáciles de obtener: MAX232 y MAX485. En el caso del último IC mencionado se lo suele reemplazar por el SN76156, que cumple la misma función y es de menor coste. Si se construye un sistema pequeño de pocas terminales que utilizan este IC la diferencia monetaria es poca, pero al emplearlo en grandes cantidades el ahorro es importante.

El bus permite una velocidad de datos de 10 y hasta 20 Mbps (a 12 metros de distancia), y de 100 Kbps cuando se conectan terminales o módulos separados 1200 metros entre sí. El sistema permite "colgar" del bus hasta 32 terminales, aunque en la actualidad ya se están utilizando sistemas de 128 y hasta 256 dispositivos conectados entre sí a una misma red de sólo dos hilos trenzados. En el mejor de los casos, es preferible que el par de cables que

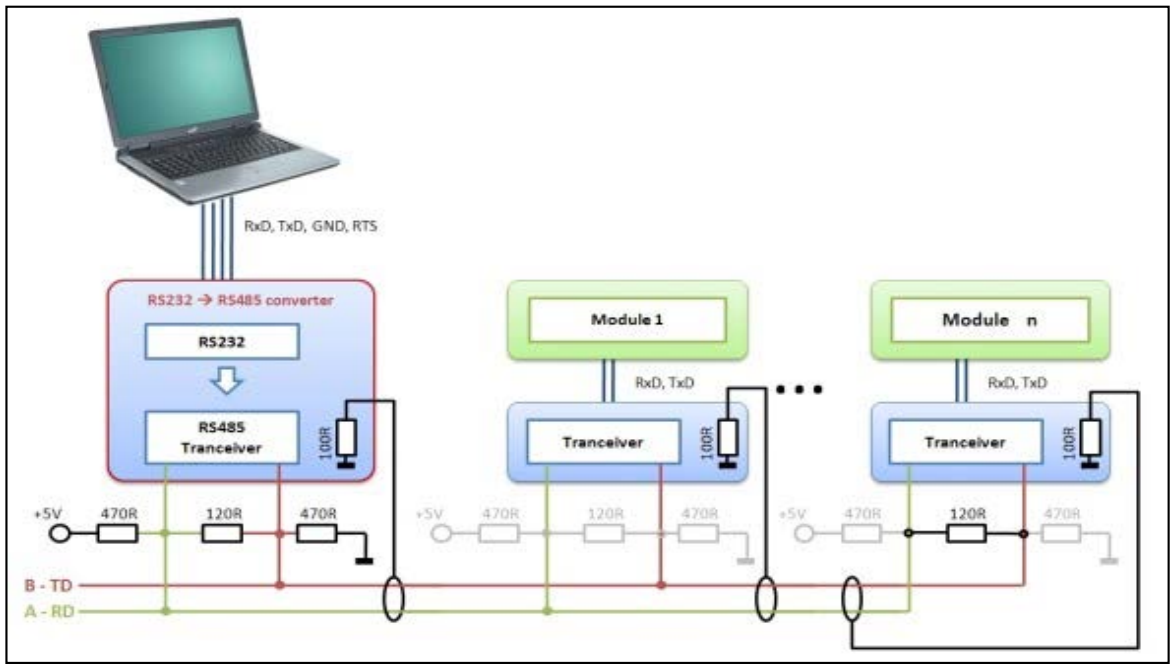


Figura 2.6 Topología clásica de un Bus RS485 (Fuente (5))

transporta la información sea blindado, pero si este montaje no es posible, y debemos utilizar cables individuales, será bueno tener un tercer cable que oficie de referencia de tierra o GND. Un cable blindado ayudaría a atenuar los ruidos eléctricos que pueden filtrarse entre los datos del sistema diferencial que utiliza el estándar RS485. Disponer de un cableado con estas características sería lo mejor.

Las especificaciones del estándar RS485 (cuyo nombre oficial es TIA/EIA–485-A) no determinan claramente cómo debe ser el correcto cableado de una red. Sin embargo, algunas recomendaciones pueden interpretarse dentro del texto de la norma y han sido estudiadas y ensayadas por ingenieros, tanto en forma conceptual como en función del método de prueba y error. Dichos ingenieros han llegado a delinear los conceptos que se utilizan hoy en día.

Debido a que altas frecuencias intervienen en el intercambio de datos, que las distancias entre las terminales siempre son inciertas, y que los cables apropiados a utilizar no se determinan en el estándar, se acepta el uso de un par de cables trenzados comunes que tienen una impedancia aproximada de 120 Ohms. Las terminaciones inapropiadas de la línea mostradas en la figura **2.9** se traducen en reflexiones no deseadas de la señal, tal como muestran en la misma figura.

En las figuras se puede apreciar claramente la distorsión sufrida en la señal, cuando el final de una línea no tiene una terminación adecuada. La reflexión ocasionada puede llevar a

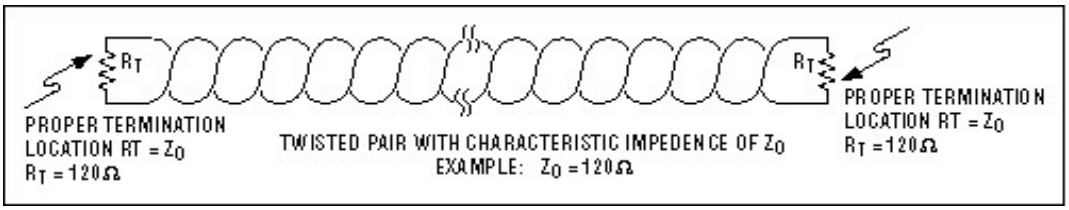


Figura 2.7 Terminaciones apropiadas a los extremos de una Red RS48 (Fuente (5))

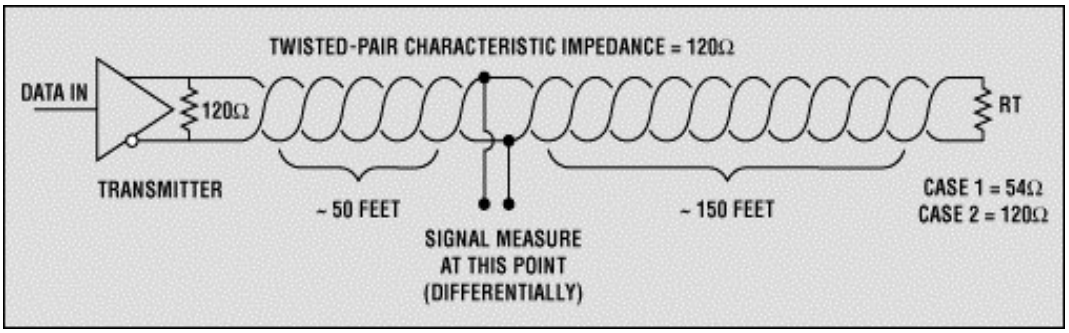


Figura 2.8 Terminaciones incorrectas provocan reflexiones indeseadas (Fuente (5))

distorsionar y perder por completo los datos transmitidos. La base del sistema se fundamenta en la transmisión de datos en forma diferencial. Es decir, por ambos cables

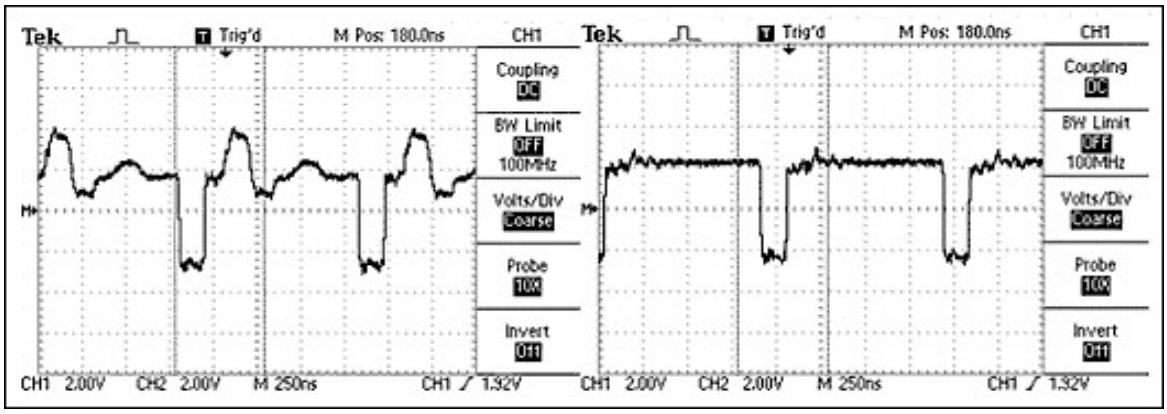


Figura 2.9 Oscilogramas indicando terminaciones incorrectas (Izquierda) (Fuente (5))

viaja la misma información, pero desfasada 180° en un cable respecto al otro. De esta forma, cualquier interferencia que pueda introducirse en el cableado lo hará en ambos hilos por igual, con la misma polaridad y amplitud. En el destino de la terminal, sea en el ordenador o en el dispositivo colocado a la distancia, las señales se restituyen en polaridad y los picos de ruidos que se habían introducido con la misma polaridad en ambos cables, al invertirse las señales, se neutralizan y eliminan entre sí, y se recupera de esta forma la señal útil que se desea transmitir como se muestra en la figura **2.10**.

Cuando el cableado recorre un ambiente ruidoso y hostil, como puede suceder en una instalación industrial, el tercer cable que oficia de tierra o GND también se transforma en un elemento que recepciona y lleva hasta las terminales o módulos los ruidos inducidos en él. Por este motivo siempre es recomendable colocar una resistencia de 100 Ohms en la

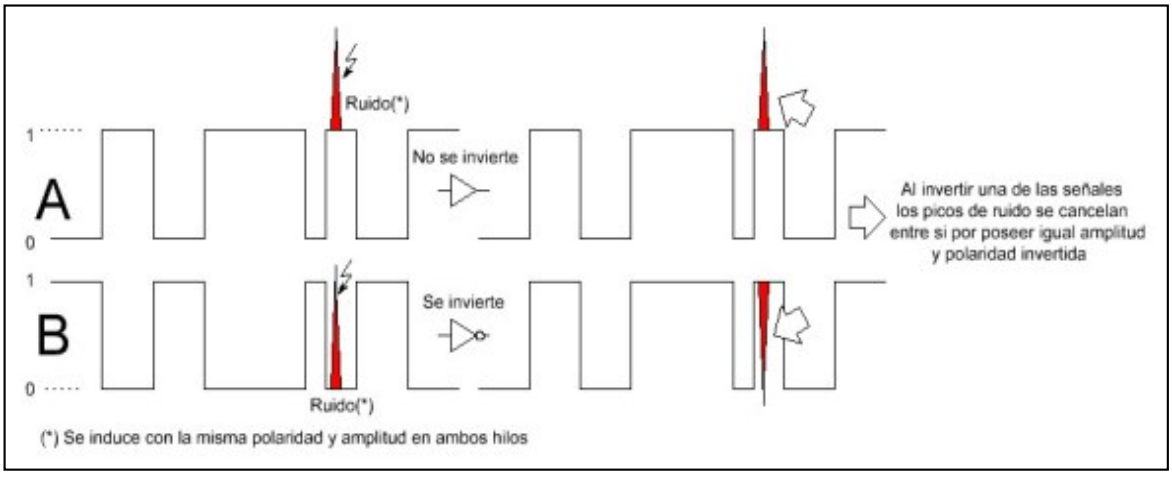


Figura 2.10 Picos de ruido inducidos en la Red (Fuente (5))

conexión a GND en cada uno de los circuitos de las terminales.

Entre las múltiples diferencias fundamentales que existen respecto al estándar RS232 es que el RS485 se maneja con niveles TTL de tensión, mientras que el RS232 maneja tensiones de ambas polaridades con valores absolutos de 3 a 15 Volts. RS232 permite

comunicaciones “full-duplex” (ambos terminales transmiten y reciben datos en forma simultánea), pero su distancia de trabajo es de tan sólo 12 metros; además, se requieren al menos 8 cables para una comunicación full y es muy propenso a ser afectado por el ruido eléctrico.

CAPÍTULO III
DISEÑO DEL SISTEMA

3.1 Sistema de Adquisición de Datos.

3.1.1 Introducción.

De la figura **3.1** podemos observar que el sistema de adquisición de datos está conformado por una interfaz paralela de 16 bits, señales de control y una configuración maestro-esclavo de dspic33f **(6)** a través del bus 485. Los dspics usarán un cristal de 18.434MHZ y su frecuencia de ciclo de instrucción (FCY) es de 36.864MHZ. El bus 485 trabajará aproximadamente a 1 Mbps.

La interfaz paralela usa el bus de datos del sistema Linux embebido, el dspic que se comunica con el sistema embebido desde ahora denominado concentrador es visto desde este como un dispositivo de memoria. El sistema embebido lee o escribe desde una dirección de memoria generando ciertas señales de control de esta recogiendo y enviando información al concentrador a través del bus de datos. Para aislar el concentrador y el bus de datos del sistema Linux embebido se usará un transceiver bidireccional.

3.1.2 Tarjeta de Desarrollo MINI2440.

Como podemos observar en la figura **3.1** la interfaz paralela de 16 bits tomará la información desde el bus de datos del sistema Linux embebido por lo que en la elección de este hemos considerado fundamental que nos proporcione tal bus a través de uno de sus conectores, además de ciertas señales de control que analizaremos detalladamente más adelante.

Considerando lo antes mencionado se decidió elegir la MINI2440 **(7)** mostrado en las figuras **3.2** y **3.3** , el cual es un sistema embebido chino, líder en el área automotriz, que nos permite tener lo antes mencionado a través de su conector CON5.

A) Características de hardware de la MINI2440 y procesador ARM S3C2440A.

POWER

5 voltios.

CPU

Samsung S3C2440A, 400MHz, Max. 533 MHz

SDRAM

64M SDRAM

32bit Bus de dato

SDRAM Clock 100MHz

Flash

256M Nand Flash,

2M Nor Flash,BIOS installed

LCD

Interface touchscreen 4 hilos resistivos.

Hasta 4096 color STN, 3.5 pulgadas a 12.1 pulgadas, hasta 1024x768 píxeles.

Hasta 64K color TFT, 3.5 pulgadas a 12.1 pulgadas, hasta 1024x768 píxeles.

Interfaces y recursos

1 10/100M Ethernet RJ-45(DM9000)

3 Puertos seriales

1 USB Host

1 USB Slave

1 SD Card Interface

1 Stereo Audio out, 1 Micro In:

1 20-Pin JTAG

4 leds de usuario

6 Botones de usuario

1 PWM Beeper

1 POT puede ser usado para ajustar conversor A/D

1 AT24C08 for I2C test

1 20-Pin Interface de cámara

1 Battery for RTC

Power In(5V)

Oscillator Frequency

12MHz

RTC

Internal

Interfaz

1 34-Pin 2.0mm GPIO

1 40-Pin 2.0mm Bus del sistema

Dimensiones

100 x 100(mm)

Sistemas operativos

Linux 2.6.32

Windows CE.Net 5.0

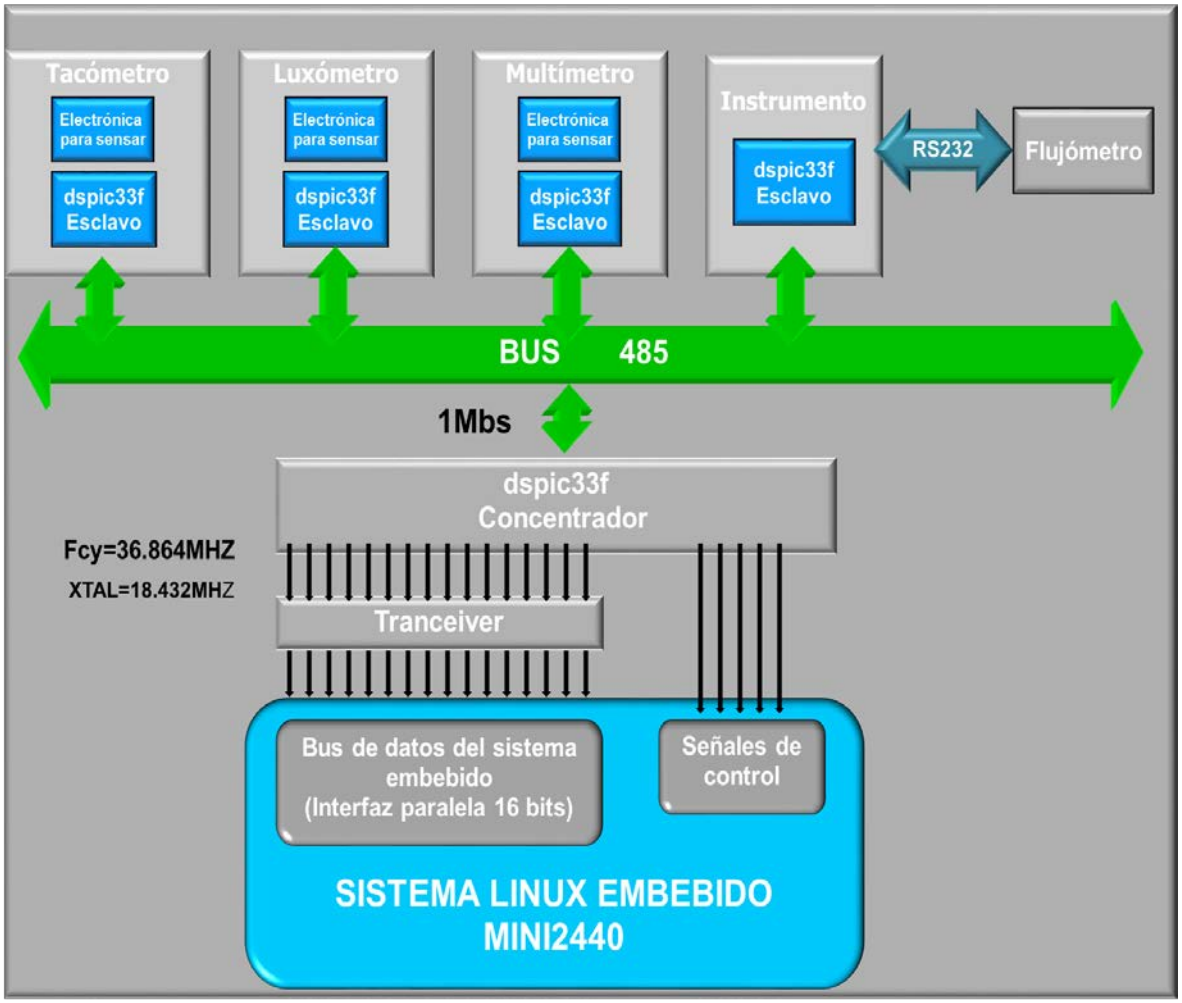


Figura 3.1 Sistema de Adquisición de datos (Fuente Propia)

El S3C2440A está desarrollado con el núcleo ARM920T mostrado en la figura **3.4** . Está compuesta por un núcleo ARM9TDMI más una unidad de gestión de memoria y memoria caché. ARM9TDMI es un dispositivo de arquitectura Harvard, implementado utilizando un pipeline de cinco etapas: fetch, decode, execute, data memory access, y write. Las cachés de instrucciones y datos tienen 16 KB cada una. ARM920T implementa el AMBA (Advanced Microcontroller Bus Architecture) que consiste de el bus AHB (Advanced High-performance Bus) y del bus APB (Advanced Pheriferical Bus).

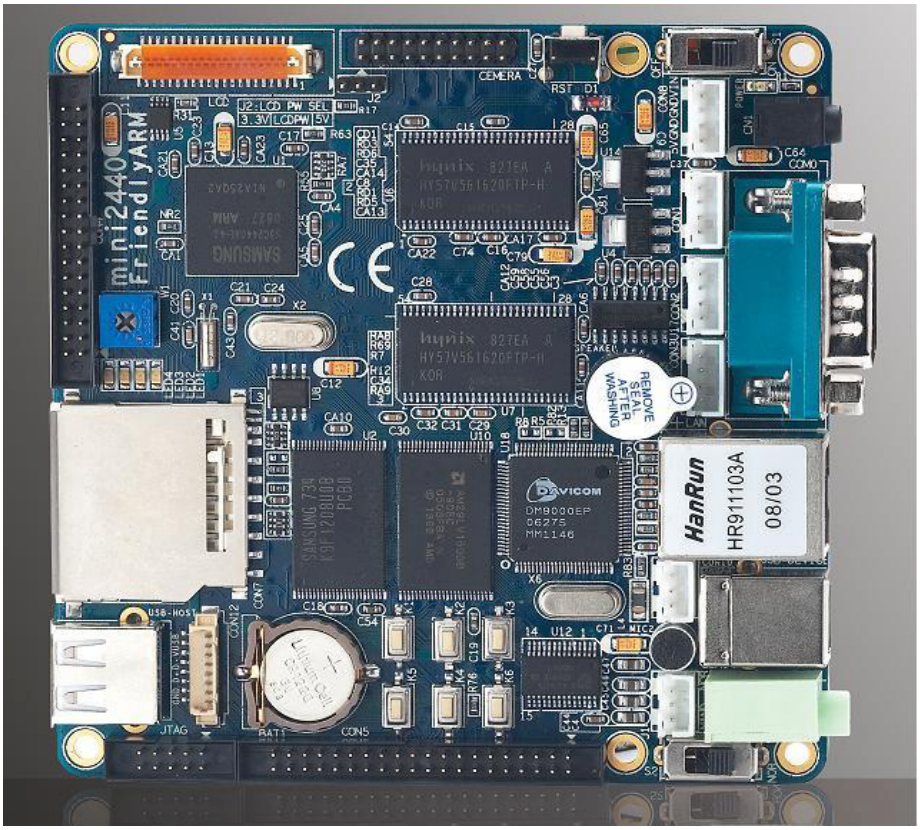


Figura 3.2 Vista Top de la Tarjeta (Fuente (7))

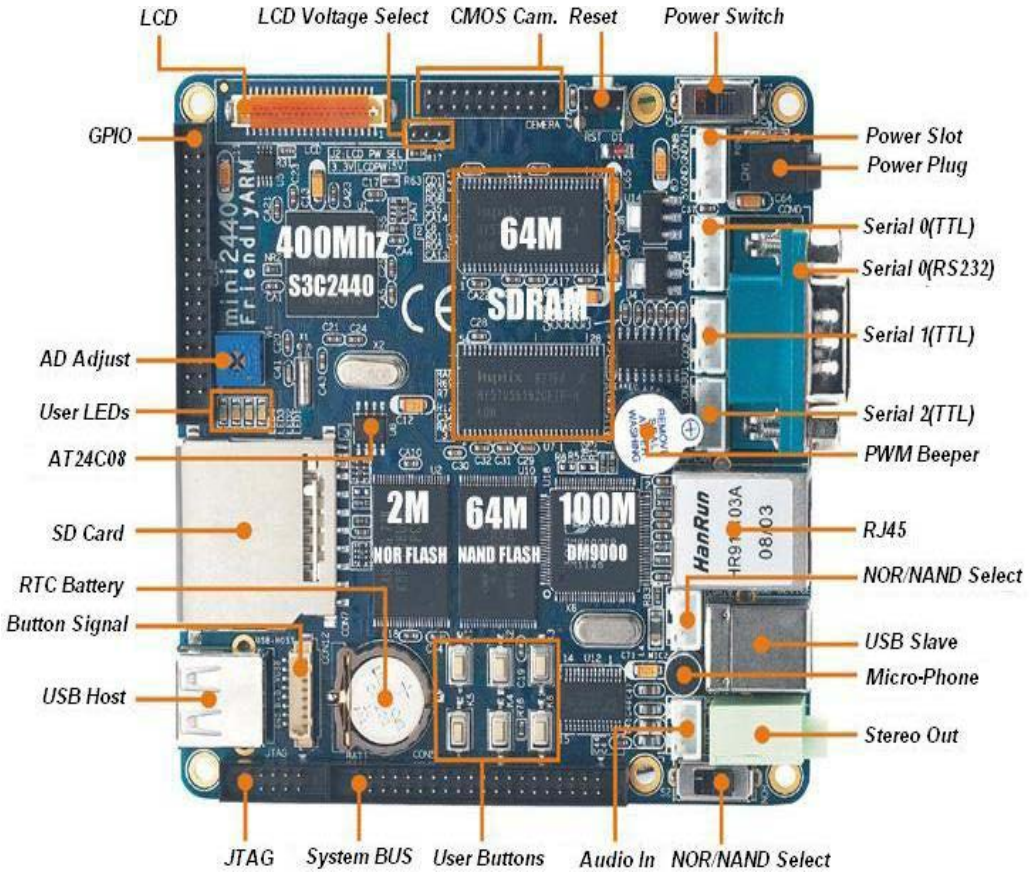


Figura 3.3 Interfaces de la Tarjeta (Fuente (7))

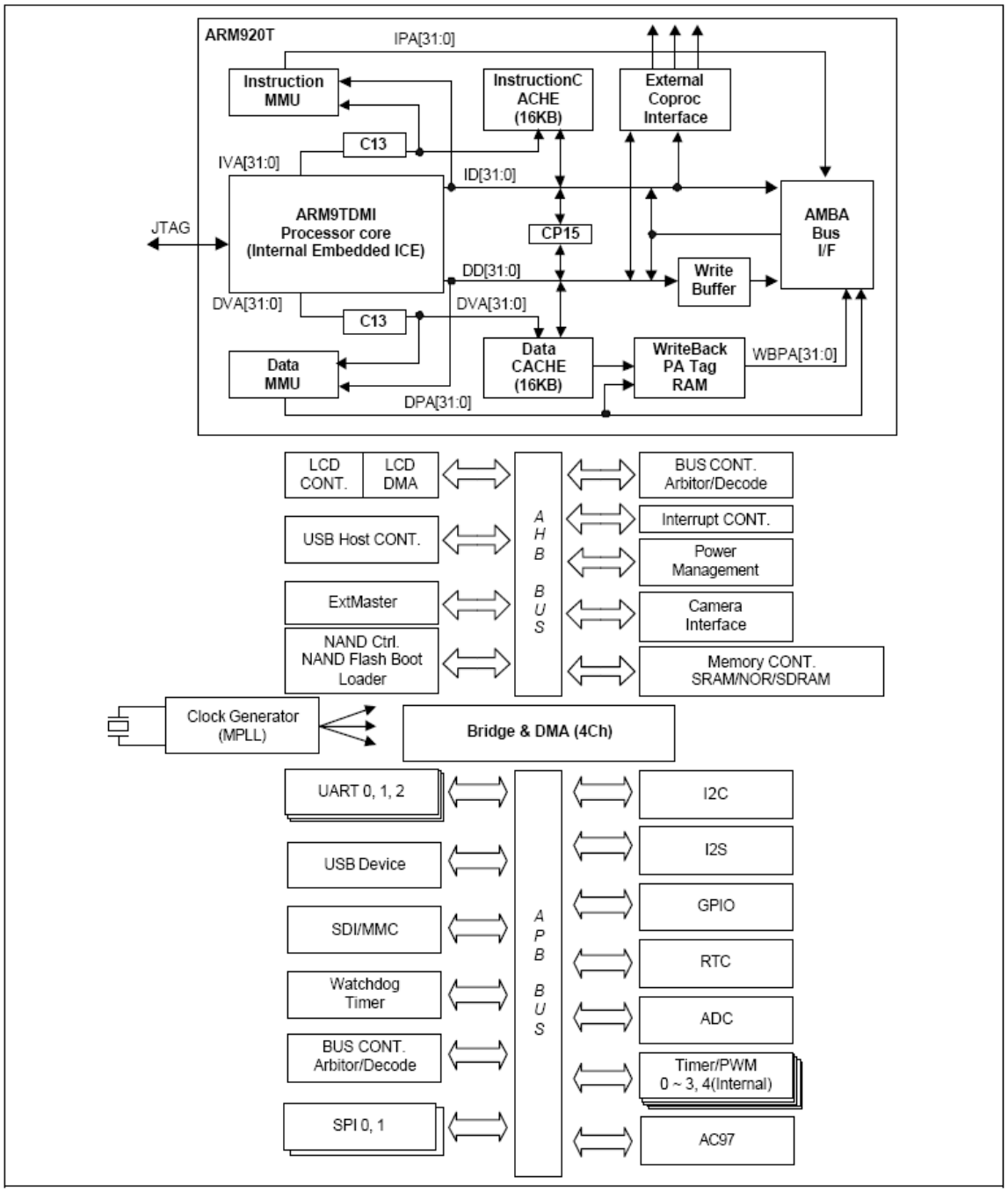


Figura 3.4 Diagrama Bloque S3C2440A (Fuente (11))

B) Mapeo de memoria y selección de chip de la MINI2440.

El S3C2440 soporta dos boot mode: Nand Flash Boot y Nor Flash Boot. El mapeo de memoria y la selección de chip select es diferente dependiendo del boot mode.

De la figura **3 .5** podemos observar ocho chip select desde el nGCS0 hasta el nGSC7, el nGCS6 está asociado al SDRAM.

Para poder hacer la transferencia de datos del concentrador al embebido debemos de elegir una dirección de memoria adecuada, de la figura **3 .5** el nGCS6 y nGCS7 están ocupados

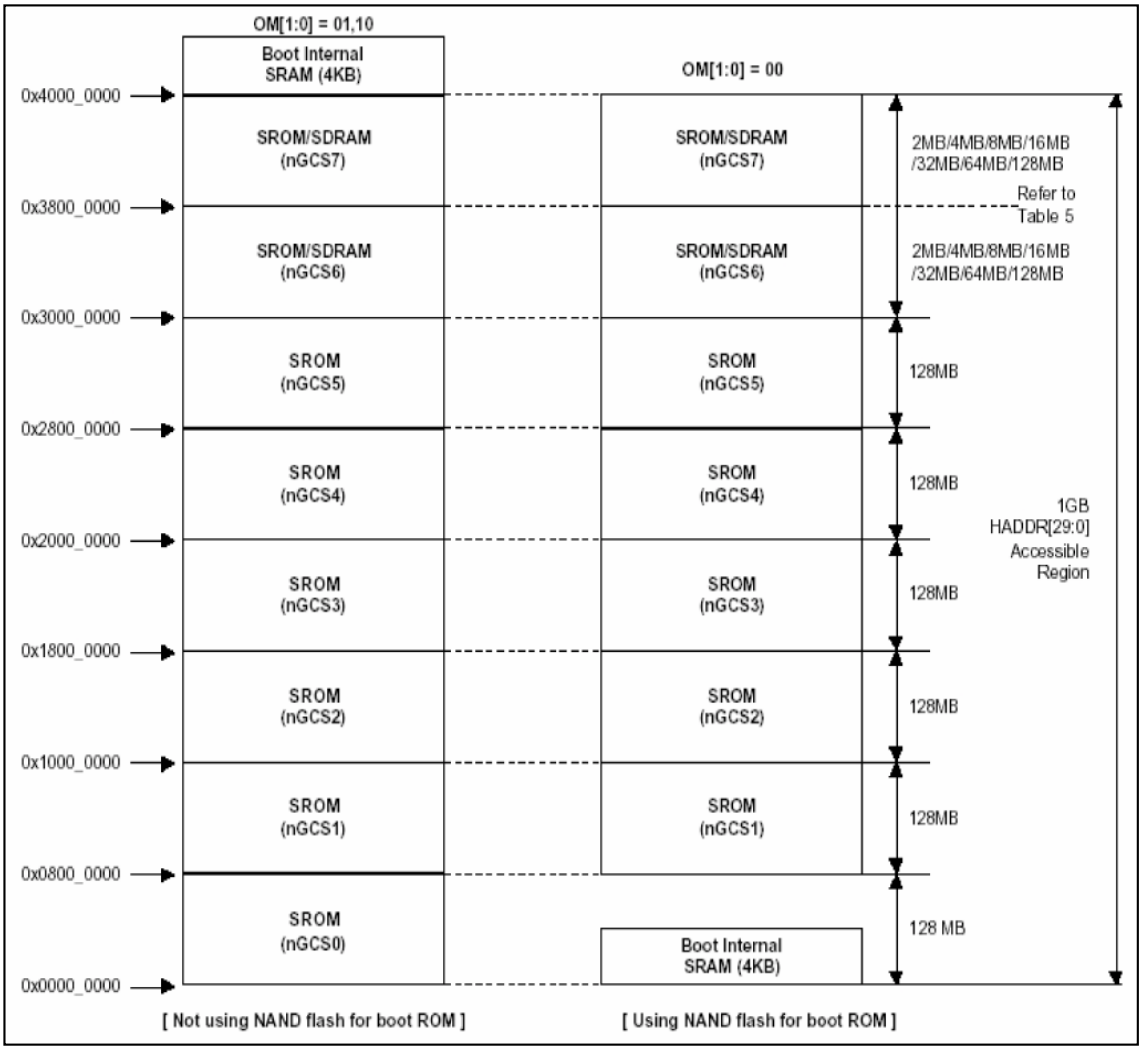


Figura 3.5 Mapeo Memoria y chip select (Fuente (11))

SDRAM address space: 0x30000000-0x34000000

por la SDRAM , pero observamos que el nGCS5 está disponible, de manera que seleccionaremos la dirección de memoria 0x28001000.

C) Controlador de memoria de la S3C2440A.

El controlador de memoria del S3C2440A provee señales de control de memoria las cuales son requeridas para un acceso de memoria externo.

El S3C2440A tiene las siguientes características:

Little/Big endian (seleccionable por software).

Espacio de dirección de 128Mbytes por banco (total 1Gb/8 bancos).

Tamaño de acceso programable (8/16/32) para todos los bancos excepto banco 0 (16/32).

Ocho bancos de memoria

Seis bancos de memoria para ROM , SRAM , etc.

Dos bancos de memoria para ROM, SRAM ,SDRAM , etc.

Ciclo de acceso programable para todos los bancos

Tacs : Address set-up time before nGCSn

Tcos : Chip selection set-up time before nOE

Tacc : Access cycle

Tcoh : Chip selection hold time after nOE

Tcah : Address hold time after nGCSn

Wait externo para prolongar el ciclo de lectura/escritura.

nWait Pin Operation:

Si el WAIT bit (WSn bit in BWSCON) correspondiente a cada banco de memoria es habilitado, la duración del OE (acción lectura) debería ser prolongada por el pin externo nWait mientras el banco de memoria está activado. nWait es checkeado desde tacc-1. nOE será liberado en el siguiente clock después que la muestra nWAIT esté en nivel alto.

Por defecto el tacc es de 14 clocks, siendo HCLK 100MHZ , entonces tacc toma un valor de 140ns.

A pesar de que el dspic usa el PLL para obtener un frecuencia de ciclo de instrucción (Fcy) de 36.864MHz sigue siendo muy lento en comparación con el HCLK, por lo que usaremos la señal pin externo Wait para prolongar el ciclo de Lectura/Escritura de tal forma que el dspic33f tenga el tiempo suficiente para realizar las instrucciones necesarias propias del programa que se detallará en el capítulo correspondiente antes de leer o escribir. El dspic33f tendrá la responsabilidad de liberar el wait.

La señales de control utilizadas por la interfaz paralela para controlar el acceso a memoria son:

"nCS" --> Es generada cada vez que el embebido realiza una instrucción de Lectura o Escritura de memoria del banco 5 usado.

"nWE" --> Es generada cada vez que el embebido realiza una instrucción de Escritura de memoria del banco 5 usado.

"nOE" --> Es generada cada vez que el embebido realiza una instrucción de Lectura de memoria del banco 5 usado.

Los diagramas de tiempo de ciclo de Lectura/Escritura son mostrados en las figuras **3 .6** , **3 .7** , **3 .8** y **3 .9.**

Nuestra interfaz paralela estará acorde con los diagramas de las figuras y **3 .9** puesto que como lo hemos mencionada usaremos la señal de Wait, mientras que en las figuras **3 .6** y **3 .7** se muestra el diagrama sin la señal Wait.

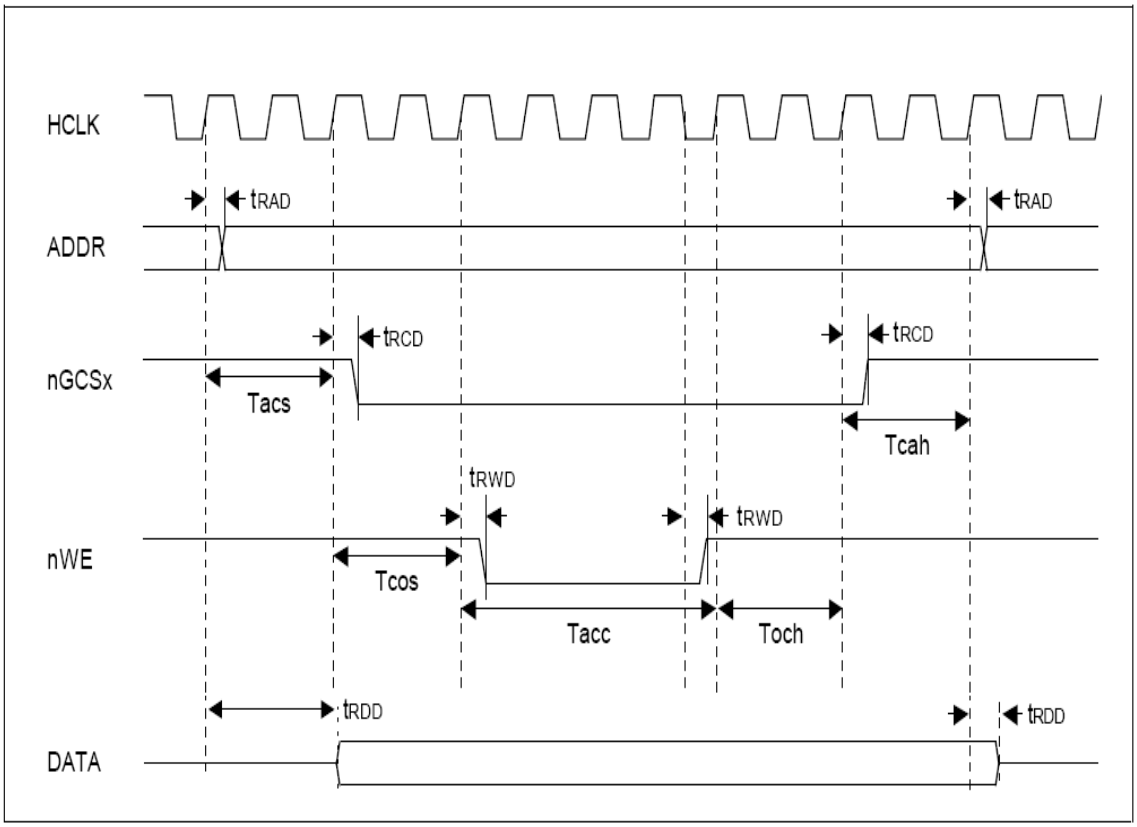


Figura 3.6 Write Timing Diagram (Fuente (11))

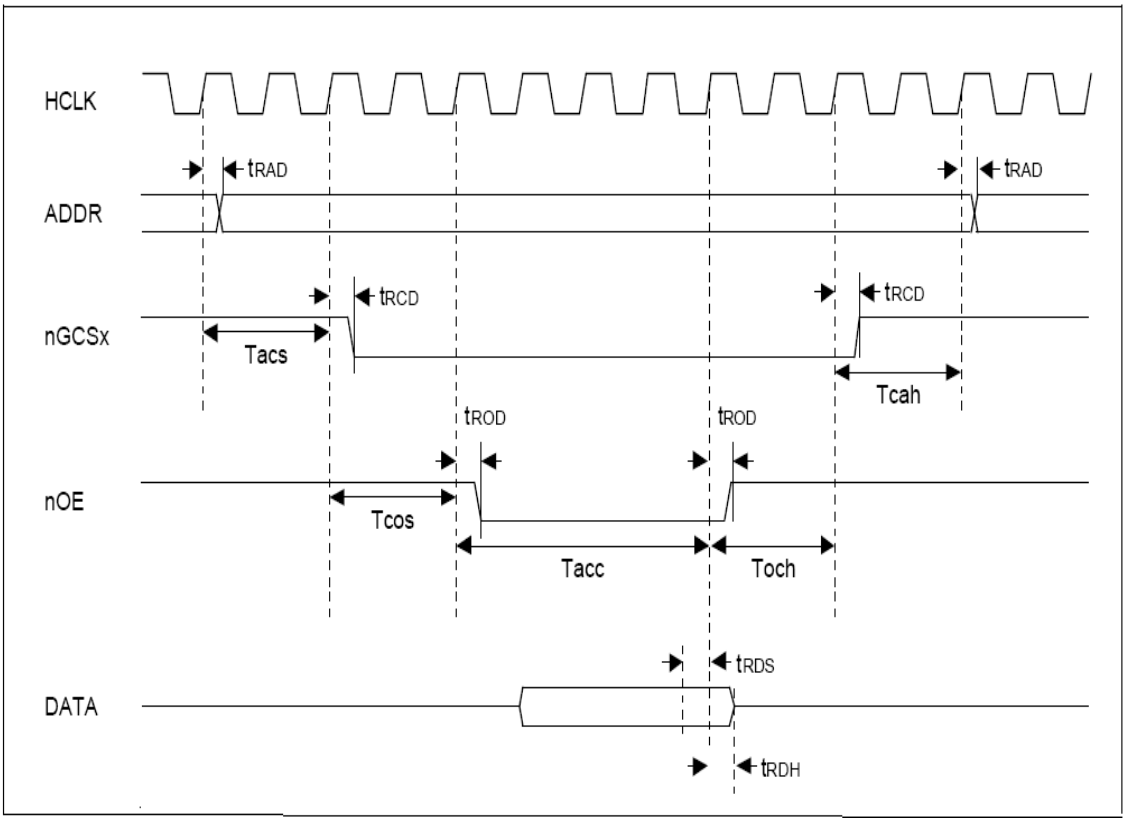


Figura 3.7 Read Timing Diagrama (Fuente (11))

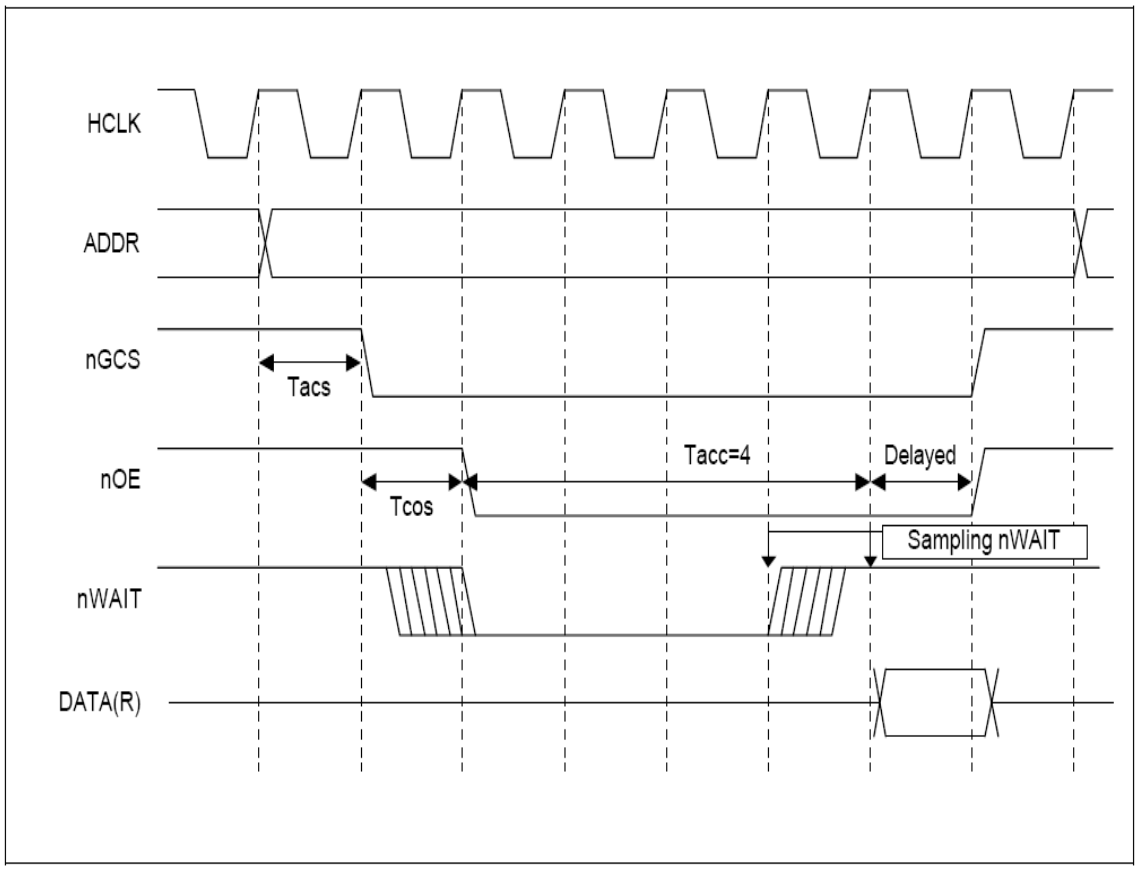


Figura 3.8 External nWait Read Timing Diagram (Fuente (11))

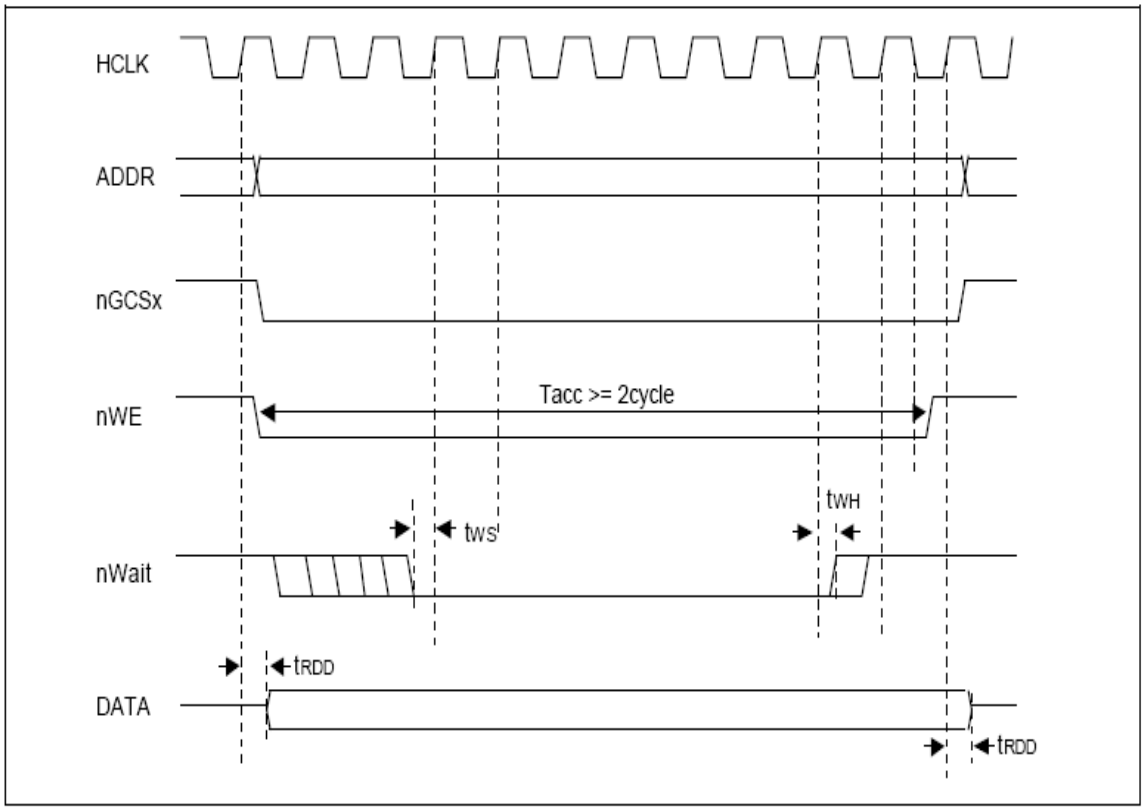


Figura 3.9 External nWait Write Timing Diagram (Fuente (11))

D) Conector CON5 de la MINI2440

La figura **3 .10** muestra el Conector CON5 de la tarjeta MINI2440 del cual utilizaremos el bus de dato desde LDATA0 a LDATA15 para la generación de nuestra interfaz paralela de 16 bits. Como se mostró en la figura **3.1** los LDATA se conectarán a un transceiver para poder aislar el bus de datos del concentrador cuando no haya ninguna actividad de lectura/escritura en el banco 5.

Del CON5 nos interesa además del bus de datos señales nGCS5 , LnWE , LnOE ,nWAIT vistas anteriormente.

Cabe resaltar que el CON5 nos proporciona pines de interrupción externa, siendo el EINT3 clave en la elaboración del driver que manejará la interfaz paralela y que en su momento detallaremos.

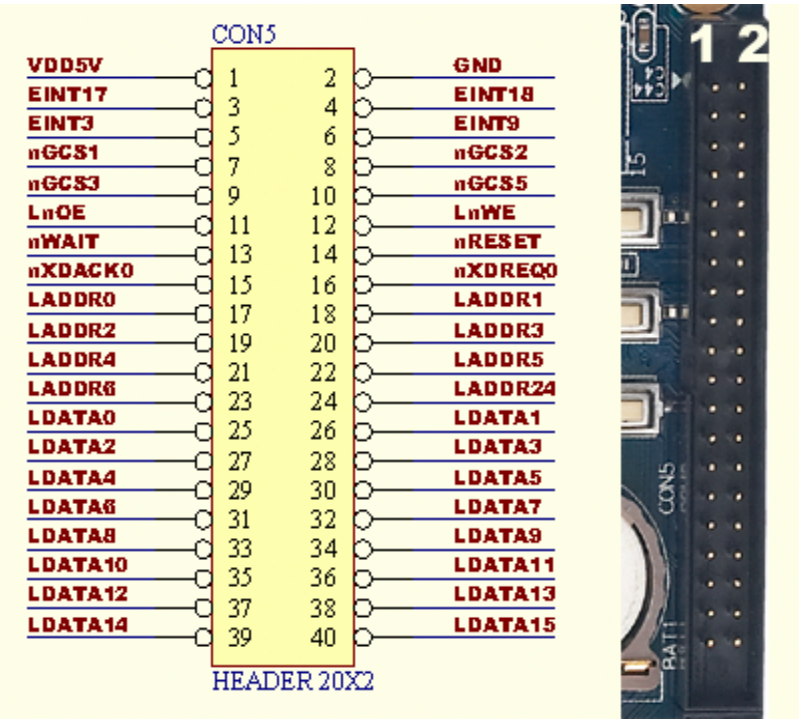


Figura 3.10 Conector CON5 de la MINI2440 (Fuente (7))

3.2 Interfaz Gráfica de Usuario.

3.2.1 Introducción.

El equipo que se desea diseñar deberá contar con una interfaz gráfica de usuario que nos permita diseñar las OTM personalizadas, además de los programas gráficos asociados a los instrumentos de medición.

El equipo deberá contar con un ambiente gráfico orientado a un PDA, por tal motivo se decidió usar como GUI Qtopia en su edición PDA, la cual personalizaremos a nuestros requerimientos.

3.2.2 Plataforma de Aplicación Integral.

Qtopia es la plataforma de aplicación integral de Trolltech para PDA basado en Linux embebido para teléfonos móviles, y otros dispositivos electrónicos.

Qtopia es ambos una plataforma y una serie de productos destinados a facilitar la creación de interfaz gráfica de usuarios por parte de los programadores.

Trolltech ofrece dos categorías de licencias para Qtopia:

1. Versión comercial
2. Libre (GPL), versión

Las versiones comerciales de Qtopia están destinadas a la creación de software comercial, propietario, con un número de licencias.

La versión gratuita de Qtopia es la intención de apoyar y fomentar el desarrollo de software libre.

Qtopia ofrece dos ediciones:

1. Qtopia Phone

Qtopia Phone Edition para Linux embebido está diseñado para teléfonos inteligentes, teléfonos con funciones especiales. Ofrece una amplia gama de mensajería y herramientas para entretenimiento y las combina con las funciones del teléfono, ahora estándar.

2. Qtopia PDA

Edición PDA Qtopia (Asistentes Personales Digitales) ha sido diseñada para PDAs con sistema operativo basado en Linux ofreciendo la mayor parte de las mismas herramientas personales y de entretenimiento como el Phone Edition, la edición PDA amplía la gama de herramientas de productividad y toma ventaja de las funcionalidades mejoradas del dispositivo. En la figura **3 .11** se muestra Qtopia en su edición PDA.

Como un sistema, Qtopia, ofrece las siguientes características:

Ambiente de desarrollo.

Juegos y multimedia.

Métodos de Entrada.

Aplicaciones de Internet.

Integración Java.

Opciones de personalización.

PIM applications.

Aplicaciones de productividad.

Framework de sincronización.

Sistema de ventanas.

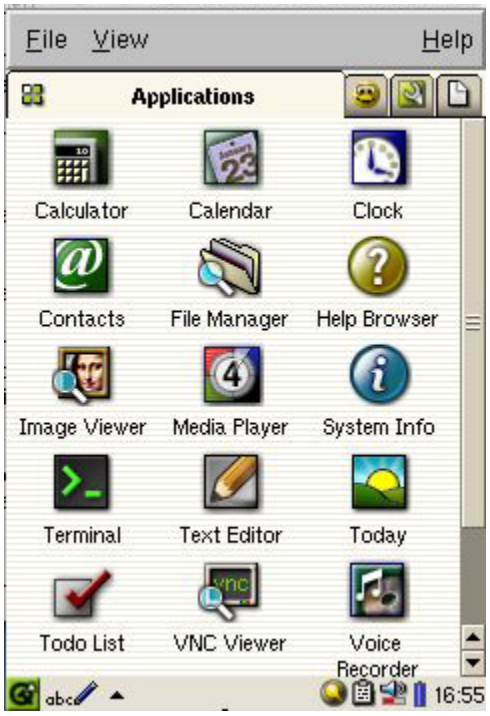


Figura 3.11 Qtopia Edition PDA (Fuente (9))

Qtopia es disponible también en un número de diferentes paquetes. Un paquete es una combinación de Qtopia más otros productos como por ejemplo las QT.

Dependiendo del paquete seleccionado se podrá contar con el código fuente completo o un subconjunto con binarios pre-construidos. Los paquetes pueden ser comerciales o libres.

Dentro de los paquetes comerciales se encuentran :

Qtopia OEM Kit.

El Qtopia OEM Kit involucra un número de licencias y el código completo de Qtopia permitiendo la personalización de las aplicaciones de Qtopia.

Qtopia Software Development Kit (SDK) Packages .

Presenta dos opciones :

Qtopia SDK Light.

Qtopia SDK Professional .

Ambos paquetes proveen binarios preconstruidos del producto Qtopia y archivos de cabecera. El paquete profesional proporciona parte del código fuente de Qtopia y un contrato de soporte.

Los binarios de Qtopia no pueden ser reconstruidos con esos paquetes.

Dentro del paquete libre se destaca:

Qtopia PDA GPL edition

Es disponible bajo la licencia GPL. Actualmente esta versión está basada en la edición PDA y está destinado a apoyar a la comunidad Open Source.

La plataforma Qtopia comprende:

Qtopia libraries (Qt/E, libqpe, libqtopia1, qtopiapim , etc)

Qtopia server/launcher.

Los programadores de aplicaciones deberían usar esas librerías cuando escriban aplicaciones para un dispositivo Qtopia.

El launcher/server de Qtopia es el proceso principal que controla el sistema de ventanas, comunicación entre procesos, disparar todos los procesos y otras tareas centrales.

CAPÍTULO IV
PROTOCOLO DE COMUNICACIÓN

4.1 Introducción.

Como se mencionó en el capítulo anterior el sistema de adquisición de dato queda dividido en dos bloques: la comunicación entre le concentrador y los instrumentos y la comunicación entre el concentrador y el sistema embebido.

Para lograr una transferencia de datos entre el concentrador e instrumentos o concentrador y sistema embebido se deberán seguir los protocolos que detallaremos a continuación.

De ahora en adelante nos referiremos a los programas de los dspic33f esclavos como programa de instrumentos, mientras que a los programas gráficos para los instrumentos de medición mostrados en el LCD de la mini2440 como programas gráfico de instrumentos.

4.2 Protocolo de comunicación entre el sistema embebido y el concentrador.

La comunicación entre los programas gráficos de instrumentos y los instrumentos es bidireccional y es realizada en bloques que se transfieren desde la interface paralela del embebido hacia el concentrador y viceversa, llegando finalmente a los instrumentos los cuales están colgados al concentrador a través del bus 485.

Esta transferencia de bloques es realizada de manera diferente dependiendo de quién inicie la comunicación, el concentrador o el embebido, así:

Concentrador inicia transferencia de un bloque.- Este inicia la comunicación generando una interrupción al embebido para lograr su atención, en respuesta el embebido inicia la siguiente secuencia:

1. Lee la primera palabra entregada por el concentrador en la cual se especifica los parámetros que definen el bloque que se está transfiriendo.

Como observamos en la tabla 4.1 los primeros ocho bits nos especifican el número de words a transferir, el bit 9 especifica si los datos son programados o no.

Consideraremos como datos programados aquellos que son enviados con un periodo configurado mientras que los no programados serán enviados aleatoriamente producto de una previa petición. Los datos programados por lo general serán utilizados para la

generación de curvas en el tiempo mientras un dato no programado podría ser por ejemplo la solicitud de tendencias de ciertos parámetros.

TABLA Nº4.1 Cabecera de información

Bits	Valor	Descripción	Observaciones
15	1	Instrumento	-
	0	Concentrador	-
14	1	Bloque de tipo: Datos	-
	0	Bloque de Tipo: Estado	-
13	1	Bloque generado para el canal 1	Solo uno de estos bits puede estar en "1"
	0		
12	1	Bloque generado para el canal 2	
	0		
11	1	Bloque generado para el canal 3	
	0		
10	1	Bloque generado para el canal 4	
	0		
9	1	Bloque de Datos a Pedido	-
	0	Bloque de Datos programado	
8	0,1	Especifica el número de words (de 16 Bits) en datos o estado que se van a transferir en el bloque.	Máximo 512 words
7	0,1		
6	0,1		
5	0,1		
4	0,1		
3	0,1		
2	0,1		
1	0,1		
0	0,1		

Los bit 13 al 10 definen el canal de comunicación, estos representan canales lógicos que nos permitirán ubicar los datos recibidos desde el concentrador a la memoria compartida que es vista por el programa de comunicaciones (que maneja la interfaz paralela) y los programas gráficos de los instrumentos.

El bit 14 específica si se trata de una transferencia de dato o estado. Al referirnos de datos

estamos hablando de datos programados o no programados mientras que el estado se refiere a un estado del instrumento o del propio concentrador.

El bit 15 hace referencia que los datos a recibir son por parte del concentrador o del instrumento.

2. El embebido a continuación lee del concentrador el número de words especificados en el comando anterior un correlativo y un "Checksum" .

(*) El cálculo del checksum involucra a todos los words transferidos incluyendo el correlativo.

3.Seguidamente el embebido luego de chequear la integridad del bloque recibido y dependiendo si tiene un comando pendiente de transferir al concentrador, envía una de las siguientes tres respuestas repuestas:

a) Respuesta 1.- Un "ACK" (0x2bcd) con el bit 15 en cero, indicándole al concentrador que el bloque ha sido recibido correctamente y que la transferencia concluye.

b) Respuesta 2.- Un "ACK" con el bit 15 en Uno, indicándole al concentrador que el bloque ha sido recibido correctamente y que la comunicación continua con el envió de un comando al concentrador o programa de instrumentos, luego del cual la comunicación termina incondicionalmente, independiente que este comando haya sido reconocido con "ACK" o un "NACK".

c) Respuesta 3.- Un "NACK" (0x7234), indicándole al concentrador que hay un error en el bloque recibido. El error será reportado por el programa de comunicaciones. Cuando se ingresa en esta secuencia ningún comando será enviado, sino que se tendrá que espera a una nueva interrupción par seguir de nuevo todo el proceso.

Embebido inicia la comunicación.- Cuando el embebido envía bloques de comandos al concentrador ó bloques de comandos a los instrumentos, lo hace de dos modos que depende del estado actual de la comunicación. Un primer modo es cuando no existe algún instrumento enviando bloques de datos o estado al embebido, en dicho caso el embebido envía el bloque correspondiente incondicionalmente. El segundo modo es cuando existe por lo menos un instrumento enviando bloques de datos o estado al embebido en cuyo caso el comando es enviado por el embebido inmediatamente después de la recepción de un bloque de datos.

En ambos casos el envió del bloque por el embebido tiene la siguiente secuencia:

1. Envía la primera palabra al concentrador, la cual define los parámetros del bloque que se está transmitiendo:

TABLA Nº4.2 Cabecera de información

BITS	Valor	Descripción	Observaciones
15	1	Instrumento.	-
	0	Concentrador.	-
14	1	Por definir	-
	0	Por definir	-
13	1	Bloque enviado para canal 1	Solo uno de estos bits puede estar en "1"
	0		
12	1	Bloque enviado para canal 2	
	0		
11	1	Bloque enviado para canal 3	
	0		
10	1	Bloque enviado para canal 4	
	0		
9	1	Por definir	-
	0	Por definir	-
8	0,1	Especifica el número de Word (de 16 Bits) que se van a transferir.	-
7	0,1		
6	0,1		
5	0,1		
4	0,1		
3	0,1		
2	0,1		
1	0,1		
0	0,1		

Como se observa en la tabla **4.2** el embebido envía al concentrador especificando en el bit 15 si se trata de una información dirigida hacia el concentrador para la programación de un canal o de un comando hacia uno de los instrumentos. El bit 10, 11, 12 y 13 especifica el canal de comunicaciones, mientras que lo últimos 9 bits el número de words a transferir.

2. A continuación el embebido envía al concentrador el número de Word especificados previamente en el paso anterior, más un Word con el "Checksum".

(*) El cálculo del checksum involucra a todos los words transferidos incluyendo el

comando.

3. El concentrador finaliza la recepción del bloque enviando al embebido el reconocimiento que la transferencia ha sido correcta con un "ACK" ó indicando que hay error de transferencia con un "NACK".

La comunicación entre el sistema embebido y el concentrador requiere de dos programas que son:

Programa de Comunicaciones: Programa que corre en el sistema embebido encargado de administrar la transferencia de entrada y salida de bloques de comandos, datos y estado entre el embebido y el concentrador.

Programa del Concentrador: Programa que corre en el concentrador que se encarga de administrar la transferencia de entrada y salida de bloques de comandos, datos y estado entre el concentrador y el embebido.

Ambos programas realizan funciones complementarias, las mismas que se han implementado para lograr una comunicación transparente de bloques de comandos, datos y estado entre los Programas gráficos de Instrumentos y los Programas de Instrumentos.

El desarrollo del sistema requiere definir los siguientes arreglos de words de 16 bits que deben existir en ambos programas para manejar hasta cuatro canales de comunicación concurrentes, estos arreglos podemos diferenciarlos en dos grupos como sigue:

1) Arreglos relacionados a la comunicación entre los Programas de Instrumentos y sus Programas gráficos de Instrumentos respectivos.

Cuatro arreglos de datos programados de 512 words c/u declarados consecutivos, uno por cada canal y/o instrumento.

Cuatro arreglos de datos no programados de 512 words c/u declarados consecutivos, uno por cada canal y/o instrumento.

Cuatro arreglos de Comandos de 100 Words c/u declarados consecutivos, uno por cada canal y/o instrumento.

Cuatro arreglos de Estado de 25 words c/u declarados consecutivos uno para cada canal y/o instrumento.

2) Arreglos relacionados al concentrador para el manejo y control de los cuatro canales de comunicación:

Cuatro arreglos de comandos de 6 words c/u declarados consecutivos donde se definen los parámetros de comunicación de cada canal.

Cuatro arreglos de Estado de 6 words c/u declarados consecutivos donde se reporta el

estado de comunicación de cada canal.

TABLA Nº4.3 Definición del Comando al Concentrador

Word	Byte	Descripción
0	0	Activa /Desactiva Instrumento en canal "n": 0: Desactiva Instrumento 1-255: Activar Instrumento numero # (de 1 a 255)
	1	
1	2	Tiempo de Muestreo con que se programa al instrumento para el canal "n" respectivo. Este número representa el tiempo (expresado en Tic´s de 2ms) en que el instrumento entrega datos al concentrador.
	3	
2	4	Número de Datos por Muestra o Frame. Este número expresa el número de Words de 16 Bits transferidos del Instrumento al concentrador.
	5	
3	6	Número de Frames que contiene el buffer para ser enviado al Embebido. (Este número multiplicado por el número de datos por nuestra ó Frame nos da la longitud del buffers que trasfiere el concentrador al Embebido).
	7	
4	8	No definido
	9	No definido
5	10	No definido
	11	No definido

Para poder programar un instrumento lo primero que se debe hacer es enviar el comando especificado en la Tabla 4.3 en donde se establece los parámetros de funcionamiento como son el número del instrumento que deseamos direccionar, el tiempo de muestreo expresados en ticks de 2ms con el cual el instrumento comienza a tomar datos, el número de datos por frame que recolecta el instrumentos y por último el número de frames a

transmitir. En total son 6 words a enviar de los cuales los dos últimos se reservarán para un futuro.

De la Tabla 4.4 observamos el Estado del concentrador el cual es enviado cada vez que se envíe un comando al concentrador con la finalidad de especificar si el instrumento se logró hallar o si el canal por donde se quiso programar un determinado instrumento esta libre o

TABLA Nº4.4 Definición del Estado del Concentrador por cada canal

Word	Byte	Descripción	Observaciones
0	0 1	Estado del instrumento del canal "n": 0: Instrumento no hallado #(1-255): Instrumento numero(#) ubicado	Instrumento ubicado y programado
1	2 3	Estado del canal "n": 0: Canal libre o desocupado. #(1-255): Instrumento numero(#) programado en el canal.	Canal programado pero no necesariamente el instrumento.
2	4 5	Siempre en valor "1" cuando el concentrador envía el Estado al embebido para indicar nuevo Estado.	El Programa del Instrumento clarifica a cero al leer el estatus
3	6 7	No me comunico con el instrumento. El instrumento a dejado de enviar información Una posible causa es la desconexión del instrumento	Sólo para información del programa del instrumento correspondiente.
4	8 9	Concentrador se recupera de un reset. "1": El concentrador reprogramó todos los canales luego de un reset, y sigue operando normalmente. "0": El concentrador está funcionando sin ningún problema.	El programa de instrumento deberá señalizar con un mensaje tal condición.
5	10 11	No definido No definido	Ninguna

ocupado. Además otra información que este Estado nos proporciona es indicar si el concentrador se recupera de un reset.

El sistema en desarrollo requiere aparte de los arreglos mencionados unos uno flags para

poder discriminar el tipo de información, dichos flags los podemos agrupar en:

Flags para Estado del Concentrador:

Un Word indicando que tenemos un Estado del Concentrador correspondiente a un canal previamente programado.

Flag para Datos no Programados:

Un Word indicando que tenemos Datos no Programados correspondiente a un determinado instrumento.

Flag de Estado de los Instrumentos:

Un Word indicando que tenemos un Estado correspondiente a un determinado instrumento.

4.3 Protocolo de comunicación entre el concentrador y los instrumentos.

Esta comunicación es realizada por Pooling por el concentrador con cada uno de los instrumentos, y la información que se transfiere es la siguiente:

Comandos: Los comandos son bloques de datos trasferidos al instrumento que provienen del concentrador o del programa gráfico del instrumento.

Datos y Estado: Son bloques de datos o Estado transferidos desde el Instrumento hacia el programa gráfico del instrumento.

Procedimiento:

1. El concentrador mediante Pooling cada 2ms enviará un Byte que contendrá la dirección del instrumento con el que desea comunicarse.
2. Luego del reconocimiento del instrumento durante el Pooling el concentrador envía un word para el caso que este enviando comandos al instrumento, y solo el primer byte del word de la Tabla **4.5** , en caso esté requiriendo lectura de datos, Estado o reinicio del instrumento:

Nota: El word es enviado con el byte más significativo primero.

Si se está enviando un comando del Programa Gráfico de Instrumento al instrumento, a continuación el concentrador envía al instrumento el número de words especificados previamente en e l paso anterior, más un Word con el "Checksum".

(*) El cálculo del checksum involucra a todos los words transferidos incluyendo la palabra de control inicial.

Si se está enviando un comando del concentrador al instrumento, para programar la transferencia de datos periódica, el concentrador envía las siguientes dos palabras a continuación:

TABLA N°4.5 Word de Pooling

BITS	Valor	Descripción	Observaciones
15	1	Comando del Programa Gráfico del instrumento al instrumento.	
	0		
14	1	Comando del Concentrador al Instrumento.	Solo uno de estos bits puede estar en "1"
	0		
13	1	Lectura de Datos o Estado	
	0		
12	1	Reinicia Instrumento (Reset)	
	0		
11	1	Por definir	
	0	Por definir	
10	1	Por definir	
	0	Por definir	
9	1	Por definir	
	0	Por definir	
8	0,1	Especifica el número de words (de 16 Bits) que se van a transferir.	Valor válido para comandos (En caso del comando del concentrador son dos words, un divisor y palabras por frame)
7	0,1		
6	0,1		
5	0,1		
4	0,1		
3	0,1		
2	0,1		
1	0,1		
0	0,1		

a) Una palabra con el Tiempo de Muestreo que usará el instrumento en la adquisición de su data. Este número representa un tiempo (expresado en Tic´s. de 2ms).

b) Una palabra con el Número de Datos por Muestra o Frame, que enviará periódicamente al programa del instrumento respectivo.

Adicionalmente una palabra de Checksum para preservar la información y mantener el procedimiento de comandos como en el caso anterior.

TABLA Nº4.6 Word de Respuesta del Instrumento

BITS	Valor	Descripción	Observaciones
15	1	Bloque de datos programados al Programa del Instrumento.	Solo uno de estos bits puede estar en "1"
	0		
14	1	Bloque de datos al Programa del Instrumento.	
	0		
13	1	Bloque de Estado al Programa del Instrumento.	
	0		
12	1	No hay bloques - Preguntar en "N" Tic´s	
	0		
11	1	Reconocimiento de Reset	
	0		
10	1	Por definir	
	0	Por definir	
9	1	Por definir	
	0	Por definir	
8	0,1	Especifica el número de words (de 16 Bits) en datos o estado que se van a transferir en el bloque.	Especifica el número de words a transferir
7	0,1		
6	0,1		
5	0,1		
4	0,1		
3	0,1		
2	0,1		
1	0,1		
0	0,1		

Si es el caso de una lectura de Datos o Estado y el instrumento aludido tiene datos, este responde con un word de 16 bits como el que se muestra en la Tabla **4.6** , en el caso que el instrumento no tiene bloque para enviar o está reconociendo un reset responde con el byte más significativo del word mostrado a continuación:

A continuación si el instrumento tiene un bloque para el concentrador, transfiere el número de Word especificados previamente en el paso anterior, mas dos words con el "Tiempo",

más un word con un correlativo, mas el "Checksum" que incluye a todos los words enviados.

Si el instrumento responde en la palabra de control con el Byte más significativo que no tiene bloque disponible para transferir al concentrador, se da por concluida la comunicación.

Si el instrumento responde en la palabra de control que ha reconocido el Reset o reinicio del instrumento, se da por concluida la comunicación.

CAPÍTULO V
DISEÑO DE APLICACIONES GRÁFICAS

5.1 Introducción.

En este capítulo se detallarán los pasos a seguir para instalar en nuestro sistema embebido la plataforma Qtopia **(9)** para poder ejecutar los programas gráficos de los instrumentos de medición y de las ordenes de trabajo.

Los programas gráficos de los instrumentos de medición implementarán una GUI a través de la cual se mostrarán ciertos parámetros y curvas en el tiempo. Para visualizar en nuestro programa gráfico valores reales diseñaremos una tarjeta electrónica que se comunicará con un flujómetro de masa a través del estándar rs232.

El flujómetro de masa enviará los parámetros de flujo , presión y temperatura a una tasa de 38400 baudios.

El programa gráfico para el flujómetro de masa se tomará como referencia para desarrollar aplicaciones gráficas de otros instrumentos de medición.

El programa de órdenes de trabajo de mantenimiento construirán los ítems relacionadas al mantenimiento de un equipo específico, además de interactuar con los programas gráficos de los instrumentos de medición los cuales le enviarán los datos sensados.

5.2 Instalación de la Plataforma de Aplicación Integral.

Para poder instalar qtopia debemos primero de cross-compilar la librería Tslib. Tslib es generalmente usado en dispositivos embebidos que provee un interface en el espacio usuario para el manejo de touchscreen. El código fuente puede ser descargado desde https://github.com/kergoth/tslib .

Descargamos el cross-compilador arm-linux-gcc-4.3.2 desde:
http://www.friendlyarm.net/downloads .

Descomprimimos arm-linux-gcc-4.3.2.tgz, obtendremos una carpeta dentro de la cual obtendremos nuestro cross-compilador en la ruta /usr/local/arm/4.3.2/bin.

Consideraremos que estamos trabajando en Linux usando la distribución fedora.

Descomprimimos kergoth-tslib-1.0-106-gf6c499a.zip

mv kergoth-tslib-f6c499a tslib

cd /home/carlos /Desktop/tslib

setenv PATH /usr/local/arm/4.3.2/bin:$PATH

setenv ac_cv_func_malloc_0_nonull yes

./autogen.sh

./configure CC=arm-linux-gcc CFLAGS="-O0 -msoft-float -D__GCC_FLOAT_NOT_NEEDED -march=armv4 -mtune=arm920t" CXX="arm-linux-g++ -O0 -msoft-float -D__GCC_FLOAT_NOT_NEEDED -march=armv4 -mtune=arm920t" -host=arm-linux -target=arm-linux -enable-static=no -enable-shared=yes -prefix /usr/local

make

make install //como su

En la ruta /usr/local/ encontraremos las carpetas bin, etc , include, lib. Copiamos todas estas carpetas al directorio /usr/local de la mini2440.

Con esto ya tendremos instalados la tslib , sin embargo falta configurar algunas variables de entorno para su correcto funcionamiento que las veremos más adelante.

Para compilar la plataforma qtopia descargaremos el código fuente desde http://www.friendlyarm.net/downloads .

La carpeta arm-qtopia contendrá el código fuente de qtopia.

cd arm-qtopia

./build

El script build compilará e instalará qtopia en la ruta arm-qtopia/qtopia-2.2.0-FriendlyARM/qtopia/images

El contenido de images es lo que debemos colocar en nuestro sistema embebido y tendremos instalado qtopia.

Para poder correr qtopia debemos de configurar las siguientes variables de entorno:

export TSLIB_TSDEVICE=/dev/input/event0

export TSLIB_CONFFILE=/usr/local/etc/ts.conf

export TSLIB_PLUGINDIR=/usr/local/lib/ts

export TSLIB_CALIBFILE=/etc/pointercal

export QTDIR=/opt/Qtopia

export QPEDIR=/opt/Qtopia

export PATH=$QTDIR/bin:$PATH

export LD_LIBRARY_PATH=$QTDIR/lib:/usr/local/lib:$LD_LIBRARY_PATH

export QWS_MOUSE_PROTO="TPanel:/dev/input/event0 USB:/dev/input/mice"

export QWS_KEYBOARD=TTY:/dev/tty1

Para correr qtopia ejecutar $QPEDIR/bin/qpe 1>/dev/null 2>/dev/null

Al ejecutar qpe obtendremos una interfaz como la mostrada en las figuras de abajo, sin embargo tenemos que considerar que tales figuras muestran un qtopia personalizado, es decir que se ha removido pestañas y eliminadas aplicaciones que no son necesarias para nuestros propósitos.

Nuestro qtopia personalizado consta de 5 pestañas: Aplicaciones, Configuraciones, Instrumentos, Utilidades, Documentos.

Los programas de instrumentos irán en la pestaña de instrumentos como es mostrado en la figura **5 .3** , mientras que las OTMs en la pestaña de aplicaciones como se muestra en la figura **5 .1** .

Para poder personalizar qtopia debemos de considerar lo siguiente:

En el directorio /opt/Qtopia/bin se encontrarán todas las aplicaciones instaladas.

Todas las aplicaciones deberán tener un archivo de extensión .desktop el cual es el que informa entre otras cosas sobre el icono y label que le corresponde a cada aplicación.

En el directorio /opt/Qtopia/apps existen las carpetas Aplicaciones , Configuración , Instrumentos , Utilidades dentro de las cuales están los archivos .desktop asociados a cada aplicación encontrada en /opt/Qtopia/bin.

Estas carpetas aparte de los archivos .desktop tendrán un archivo oculto .directory donde podremos editar el nombre de la pestaña.

Por ejemplo el ejecutable calculadora que vemos en la pestaña de Aplicaciones lo localizaremos en /opt/Qtopia/bin. Esta aplicación tendrá un archivo calculadora.desktop . ubicado en la ruta /opt/Qtopia/apps/Aplicaciones.

Dentro de este archivo calculator.desktop editamos el nombre del icono que veremos para la calculadora el cual reside en /opt/Qtopia/pic además de el label asignado abajo del ícono que lo hemos dejado como Calculator.

En nuestro caso hemos aparte de eliminado pestañas y aplicaciones , hemos agregado la aplicación OTM y la pestañas Instrumentos con sus respectivas aplicaciones.

Por ejemplo para insertar la aplicación OTM hemos colocado el executable otm en la ruta

/opt/qtopia/bin, el otm.desktop en /opt/Qtopia/apps/Aplicaciones, el icono en /opt/Qtopia/pic/otm/ , un help de la aplicación será ubicado en /opt/Qtopia/help/html .

Para poder agregar la pestaña Instrumentos lo único que hacemos es crear un directorio con el nombre Instrumentos en /opt/Qtopia/apps y creamos un archivo .directory donde editaremos el título de la pestaña.

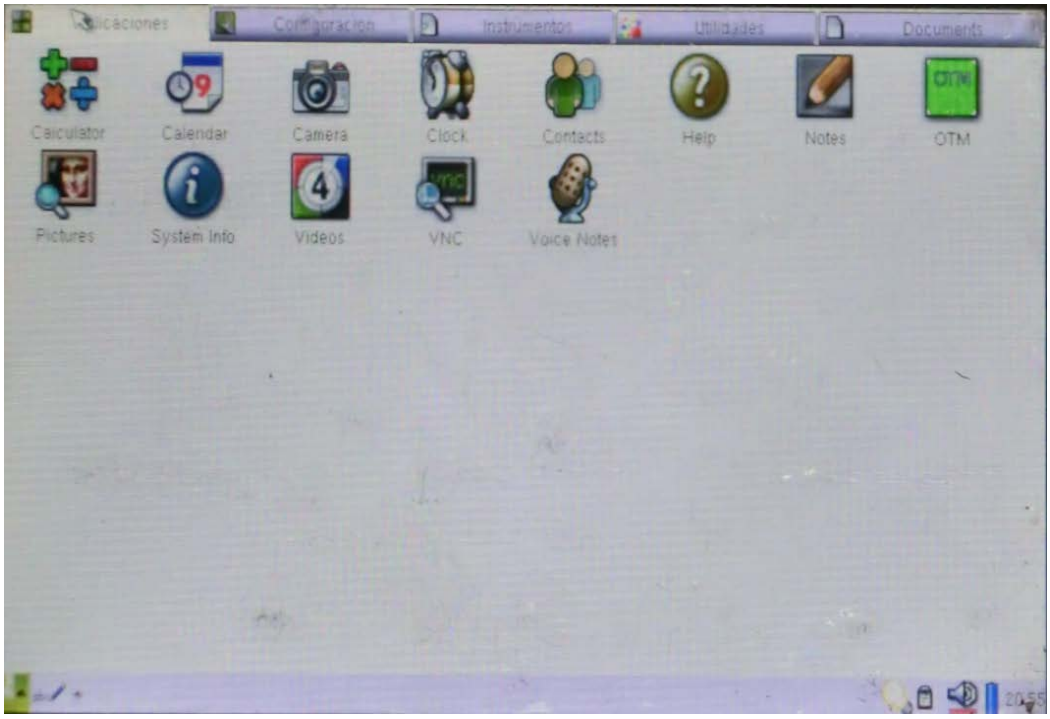


Figura 5.1 Pestaña de aplicaciones de qtopia.(Fuente Propia)

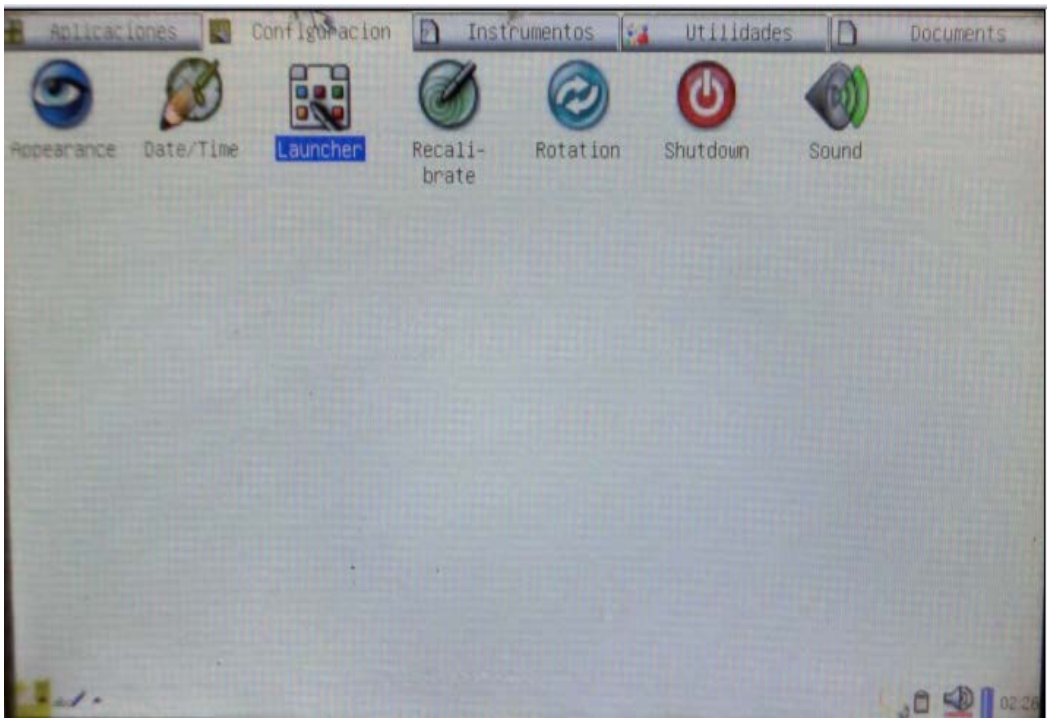


Figura 5.2 Pestaña de configuración de qtopia. (Fuente Propia)

En la figura **5 .1** observamos la pestaña de Aplicaciones donde colocaremos nuestra OTM y en la figura **5 .2** observamos la pestaña de configuraciónes para personalizar nuestro sistema Qtopia.

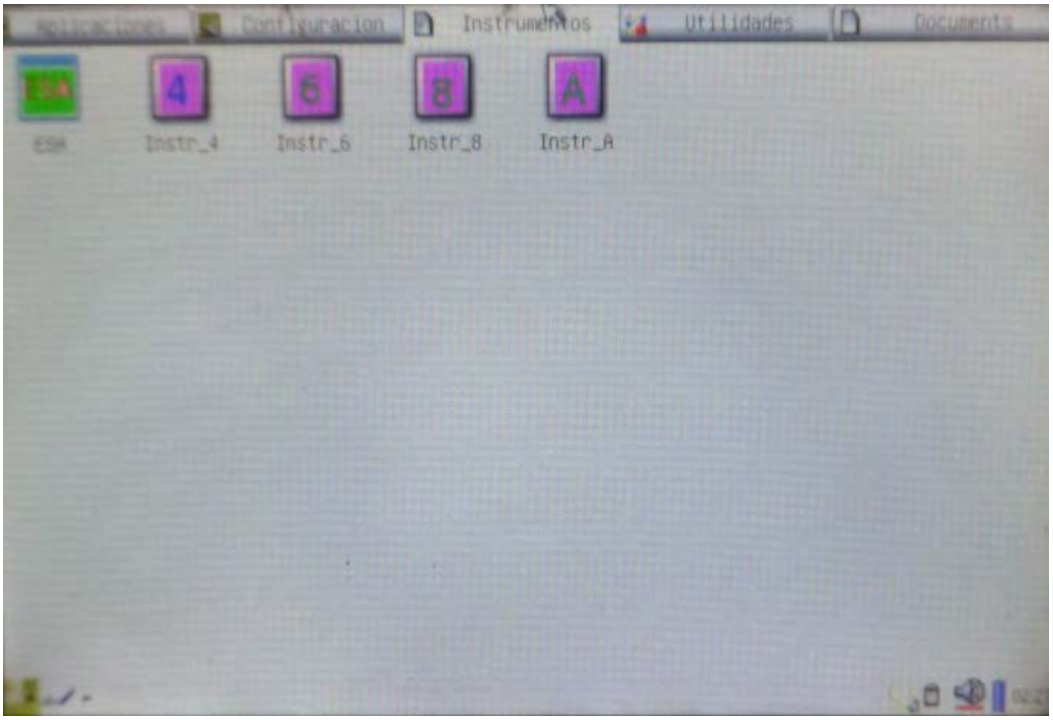


Figura 5.3 Pestaña de Instrumentos de qtopia. (Fuente Propia)

En la figura **5 .3** observamos la pestaña de Instrumentos la cual contendrá nuestras aplicaciones de todos los instrumentos que se han de diseñar.

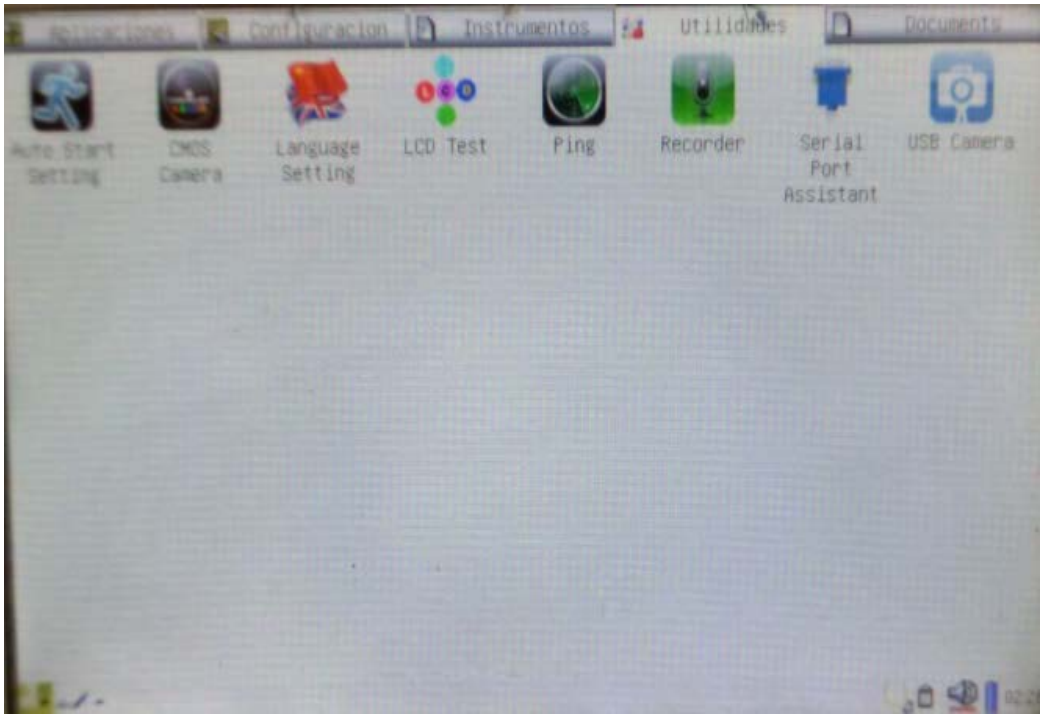


Figura 5.4 Pestaña de Utilidades de qtopia. (Fuente Propia)

En la figura **5 .4** observamos la pestaña de Utilidades en la cual destacamos la camará cmos que utilizaremos para tomar fotos de repuestos entre otras cosas.

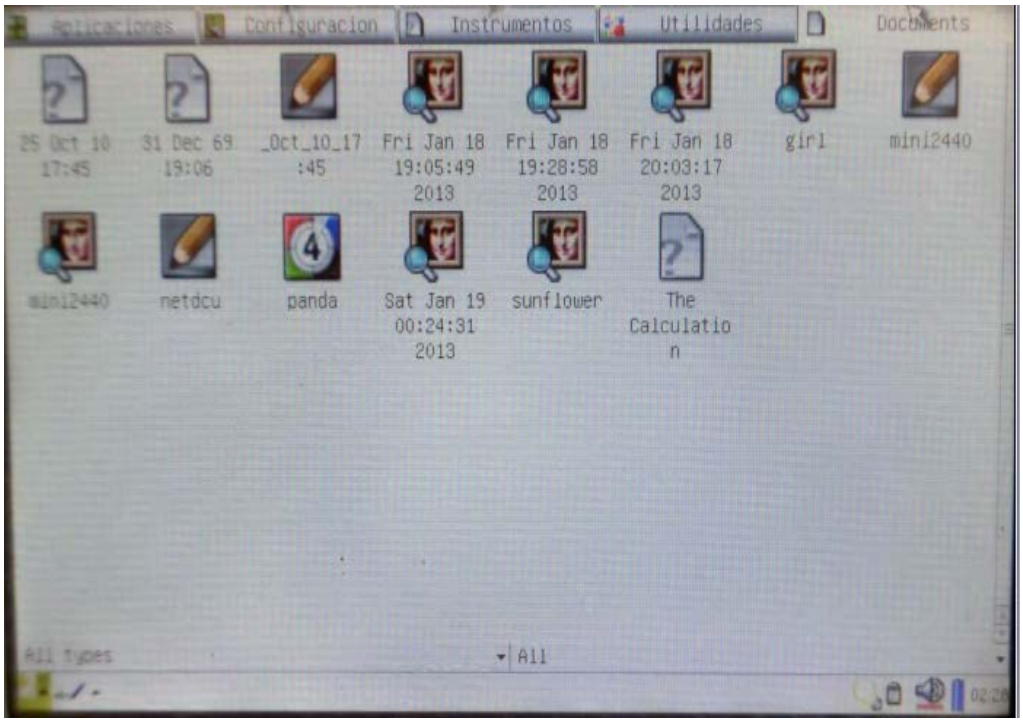


Figura 5.5 Pestaña de Documentos de qtopia. (Fuente Propia)

5.3 Flujómetro de masa .

Para poder probar el sistema de adquisición de datos y la GUI mostrado en la figura **1.2** usaremos el instrumento TSI (Flujómetro de masa térmico). El dspic slave será encargado de recibir la información de flujo, temperatura, presión vía rs232.

Configuración RS232 : Baude Rate 38.4 k

Data Bits 8

Parity None

Stop Bits 1

Flow Control None

El TSI tiene una serie de comandos de configuración de los cuales usaremos:

DmFTPnnnn

Retorna los datos de flujo, temperatura y presión en intervalos igual a la tasa de muestreo. Los datos son retornados en el orden flujo, temperatura y presión.

D: Denota transferencia

M: Denota formato de la data: A = ASCII, B = binario, C = ASCII seguido por CR and LF

F: Solicitud de lectura de flujo (remplazado con ‘x’ si el flujo no es deseado)

T: Solicitud de temperatura (remplazado con ‘x’ si la temperatura no es deseada).

P: Solicitud de presión (remplazado con ‘x’ si la presión no es deseada).

nnnn: Denota máximo número de muestras a retornar, su rango va de 1 a 1000. (‘0500’ denota 500 lecturas, es obligatorio completar con ceros)

Ejemplo a) DAFxP0250

Solicitud de 250 lecturas de flujo y presión en formato ASCII.

Ejemplo b) DBxTx1000

Solicitud de 1000 lecturas de temperatura en formato binario.

El flujo es retornado en unidades de Std L/min or L/min.

La temperatura es retornada en unidades de °C.

La presión es retornada en unidades de kPa.

Ejemplo c) DAFxx0005

Solicita 5 muestras de flujo en formato ASCCI

Flujómetro retorna OK <CR> <LF>

Flujómetro retorna la data como sigue.

1.10 ,1.20 ,1.25 ,1.23, 1.20 <CR> <LF>

Ejemplo d) DBFxx0005

Solicita 5 muestras de flujo en formato binario.

Un ejemplo de los datos retornados podría ser:

0x00 0x33 0x09 0x33 0x1f 0x33 0x25 0x33 0x2d 0x33 0x2e 0xff 0xff

Después de la conversión la data sería:

130.65 130.87 130.93 131.01 131.02

Ejemplo e) DCFTx0005

Solicitud de 5 muestras de flujo y temperatura en formato ASCCI pero con CR y LF seguido de cada grupo de datos.

La data retornada sería:.

1.10 , 23.45 <CR> <LF>

1.20 , 23.53 <CR> <LF>

1.25 , 23.48 <CR> <LF>

1.23 , 23.39 <CR> <LF>

1.20 , 23.50 <CR> <LF>

SSRnnnn.

Configura la tasa de muestreo a nnnn milisegundos.

El rango de nnnn es de 1 a 1000, nnnn=’0005’ denota 5 milisegundos, es obligatorio

completar con ceros.

Después que el comando es procesado, la secuencia de reconocimiento “OK” CR LF es enviada. Si el comando no es reconocido el código de error ERRn es enviado.

Ejemplo f) SSR0010

Configura la tasa de muestreo a 10ms.

Flujómetro de masa retorna OK <CR> <LF>

Figura 5.6 Flujómetro de masa TSI (Fuente (10))

5.4 Programa de control de instrumento .

De la figura **5.7** podemos observar que la mayoría de los programas de instrumentos tendrán zonas para visualizar curvas en el tiempo además de otras para valores numéricos los cuales serán enviados a las OTMs mediante el botón send.

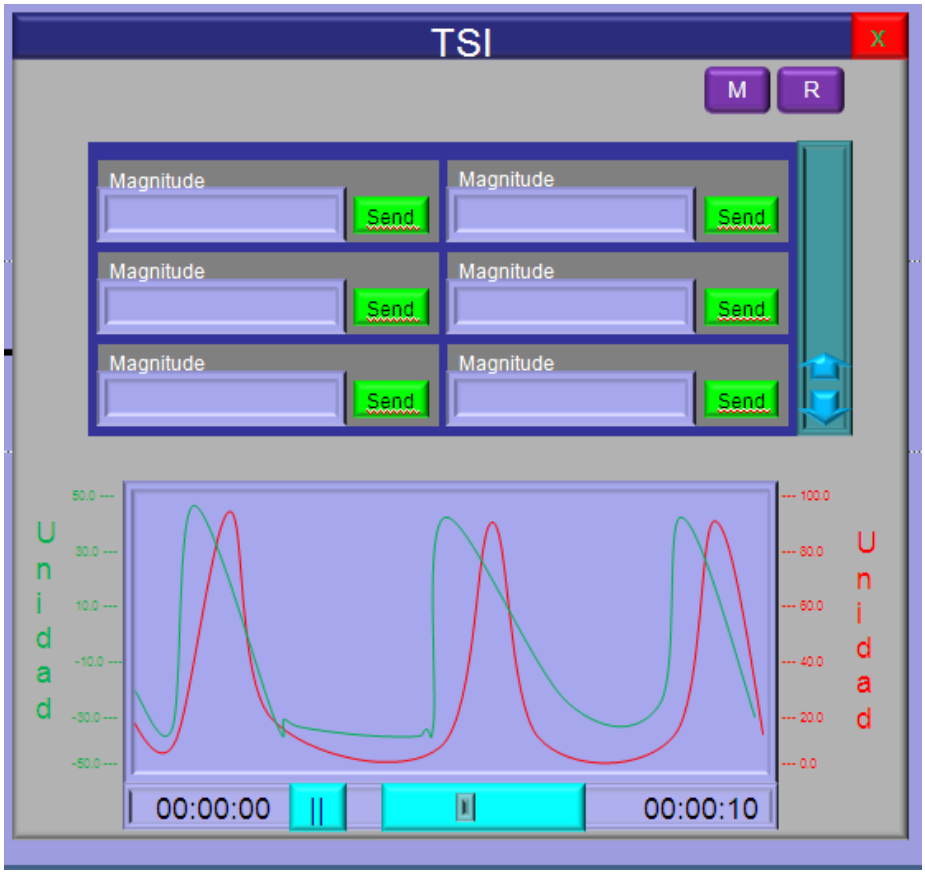


Figura 5.7 Interfaz genérica para un programa gráfico de instrumento (Fuente Propia)

De la figura **5 .8** podemos observar que el programa de comunicaciones solicitará una zona de memoria compartida la cual es creada por el Programa de Comunicaciones que maneja la interfaz paralela. El Programa de Comunicaciones y los Programas Gráficos de Instrumentos se comunicarán viendo el contenido de la memoria compartida.

Al ejecutarse el Programa de Control de Instrumentos se creará un hilo de ejecución que será encargado de configurar el canal y parámetros del instrumento como frecuencia de muestreo y número de puntos por frame.

Después se establecerá una manejador asociándolo con las señales **(8)** de usuario SIGUSR1, SIGUSR2 de tal forma que cuando haya datos disponibles en la memoria compartida el Programa de Comunicaciones envié dicha señal al Programa Gráfico de Instrumento para que este pueda graficar curvas o mostrar los valores numéricos.

Por último el programa ingresará al bucle de eventos esperando por eventos de touchscreen, mouse , keyboard.

La interfaz del Programa Gráfico de Instrumento tendrá la posibilidad de moverse y redimensionarse utilizando los botones M y R ubicados en la parte superior izquierda.

Además como observamos en la figura **5 .7** quedará dividida en dos zonas, una para mostrar valores numéricos y la otra para gráficos de curvas.

Ambas zonas son activas lo que implica que al ser presionadas saldrá una nueva ventana para su respectiva configuración.

En el caso de la zona de valores numéricos la configuración constará de la cantidad de valores que deseamos mostrar y el orden en que deseemos que aparezca como también tendrá la posibilidad de cambiar la unidad de la magnitud.

En el caso de la zona de curvas también tendrá ventas de configuración las cuales aparecerán al ser presionadas las zonas activas permitiendo elegir el tipo de curva y su unidad.

En el caso de los límites estos podrán ser cambiados usando una ventana la cual saldrá presionando el label de unidad o directamente mediante el teclado físico o virtual de qtopia.

En la parte de abajo del plotter podemos ver la zona de tiempo la cual posee un botón el cual nos permitirá detener la curva además de un slider el cual podrá recorrer la curva para ver valores pasados para su análisis. El plotter podrá referenciar el tiempo de dos formas, uno relativo y uno en donde podremos ver la hora, minutos y segundos transcurridos desde que se inicio el proceso. La zona de tiempo es también una zona activa de tal forma que si

la presionamos no saldrá una ventana de configuración para poder cambiar la base de tiempo entre otras configuraciones.

Todas las ventanas posibles de configuración en dicho programa serán Bloqueantes , es decir que mientras no se cierren no se podrá acceder a otras.

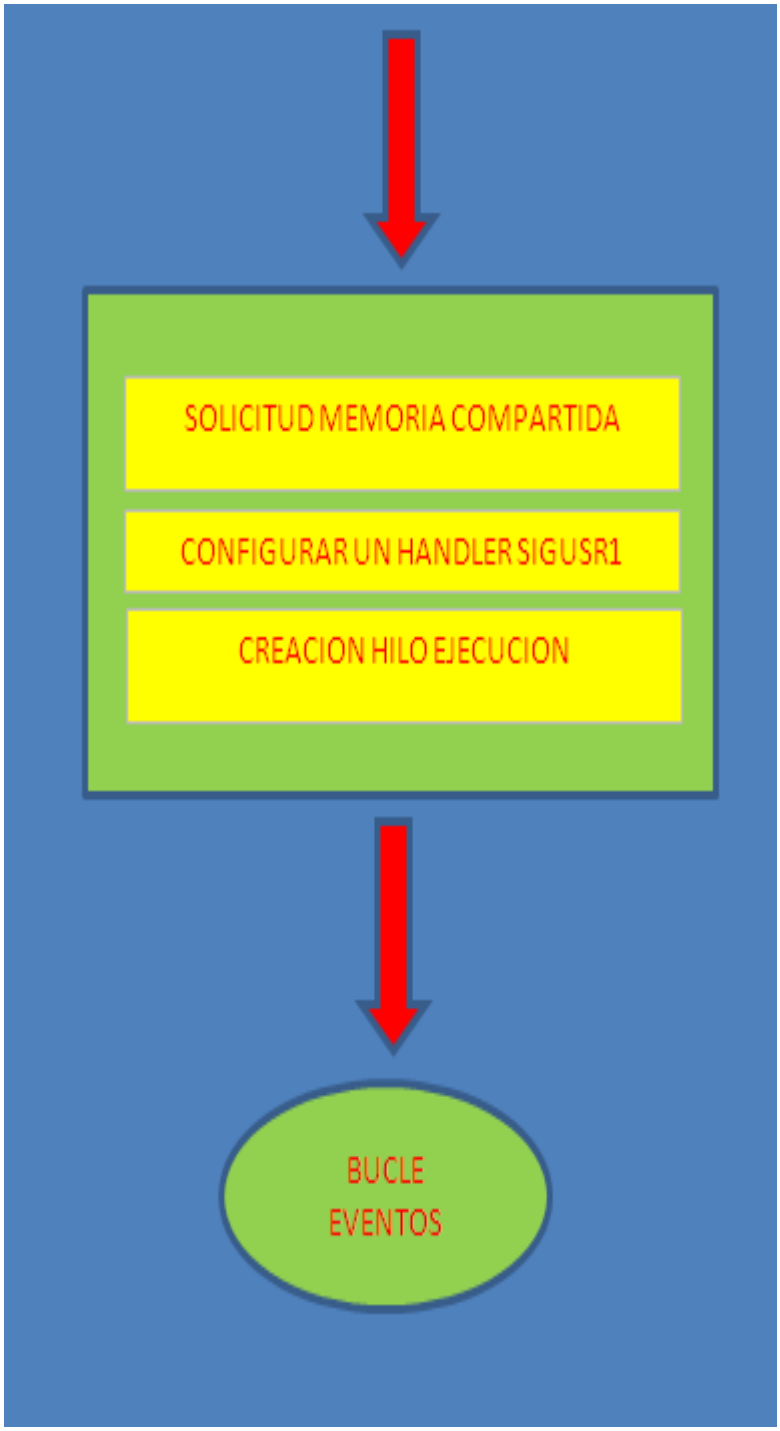


Figura 5.8 Diagrama de bloque programa de instrumento (Fuente Propia)

La sección de memoria compartida constará de la siguiente data:

Tipo de comando : Comando especificado en el protocolo de comunicaciones

Data para el canal 1 : Arreglo de 512 unsigned short int

Data para el canal 2 : Arreglo de 512 unsigned short int

Data para el canal 3 : Arreglo de 512 unsigned short int

Data para el canal 4 : Arreglo de 512 unsigned short int

Estado canal 1 : Estado del canal 1.

Estado canal 2 : Estado del canal 2.

Estado canal 3 : Estado del canal 3.

Estado canal 4 : Estado del canal 4

Comandos Instr 1 : Arreglo de comandos para canal 1.

Comandos Instr 2 : Arreglo de comandos para canal 2.

Comandos Instr 3 : Arreglo de comandos para canal 3.

Comandos Instr 4: Arreglo de comandos para canal 4.

Estado Instr 1 : Arreglo de Estado para canal 1.

Estado Instr 2 : Arreglo de Estado para canal 2.

Estado Instr 3 : Arreglo de Estado para canal 3.

Estado Instr 4 : Arreglo de Estado para canal 4.

Data No Programada Instr 1 : Data no programada canal 1.

Data No Programada Instr 2 : Data no programada canal 2.

Data No Programada Instr 3 : Data no programada canal 3.

Data No Programada Instr 4 : Data no programada canal 4.

Flags f_CmdtoInstr : Flags que indicará al programa de comunicaciones que existe datos disponibles para un determinado canal.

Pid 1 : Identificador de proceso del programa para el canal 1.

Pid 2 : Identificador de proceso del programa para el canal 2.

Pid 3 : Identificador de proceso del programa para el canal 3.

Pid 4 : Identificador de proceso del programa para el canal 4.

En la figura **5 .9** observamos el diagrama de flujo que todo programa gráfico de instrumento deberá de ejecutar.

Primero verifica la existencia de un canal disponible, después detectará si el instrumento se ha encontrado para después configurar los parámetros de funcionamiento.

Todo esto se realizará en un hilo de ejecución alterno al programa gráfico de instrumento el cual posteriormente será destruido previamente de haber recibido un Estado de que la programación del instrumento se realizó satisfactoriamente.

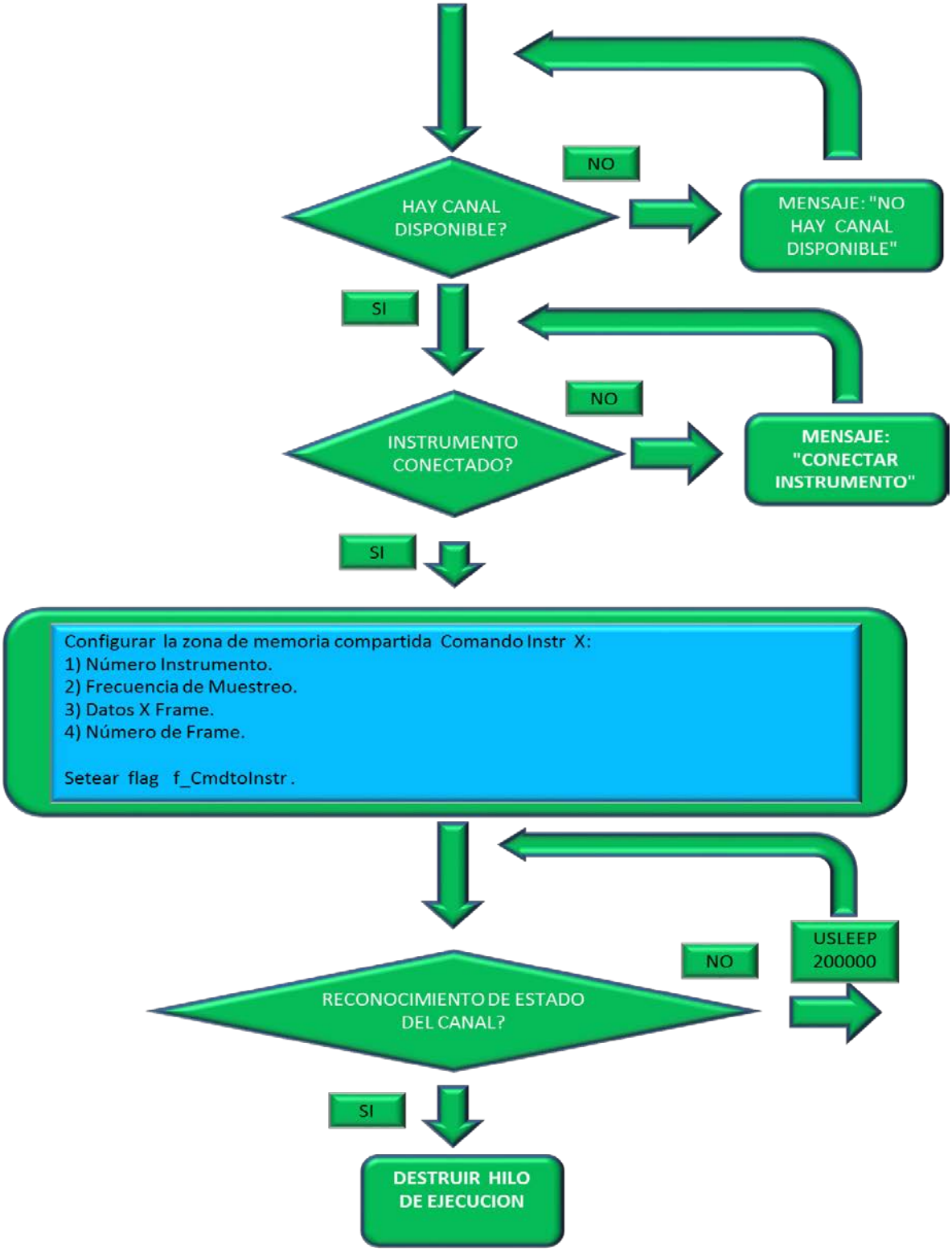


Figura 5.9 Hilo de Ejecución del Programa Gráfico de Instrumento. (Fuente Propia)

5.5 Programa Generador de la Orden de Trabajo de Mantenimiento.

El programa de la OTM será el encargado de construir dicha OTM en el sistema embebido. Las OTMs serán diseñadas con el QTCreator (Herramienta de diseño grafico para aplicaciones QT) · Cuando se realiza un diseño gráfico usando el QTCreator se

generará un archivo xml (eXtensible Markup Language) el cual contiene toda la información de los objetos usados en la implementación de dicho diseño, por ejemplo en la figura **5 .14** observamos una página de una OTM elaborada con el QTCreator , su archivo xml asociado lo podemos observar en el **Anexo** F.

Una OTM constará de varios archivos xml nombrados como: Page1.xml, Page2.xml , Page3.xml ... PageN.xml. Una vez que se tenga los archivos xml estos ingresarán a un programa en Windows/Linux el cual extraerá dicha información para colocarla en la base de datos SQLITE. Más adelante detallaremos el programa encargado para hacer esto.

Una vez que tengamos la base de datos con la información de la OTM será trasladada al sistema embebido mediante USB lo cual detallaremos al final de este capítulo.

Teniendo ya la base de datos en el embebido nuestro programa OTM extraerá la información de la Base de datos para construir dinámicamente las páginas correspondientes en el momento que se disparé el proceso correspondiente dentro del PDA.

Antes de analizar el programa OTM primero veremos el Diagrama de Proceso de Mantenimiento de la figura **5.10** . Podemos observar que este está constituido por varias etapas. Los programas involucrados son:

Programa generador de la base de datos sqlite: Como su nombre lo indica este programa recibe las plantillas elaboradas con el qtcreator para generar la Base de datos sqlite que contenga la información de la OTM.

Programa Administrador: Dicho programa será el encargado de proporcionar la lista de equipos que han de ser mantenidos en un determinado periodo.

Programa que llena parte administrativa: Las plantillas generadas no contendrán ningún tipo particular de información, la parte administrativa del equipo a mantener como la Marca, Modelo, Ubicación, Número Patrimonial, Serie, etc serán llenadas por este programa el cual interactuará con la Base de datos del Hospital que tiene el inventario de todos los equipos.

Programa Actualiza Base de datos: Este programa será el encargado de extraer la información de la base de datos sqlite que el sistema embebido entrega y colocarla en la base de datos del hospital.

Programa de generación de reportes: Dicho programa generará los reportes de los equipos.

Como lo mencionamos al principio de la tesis el objetivo de esta se centra en el desarrollo

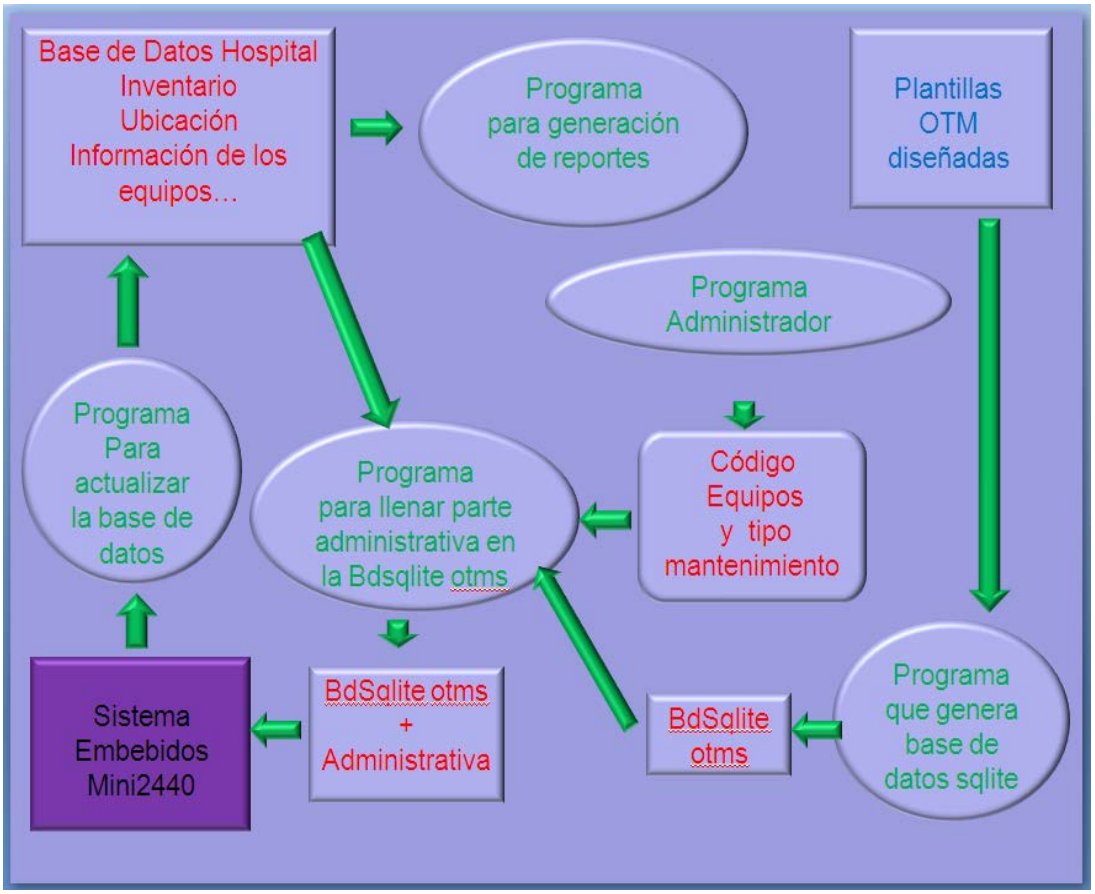


Figura 5.10 Diagrama Global del Proceso de Mantenimiento. (Fuente Propia)

de software y hardware para el sistema embebido sin involucrar los programas antes mencionados que en conjunto conforma el proyecto que se está elaborando, sin embargo antes de ir a analizar el programa de la OTM del sistema embebido detallaremos el programa generador de la base de datos sqlite que será clave para la construcción dinámica de la OTM en el embebido.

El programa Generador de Base de Datos al que llamaremos de ahora en adelante **PGBD** usará la Base Sqlite3.5 cuya cross-compilación la podemos ver en el **Anexo D**. Usamos Sqlite3.5, esta rompe el modelo Cliente, Servidor de otras base de datos haciendo que se inserte a nuestro programa logrando una mejor eficiencia en cuanto a velocidades de consulta, además de ser ligera entre otras características que la llevan a ser una de las mejores opciones para ser trabajadas en un sistema embebido.

La figura **5.11** muestra el diagrama de Bloque del PGBD, al inicio se crea la base de datos con las tablas para cada uno de los widgets usados como son: Labels, ComboBox, EditLine CheckBox, etc. Cada Tabla asociada a un Widget contendrá la ubicación de este correspondiente al número de página y posición a la que pertenece, además de sus características en particular. La lista de tablas usadas por el **PGBD** son:

TObject : Tabla de todos los objetos que constituyen la OTM

CREATE TABLE TObject (IdWindow INTEGER , IdObject INTEGER, Type TEXT, HasInfo TEXT, Name TEXT, Parent TEXT, LayoutParent TEXT, Stretch TEXT , fromRow TEXT , toRow TEXT ,fromCol TEXT, toCol TEXT , HPolicy TEXT, VPolicy TEXT , MaxSize TEXT, MinSize TEXT , PRIMARY KEY (IdWindow,IdObject));

Campos:

IdWindow , IdObject :Información de la página y ubicación del objeto.

Type : Tipo de Objeto

HasInfo : Indica si el objeto tiene información.

Name : Nombre del objeto

Parent : Padre del objeto

LayoutParent : Layout al que pertenece el objeto

Stretch : No usado

fromRow , toRow ,fromCol , toCol :Expansión en caso layout de grilla

HPolicy , VPolicy , MaxSize , MinSize : Políticas de dimensionamiento

TLabel: Tabla de un Label

CREATE TABLE TLabel (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT, Background TEXT, FontFamily TEXT, FontSize TEXT, PixmapImage TEXT, Text TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

Foreground , Background : Colores

FontFamily , FontSize : Tipo de fuente y tamaño

PixmapImage : Pixmap

Text : Texto del Label

TGroupBox : Tabla de Contendor de objetos genéricos

CREATE TABLE TGroupBox (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT, Title TEXT ,PRIMARY KEY (IdWindow,IdObject));

Campos:

Title : Título del contenedor de objetos.

TLineEdit: Tabla de editor de línea

CREATE TABLE TLineEdit (IdWindow INTEGER , IdObject INTEGER, Text TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

Text: Texto del editor de línea.

TComboBox : Tabla de combo box .

CREATE TABLE TComboBox (IdWindow INTEGER , IdObject INTEGER, Item1 TEXT, Item2 TEXT, Item3 TEXT, Item4 TEXT, Item5 TEXT, Item6 TEXT, Item7 TEXT, Item8 TEXT, Item9 TEXT, Item10 TEXT ,PRIMARY KEY (IdWindow,IdObject));

Campos:

item: Items del combo box, Fijado hasta máximo 10.

TSpacer :Tabla que representa los espaciadores

CREATE TABLE TSpacer (IdWindow INTEGER , IdObject INTEGER, Orientation TEXT, Width TEXT , Heigth TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

Orientation: Vertical /Horizontal

Width ,Heigth : Ancho, Alto

TChecBox : Tabla para checkbox

CREATE TABLE TCheckBox (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT, Background TEXT, FontFamily TEXT, FontSize TEXT, Enable TEXT, Text TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

Enable: Está checkeado o no

Text : Texto asociado al checkbox

TRadioButton : Tabla radio button

CREATE TABLE TRadioButton (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT, Background TEXT, FontFamily TEXT, FontSize TEXT, Checked TEXT, Text TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

Checked: Está checkeado o no

Text : Texto asociado al radio button

TMultiLineEdit: Tabla de editor de texto plano

CREATE TABLE TMultiLineEdit (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT, Background TEXT, FontFamily TEXT, FontSize TEXT, ReadOnly TEXT, Text TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

ReadOnly: Editor sólo lectura

TTable: Representación de una Tabla (filas,columnas)

CREATE TABLE TTableImpl (IdWindow INTEGER , IdObject INTEGER, Num_Rows TEXT, Num_Cols TEXT, Proced TEXT, Proportion TEXT, UnidMinCharacters TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

Proced: Procedimiento aritmético a calcular

Proportion: Proporción de las columnas.

UnidMinCharacters: Longitud mínima de una columna

THeadersTableImpl: Tabla nombre los headers de TTable

CREATE TABLE THeadersTableImpl (IdWindow INTEGER , IdObject INTEGER, Position TEXT, Label TEXT ,PRIMARY KEY (IdWindow,IdObject,Position));

Campos:

Position: Posición de la columna

Text: Título de la columna

TCellTableImpl : Tabla que nos da el contenido de una celda específica de un TTable

CREATE TABLE TCellTableImpl (IdWindow INTEGER , IdObject INTEGER, CellRow TEXT, CellColumn TEXT ,value TEXT , PRIMARY KEY (IdWindow,IdObject,CellRow,CellColumn));

Campos:

CellRow: Número de fila de la celda

CellColumn: Número de columna de la celda

TEditValueInstr : Tabla Recolección de datos sensados por los instrumentos

CREATE TABLE TEditValueInstr (IdWindow INTEGER , IdObject INTEGER, Text TEXT, InstrUnit TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

InstrUnit: Unidad

TCPushButton : Tabla de Botones que disparan procesos o aplicaciones

CREATE TABLE TCPushButton (IdWindow INTEGER , IdObject INTEGER, isAProcess TEXT, nameProcess TEXT, isAProcedure TEXT, N_PageProcedure TEXT, PRIMARY KEY (IdWindow,IdObject));

Campos:

isAProcess: Identificador si es un proceso

nameProcess: Nombre del proceso.

isAProcedure : Identificador si es un procedimiento

N_PageProcedure: Número de páginas del procedimiento

En todas las tablas mencionadas tendrán un **UpdateCode** el cual será usado por el programa que actualiza la base de datos del Hospital.

El **PGBD** leerá los archivos .xml extrayendo su información llenando las Tablas mencionadas.

En la figura **5.11** observamos el diagrama del **PGBD**, al empezar el programa se creará la base de datos con sus respectivas tablas para posteriormente instanciar QDomDocument la cual es una clase que nos permitirá leer archivos con formato .xml.

Después configuramos la página de la OTM a leer e ingresamos al corazón del programa que es la función recursiva ListElemt que será la encargada de llenar las tablas con la información que vayamos a extraer de la pagina.xml.

En la figura **5.12** observamos el diagrama de flujo de la función ListElemet, que como mencionamos anteriormente es una función recursiva que tiene como punto de parada la ubicación de un widget como un Label , Lineedit , Checkbox.

Debemos tener en consideración que nuestros widget podrán estar dentro de contenedores como son QGroupBox o ButtonGroup además de que pertenecerán a un determinado Layout. Un layout es un objeto que tomará el control geométrico de nuestros widget.

Nosotros utilizaremos Layout Horizontales , Verticales y de Grilla. La función recorrerá los tags del formato xml ingresando recursivamente a esta cuando encuentre objetos Contenedores o Layout y se detendrá cuando ubique a un determinado Widget en particular decodificándolo.

Ahora que ya tenemos un visión global de cómo se genera las base de datos que trasladaremos al sistema embebido analizaremos el Programa OTM.

Al disparar el Icono de OTM que se muestra en la pestaña de aplicaciones de la figura **5 .1** aparecerá una ventana pidiéndonos que seleccionemos si deseamos ingresar a las OTM actuales o pendientes. Las OTMs actuales son aquellas que son entregados a los técnicos al mes, pudiendo ser aproximadamente diez. Si recibe más órdenes de trabajo (más de cinco) entonces las que tenía previamente pasan a ser Pendientes.

Al seleccionar una OTM se mostrará un ventana indicando el porcentaje del proceso de construcción de la OTM seleccionada mostrado en la figura **5 .13** , una vez que este lista aparecerá la primera página de esta.

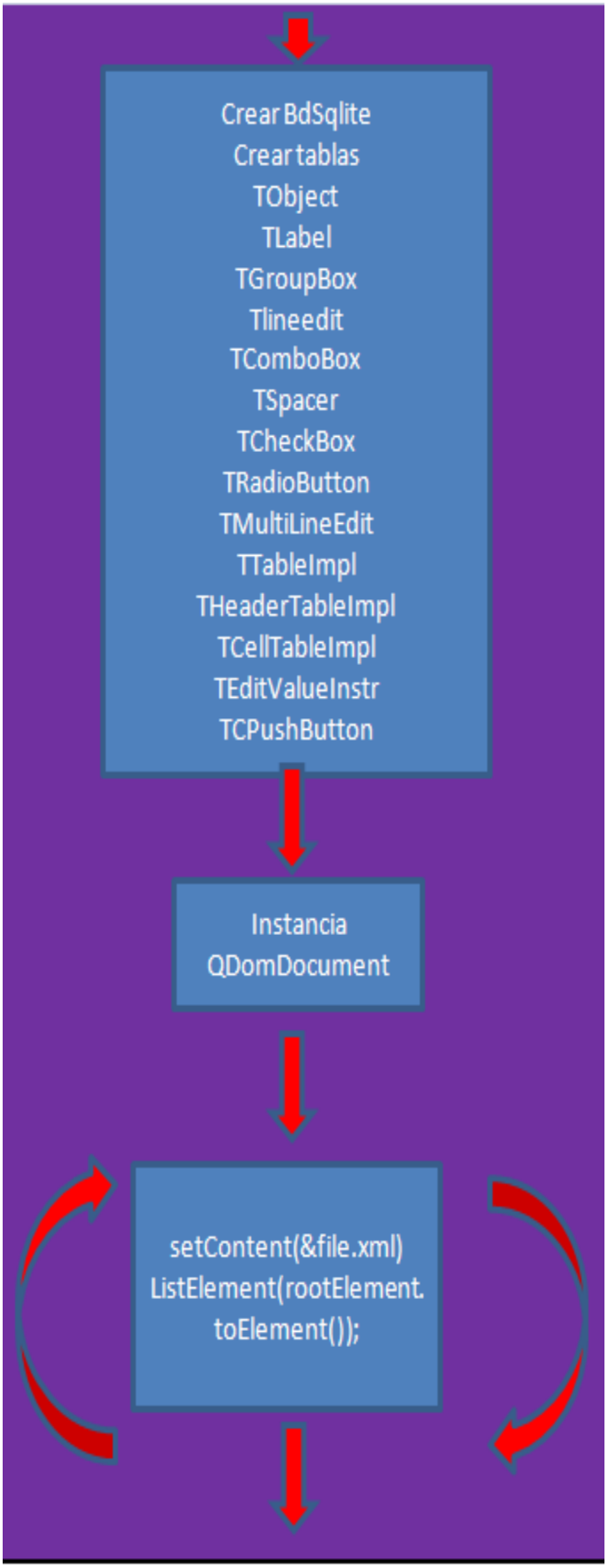


Figura 5.11 Diagrama de bloque programa generador BdSqlite. (Fuente Propia)

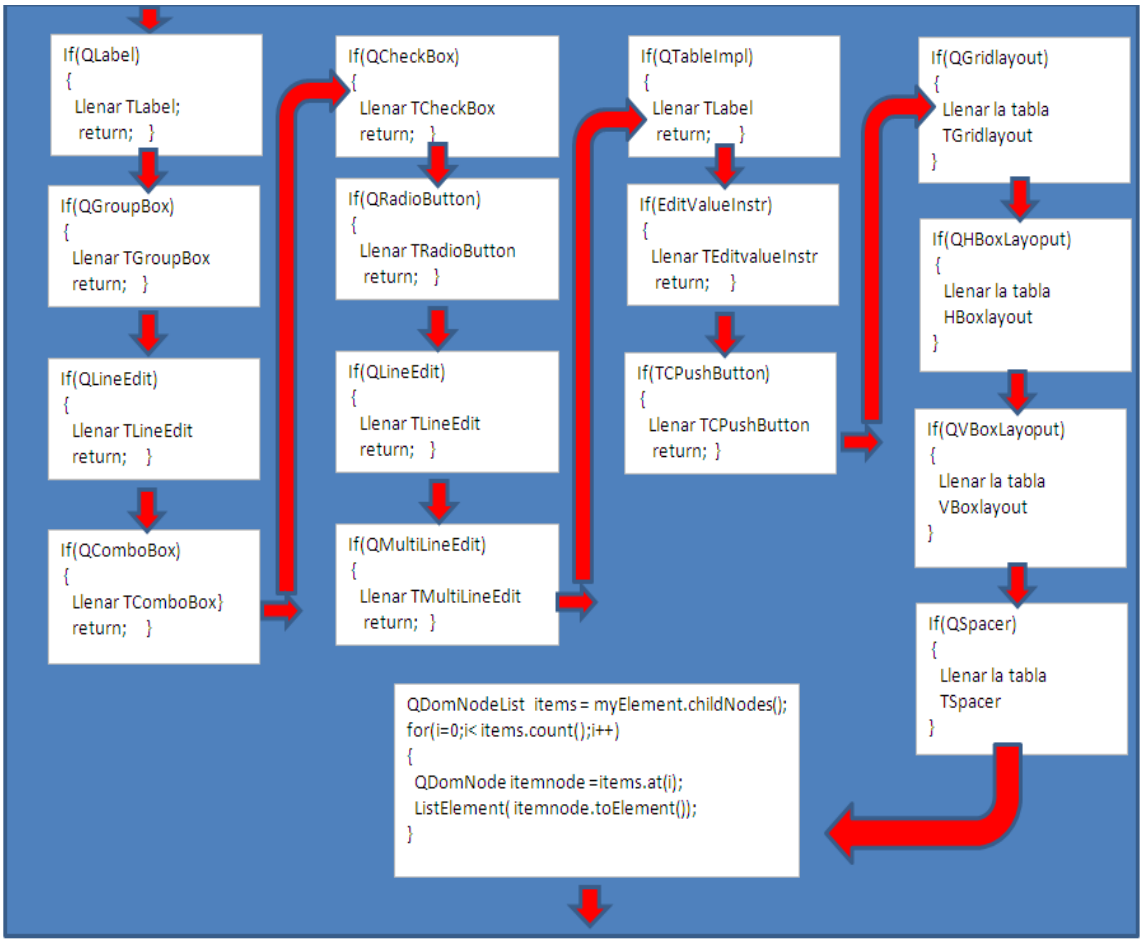


Figura 5.12 Diagrama del método ListElement del generador BdSqlite. (Fuente Propia)

En la figura **5 .14** observamos la primera página de una OTM, como se puede ver esta contiene una parte administrativa indicando la marca, serie, modelo, etc. En su parte inferior encontremos botones para desplazarnos a lo largo de la OTM desde la última a la primera página o en forma correlativa. Podremos salvar la página actual con el botón Save o en el momento de cerrar el programa actualizando la Base de datos.

El programa utilizará la clase stackwidget la cual me permite almacenar un conjunto de widget y sólo el que se encuentre en el top será el actualmente visible, de tal forma que con los botones que se muestran en la parte inferior izquierda antes mencionados podremos avanzar y retroceder para poder llenar todos los campos disponibles como lo mencionamos anteriormente.

Un widget particular llamado editValueInstr será el encargado de almacenar datos desde el el Programa de Control Instrumentos. Para esto cada vez que se ejecute la aplicación OTM esta creará un canal de comunicaciones basado en el protocolo **QCop (inter-process Message)** el cual permite que las aplicaciones Qt puedan intercomunicarse.

Como se muestra en la figura **5 .15** el programa de instrumento enviará un valor a través

de este canal al Widget editValueInstr que tenga el enfoque dentro de nuestra OTM.

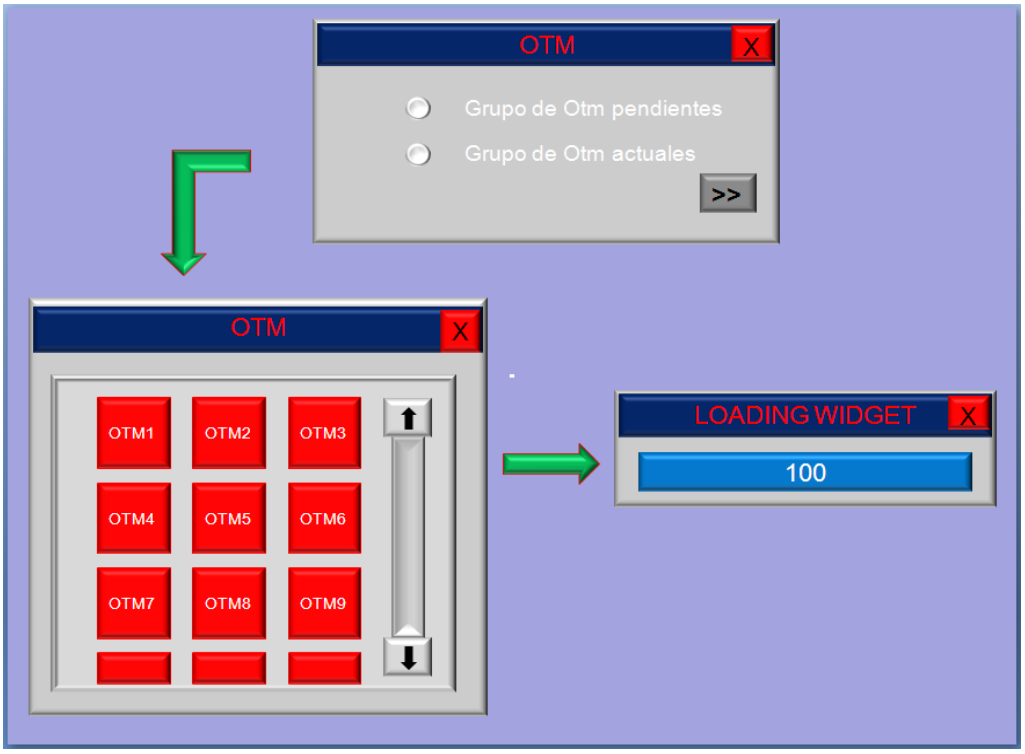


Figura 5.13 Pasos para cargar una OTM (Fuente Propia)

Figura 5.14 Página uno de la OTM (Fuente Propia)

Nuestro Widget editValueInstr tendrá la capacidad de conectarse al canal creado por la OTM cuando este tenga el enfoque, y cuando lo pierda se desconectará. El editValueInstr tendrá un led que indicará el estado de conexión, verde conectado , rojo desconectado. Una característica importante del editValueInstr es que manejará el Buzzer del sistema embebido de tal forma que cuando se intente enviar un dato que no corresponda con las unidades se generará dos pitido indicando un error en las unidades, por ejemplo cuando intentemos enviar 10ma a un editValueInstr que almacena Voltaje. Si ningún editValueInstr tiene el enfoque el valor enviado se perderá. Una vez que el editValueInstr toma un valor automáticamente dicho objeto pasa el enfoque al siguiente widget para agilizar y facilitar el llenado de la OTM a los técnicos. El editvalueInstr podrá ser llenado manualmente por el técnico con el teclado por lo que esta clase tiene la capacidad de autocorrección. Dicha clase se conecta a una tabla de una base de datos que contiene las unidades y pseudounidades permitidas. Por pseudounidades nos referimos por ejemplo en el caso de la unidad de Corriente a MA, Ma , miliApere, Ampere, ma, ampere etc de tal forma que si el técnico llene con una de estas automáticamente se corrija al sistema apropiado como A,mA. En caso de error el valor ingresado será borrado y se emitirá un pitido a través del buzzer.

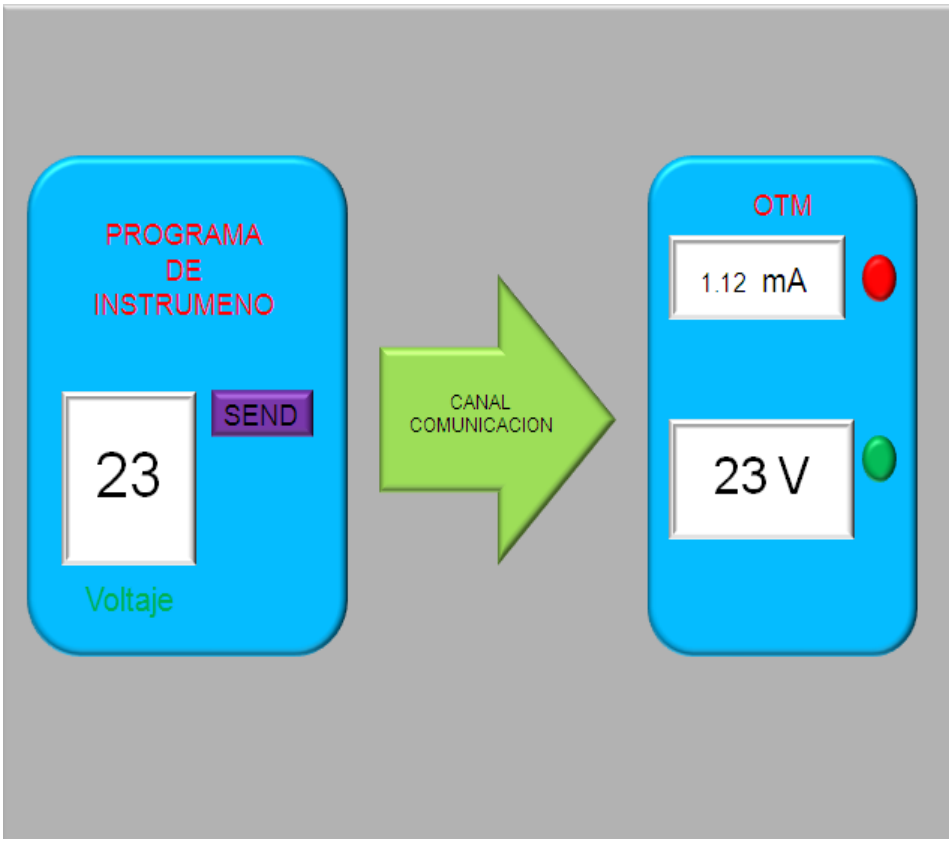


Figura 5.15 Transferencia datos desde el programa instrumento y la OTM (Fuente Propia)

En la figura **5 .16** se muestra el proceso de construcción dinámico de un Label dividiéndose en tres bloques. AllocateLabel el cual separa memoria dinámica, SettingLabel para la configuración propia del Label como es el widget padre de este , infomacion del texto , tipo de fuente y tamaño , color foreground , background , etc por último AddLabellayout para agregar el Label a Layout al que pertenece.

Las figuras **5 .17** , **5 .18** , **5 .19** muestran la implementación de los bloques mencionados arriba. Como se puede observar se han empleado las librería STL (**standard template library**) para el manejo de contenedores. La construcción de los demás Widget son similares.

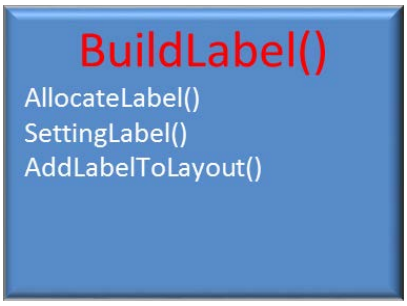


Figura 5.16 Construcción dinámica Label (Fuente Propia)

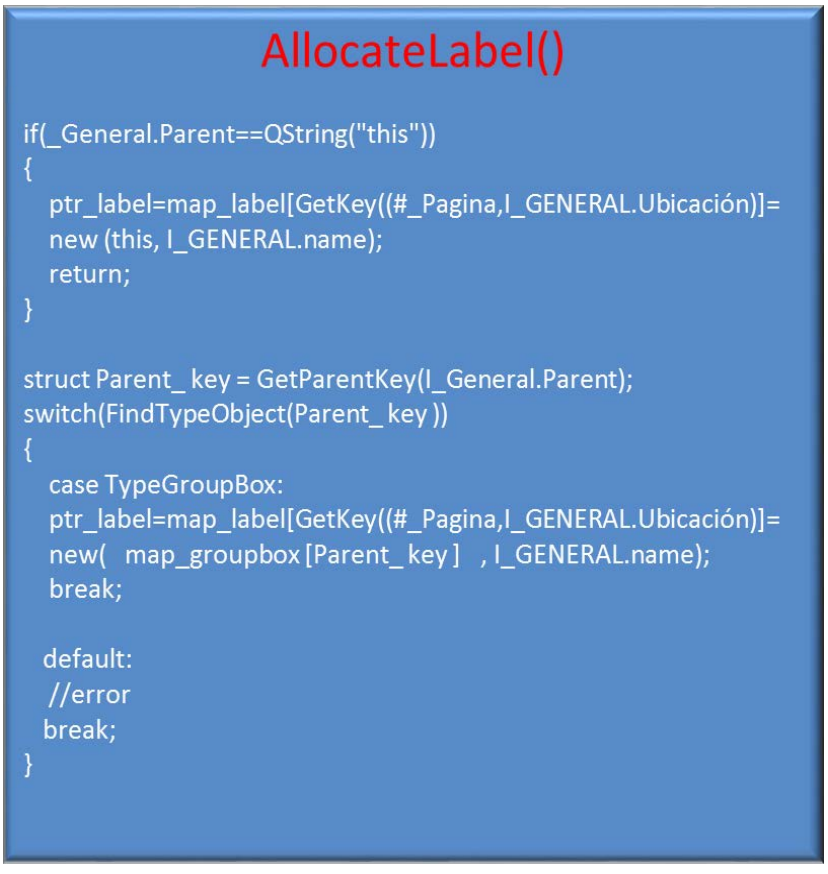


Figura 5.17 Asignación Dinámica Label (Fuente Propia)

SettingLabel()

```
bool enable=false;
QString ruta("/op/.../");

        if(I_LABEL.pixmap!=QString("-"))
        {
             ptr_label->setSizePolicy( (QSizePolicy::Sizetype)0, (QSizePolicy::Sizetype)0 ,
                                       ptr_label->sizepolicy() .hasHeightForWidth() );

             QPixmap image(ruta.append(I_LABEL.pixmap);
              ptr_label->setPixmap(image);
              ptr_label->setScaledContents(true);

          return;
        }

        if(  I_GENERAL.parent ==QString("this"))
        {
           if(I_LABEL.foregroud==QString("black")
              enable = false;
            else
              enable=true;
        }
        else
        {
           if(I_LABEL.foregroud==QString("black")
            {
              ptr_label->setPalette(palette());
              enable=false;
            }
            else
              enable=true;
        }

        if(enable)
        {
          pal=ptr_label->palette();
          cg=pal.active();
          cg.setColor(QColorGroup:: Foreground, I_LABEL.foregroud);
          pal.setActive(cg);

          cg=pal.inactive()
          cg.setColor(QColorgroup::Foreground, I_LABEL.foregroud);
          pal.setInactive(cg);

         ptr_label->setPalette(pal);
        }

        if(I_LABEL.background != QString("inherited") )
        {
        pal=ptr_label->palette();
        cg=pal.active();
         cg.setColor(QColorGroup:: Background ,I_LABEL.background);
         pal.setActive(cg);

        cg=pal.inactive()
         cg.setColor(QColorgroup::Background ,I_LABEL.background);
          pal.setInactive(cg);

          ptr_label->setPalette(pal)

        }

         QFont  font_Ptr_label;
          font_Ptr_label.setFamily(I_LABEL.Fontfamily);
           font_Ptr_label.setPointSize(I_LABEL.PointSize.toInt());

        Ptr_label->setFont(font_Ptr_label);
        Ptr_label->setText(tr( I_LABEL.text ));
```

Figura 5.18 Configuración del Label (Fuente Propia)

AddLabelToLayout()

```
struct LayoutParent_ key = GetParentKey(I_General.Layout);
switch(FindTypeObject(LayoutParent_ key ))
{
   case TypeHBoxLayout:
   map_LayoutH[LayoutParent_ key ]-> addWidget(ptr_label, I_General.Stretch.toInt());
   break;

   case TypeVBoxLayout:
   map_LayoutV[LayoutParent_ key ]-> addWidget(ptr_label, I_General.Stretch.toInt());
   break;


   case TypeGridLayout:
   map_GridLayoutH[LayoutParent_ key ]-> addMultiCellWidget( ptr_label, I_General.from_ROW.toInt(),
    I_General.to_ROW.toInt(), ,      I_General.from_COL.toInt(),      I_General.to_COL.toInt(),);
   break;

 }
```

Figura 5.19 Agregar Label a Layout (Fuente Propia)

5.6 Programa para enviar las OTMs (Base de Dato) hacia mini2440 vía USB.

En el Programa de la OTM visto anteriormente se analizó la construcción de formularios OTMS en forma dinámica desde una base de datos sqlite, pero no se habló de la forma como esta base de datos es enviada al sistema embebido.

Para poder enviar la base de datos desde Windows o Linux debemos configurar nuestra mini2440 para que sea vista como un dispositivo de almacenamiento.

De la figura **5 .20** observamos que dicho programa leerá el directorio 1 (Dir 1) donde se alojarán las OTMs a desarrollar mientras que el directorio 2 (Dir 2) nos mostrará las OTM terminadas. Ambos directorios serán los que podremos observar desde Windows o Linux cuando conectemos la mini2440 vía USB.

El botón importar OTM permitirá trasladar las OTMs del directorio 1 (Dir 1) hacia el sistema de archivos de la mini2440 (Directorio Actual), mientras que el botón de exportar pasará las OTMs acabadas desde el sistema de archivos de la mini2440 (Directorio Pendiente) hacia el directorio 2 (Dir 2).

El programa de OTM mencionado anteriormente en el momento de iniciarse lee los directorios Actual y Pendiente mostrando las OTM como se muestra en la figura **5 .13** .

Una vez que presionemos el botón importar las OTM actuales pasarán automáticamente a ser pendientes si la cantidad de OTMs nuevas a enviar son mayor a cinco, por lo general un técnico tendrá mensualmente de 10 a 15 OTMs. Para resumir, dicho programa tendrá que manipular 4 directorios, dos de ellos en el sistema de archivos de la mini2440 (Directorio

Actual y Pendiente) y otros dos que serán accesibles cuando conectemos la mini2440 vía USB (Dir 1 y Dir2).

A continuación veremos los pasos para poder ver a nuestra mini2440 como un dispositivo de almacenamiento USB.

El primer paso es compilar los módulos necesarios para poder habilitar el FSG (Gadget Stored file-backed). El FSG dará soporte para la clase de almacenamiento masivo USB.

El host lo verá como una unidad Lógica (LUN) , la información almacenada en dicha LUN tendrá que ser mantenida en algún lugar del Gadget en un archivo normal , un device block o incluso en la ramdisk.

Compilamos los módulos del kernel de la mini2440 con la opción M (Montado)

(M) --> Support USB_Gadget

(M)--> File-Storage

En /lib/modules/2.6.32.2-FriendlyARM de la mini2440 colocamos los módulos obtenidos.

s3c2410_udc.ko (USB Device Controller Gadget).

g_file_storage.ko (File-Backed Storage Gadget).

El s3c2410_udc.ko es una dependencia para el g_file_storage.ko.

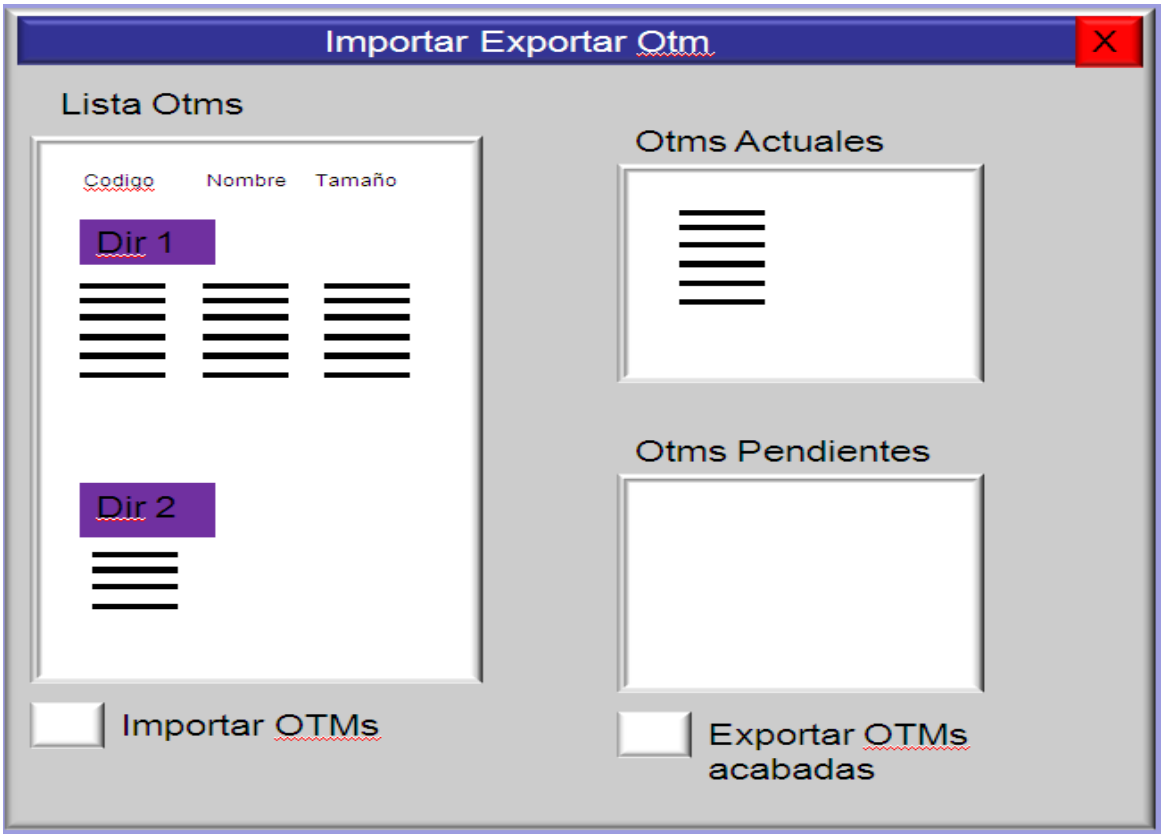


Figura 5.20 Programa importación y exportación de las Otms. (Fuente Propia)

Para poder usar el FSG debemos insertar el modulo g_file_storage.ko teniendo en cuenta las dependencias de este. Como lo mencionamos anteriormente debemos tener un backing-file en el Gadget para poder mantener la información, por lo que en el momento de insertar el módulo g_file_storage.ko debemos de indicarle la ruta de acceso a tal archivo el cual deberá estar creado de antemano.

Para crear un baking storage file desde una máquina host seguiremos los siguientes pasos:

1) Creamos un archivo limpio de 64M llamado backinf_file en la ruta root/

bash# dd bs=1M count=64 if=/dev/zero of=/root/backing_file

64+0 records in

64+0 records out

2) Particionar el backing storage

fdisk /root/data/backing_file

Se nos pedirá el número de cabeza, sectores y cilindros.

Command (m for help): x

Expert command (m for help): s

Number of sectors (1-63): 8

El tamaño de un sector es 512 bytes

Expert command (m for help): h

Number of heads (1-256): 16

Expert command (m for help): c

Number of cylinders (1-131071): 1024

Retornando al menú normal:

Expert command (m for help): r

Creando una partición primaria:

Command (m for help): n

Command action

e extended

p primary partition (1-4)

p

Partition number (1-4): 1

First cylinder (1-1024, default 1):

Using default value 1

Last cylinder or +size or +sizeM or +sizeK (1-1024, default 1024):

Using default value 1024

La partición es creada por defecto como una partición Linux, para poder acceder desde Windows cambiaremos el tipo de partición:

Command (m for help): t

Partition number (1-4): 1

Hex code (type L to list codes): b

Changed system type of partition 1 to b (Win95 FAT32)

Imprimiremos la información

Command (m for help): p

Disk /root/data/backing_file: 16 heads, 8 sectors, 1024 cylinders

Units = cylinders of 128 * 512 bytes

Device	Boot	Start	End	Blocks	Id	System
/root/data/backing_file1		1	1024	65532	b	Win95 FAT32

Command (m for help): w

La table de partición ha sido alterada.

Para poder agregar un sistema de archivos podemos conectar el Gadget a Usb Host y dejar que el host haga el trabajo, otra posibilidad es utilizar la utilidad mkdosfs desde linux.

Una vez que tenemos listo nuestro backing_file montaremos los módulos para el FSG:

modprobe s3c2410_udc

modprobe g_file_storage file=/mnt/backing/backing_file stall=0

La opción stall=0 es para poder acceder sin problemas desde Windows, mientras que en la opción file especificamos la ruta del backing_file creado de antemano.

Al conectar el Gadget al Host USB automáticamente será reconocido el nuevo dispositivo ya sea en Linux o Windows.

Como el programa para importar y exportar OTMs deberá acceder también al backing_file, a continuación daremos los pasos para manipular a nuestro backing_file desde nuestro Gadget. La clave es usar los loop device con el programa losetup con la opción –o (offset). Loop Devecie son pseudos device que permiten que un archivo regular sea visto como un block device.

 # losetup -o 4096 /dev/loop0 /root/backing_file

El losetup permite mapear el /dev/loop0 a nuestra partición dentro del backing_file.

 # mkdosfs /dev/loop0

Creamos un sistema de archivos si es necesario.

Para poder accederlo desde la mini2440 lo montamos.

 # mount -t vfat /dev/loop0 /mnt/loop

Una vez que terminamos lo que deseamos hacer debemos de desmontar y desenlazar el dev/loop0 con nuestro archivo.

 # umount /dev/loop0

 # losetup -d /dev/loop0

En los pasos anteriores hemos visto un parámetro offset –o 4096, el cual nos dice el inicio de la data en nuestro archivo.

Para conocer el offset de la partición tipiamos:

fdisk -lu /root/backing_file

Device Boot	Start	End	Blocks	Id	System
/root/data/backing_file1	8	8191	4092	b	Win95 FAT32

La opción –lu nos listará información en unidades de sectores. El offset lo calculamos con la información de la columna start: 8 * 512(tamaño del sector) =4096.

Al seguir todos los pasos mencionados seremos capaces de acceder a nuestro backing_file desde una máquina Host vía USB y desde nuestro propio sistema embebido (Gadget).

CAPÍTULO VI
COMUNICACIÓN ENTRE SISTEMA EMBEBIDO E INSTRUMENTOS

6.1 Introducción.

En este capítulo analizaremos los programas involucrados en la transferencia de información entre los instrumentos de medición y el sistema embebido, los cuales son:

- Programa de Comunicaciones el cual correrá en modo background en el sistema embebido y será el encargado de controlar la interfaz paralela a través del driver implementado.
- Programa del concentrador el cual es el que permite la interacción con la interfaz paralela.
- Programa de los instrumentos físicos el cual manejará los instrumentos de medición.

6.2 Controlador para la Interfaz Paralela.

Debido que nuestra tarjeta concentradora (figura **6.4**) no es auto detectable, +

Cada platform device será enlazado con su platform driver el cual contendrá por lo menos dos funciones obligatorias como son la probe() y remove(), las demás estarán relacionadas al manejo de potencia.

Tanto la platform device y driver tendrán un elemento llamado name a través del cual se enlazarán.

Nuestra platform device será agregada en el archivo machine_mini2440 de tal forma que una vez registrada nuestro platform driver será llamado el método probe para poder inicializar nuestro driver.

En el método probe llamaremos a request_mem_region para solicitar un espacio de direcciones.

Para poder hacer una transferencia de datos hacia o desde nuestra tarjeta de concentrador haremos un mapeo estático usando la estructura map_desc y la función iotable_init() la

cual creará un mapeo entre direcciones físicas y direcciones virtuales.

Los métodos implementados en el driver serán:

open: Hará un request_mem_region si este no pudo ser obtenido desde el método probe().

release: Remueve lo hecho por el procedimiento para la configuración de notificación asíncrona.

read: Lee datos provenientes del concentrador.

write: Escribe hacia el concentrador.

ioctl: Configura el modo de operación.

El driver maneja dos formas de enviar datos al concentrador las cuales son controladas por ioctl, una de ellas es escribir directamente al concentrador los datos y la otra es escribir los datos a una región de memoria del kernel para que sean transmitidos inmediatamente después de haber leído un bloque de datos, para lo cual se usa el parámetro FSBUS_MODE_W.

A través del ioctl podremos configurar un pin de interrupción el cual será utilizado por el concentrador para llamar la atención de la mini2440 una vez que este tenga datos para transferir.

fasync: Este método permitirá generar una señal SIGIO la cual será disparada desde el manejador de interrupciones del driver.

La idea de esta señal es que cuando el concentrador genere una interrupción al embebido este envíe una señal SIGIO al programa de comunicaciones transfiriendo el control a un manejador en el espacio de usuario.

En la figura **6 .1** se puede observar el diagrama de bloque del método read del driver, la primera palabra que deberá venir siempre será un Word fijo (SINCRONISMO) , si viene otro valor diferente reportamos un error de SINCRONISMO.

Como especificamos en el capítulo 4 el concentrador iniciará la comunicación generando una interrupción al embebido enviando una palabra de control que especificará la acción a tomar. Según el protocolo implementado esta palabra hace referencia a un bloque de datos o estado de un instrumento o bloque de estado del concentrador.

En cualquier caso sea el bloque que llegue se verificará el checksum, de tal forma que si se produce un error se enviará un NACK al concentrador caso contrario un ACK.

Del diagrama de flujo observamos que si isCmdEnable es true estamos en el caso de que debemos enviar un comando por lo que cualquier tipo de comando será enviado después de

leer un bloque de datos excepto el primer comando para inicializar el canal 1 el cual es enviado desde el bucle del main.c del programa de comunicaciones escribiendo directamente al concentrador.

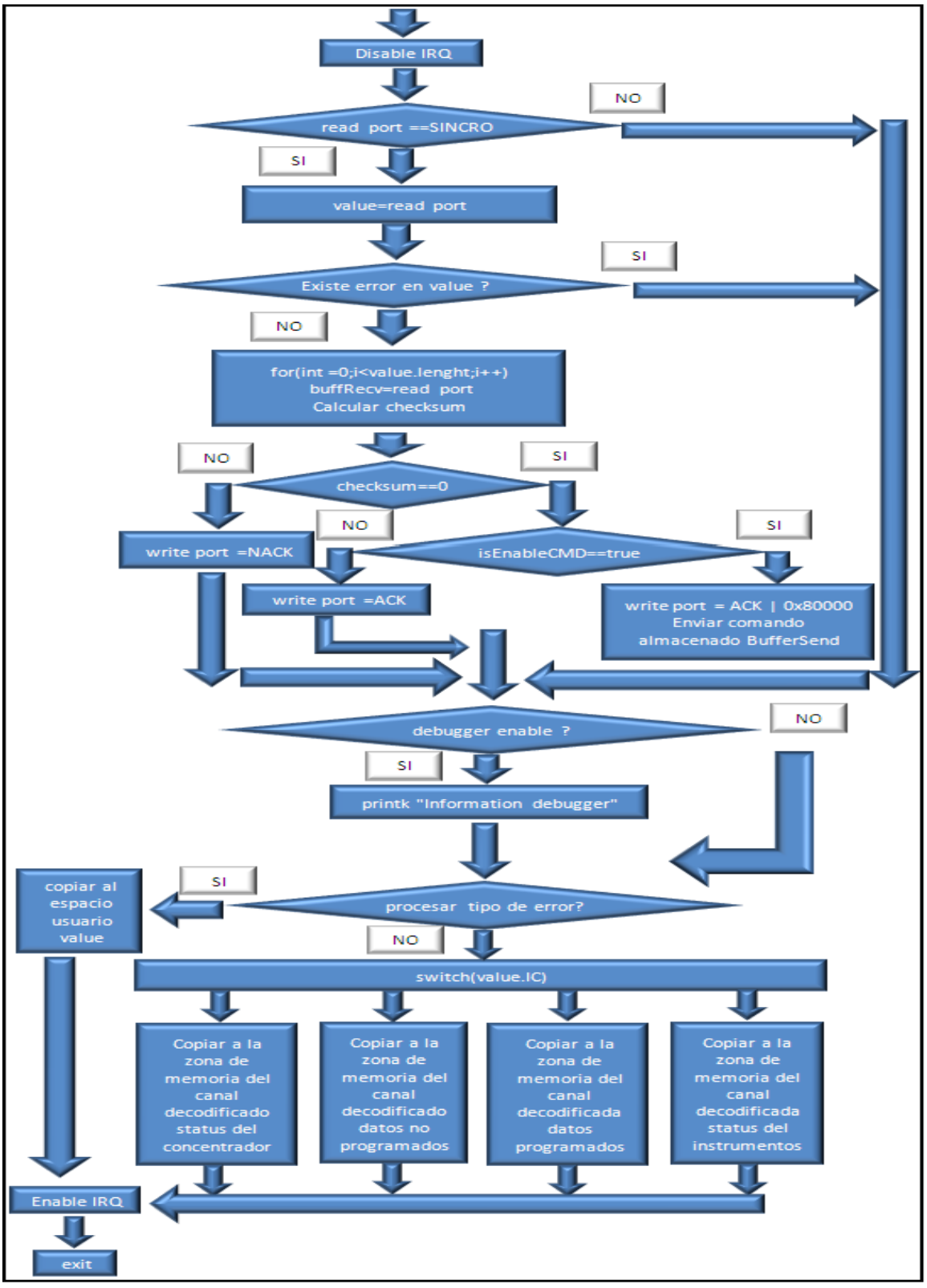


Figura 6.1 Diagrama bloque del método read del driver (Fuente Propia)

En la parte final del método read pasaremos la información recogida desde la interfaz paralela vía el driver al espacio de usuario, es decir al canal correspondiente de la zona de memoria compartida que reservó el programa de comunicaciones.

Para un mejor entendimiento del manejo del driver supongamos que tenemos dos instrumentos en el canal 1 y 2, cada vez que haya un bloque de datos disponibles del instrumento 1 el concentrador enviará una interrupción al sistema embebido de tal forma que se llama al manejador sigio del programa de comunicaciones. Estando en dicho manejador llamaremos al método read el cual leerá los datos del instrumento 1 y los colocará en la zona de memoria compartida del canal 1, posteriormente desde dicho manejador se disparará una señal de Usuario al proceso de instrumento que corresponda indicándole a este que hay datos para graficar o mostrar. El mismo procedimiento para el instrumento 1 es realizado para cualquier otro instrumento que reside en un determinado canal.

Entendemos por canal como una simple región de memoria establecida para recoger una determinada información, llegando a tener dividida nuestra región de memoria compartida en cuatro zonas, lo que nos permite en un determinado momento trabajar simultáneamente hasta con cuatro instrumentos.

El programa de driver para el manejo de la Interfaz Gráfica de Usuario lo tenemos en el **Anexo E.**

6.3 Programa de comunicaciones.

El programa de comunicaciones es el que se encargará de manejar el driver diseñado para la interfaz paralela. Dicho programa correrá en background desde que el equipo se prenda.

El programa de comunicaciones creará una zona de memoria compartida (referida en el capítulo 5) la cual será vista por todos los programas de control de instrumentos donde serán alojados los datos o estados de los instrumentos para su posterior visualización. En la figura **6 .2** observamos el diagrama de flujo del programa de comunicaciones, el cual después de inicializar la región de memoria compartida abrirá el driver configurando para que acepte interrupciones externas.

Además dicho programa se preparará para recibir notificaciones asíncronas por lo que cada vez que le llegue una señal SIGIO se llamará a un manejador previamente preparado.

Después de terminar todas las configuraciones necesarias el programa de comunicaciones ingresará en un bucle donde estará revisando si existe un comando para enviar hacia el concentrador o hacia los instrumentos.

En la figura **6 .3** se observa el diagrama de flujo de nuestro manejador SIGIO. Como mencionamos anteriormente cada vez que se genere una interrupción externa por parte del concentrador el driver enviará una señal sigio al programa de comunicaciones y este ingresará a este manejador.

En dicho manejador se llamará al método read del driver el cual cargará en una parte de la zona de memoria compartida información específica del instrumento en el canal configurado.

Por ejemplo podríamos considerar que se ha configurado el canal 1 para el instrumento X, de tal forma que cuando llamemos al método read este cargará en la zona de memoria que corresponde al canal 1 datos de medición o esatdo de tal instrumento.

Una vez que retornemos del método read se hará una verificación sobre la información recibida, en el caso de que existe un error como por ejemplo un canal equivocado, un NACK, etc retornaremos inmediatamente. Si no existe error se analizará la primera palabra de la región de memoria compartida la cual es la que nos indica que acción hizo el método read.

Si tal acción hace referencia a datos o estado de un determinado instrumento se enviará una señal de usuario al proceso que corresponde a tal instrumento para que este pueda leer dichos valores y mostrarlos en su interfaz gráfica o tomar una acción determinada.

Como parte de la memoria compartida entre el programa de comunicaciones y los programas gráficos de instrumentos se encontrarán cuatro words como se muestra en la figura **6 .2** :

unsigned short int pid1,

unsigned short int pid2,

unsigned short int pid3,

unsigned short int pid4.

En pidx se almacenará el identificador de proceso del programa gráfico de instrumento que tomó el canal x, pudiendo tomar x solamente los valores 1,2,3, o 4. De esta forma cuando el programa de comunicaciones tenga en memoria compartida para un determinado canal data o estado este podrá enviar la señal de usuario respectiva utilizando el pid del canal referido.

El programa de comunicaciones tendrá la capacidad de revisar la actividad de la interfaz paralela, de tal forma que si existe por lo menos un canal activado y por ende un instrumento enviando información periódica y por algún motivo durante un tiempo el

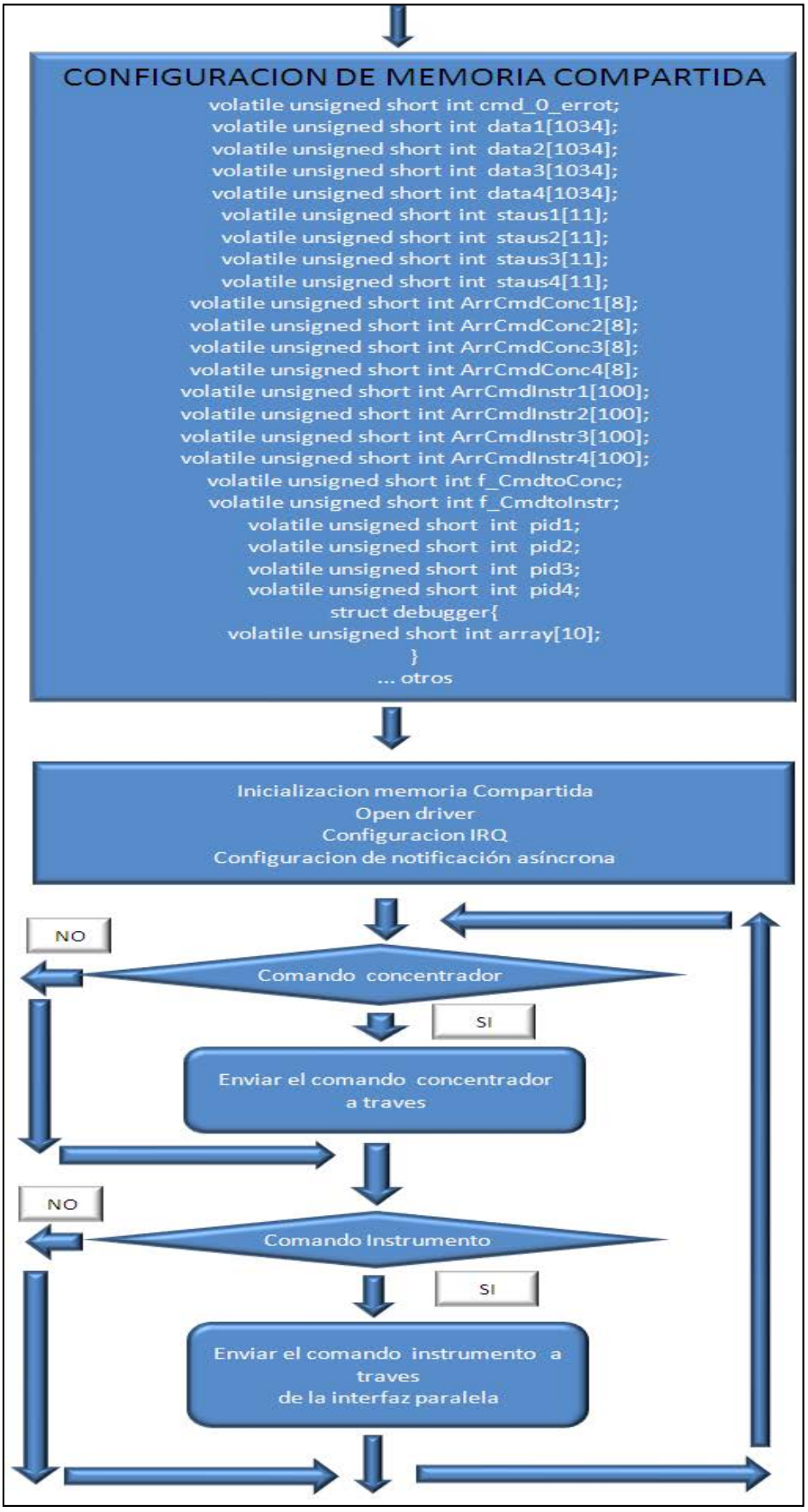


Figura 6.2 Diagrama bloque del programa comunicaciones (Fuente Propia)

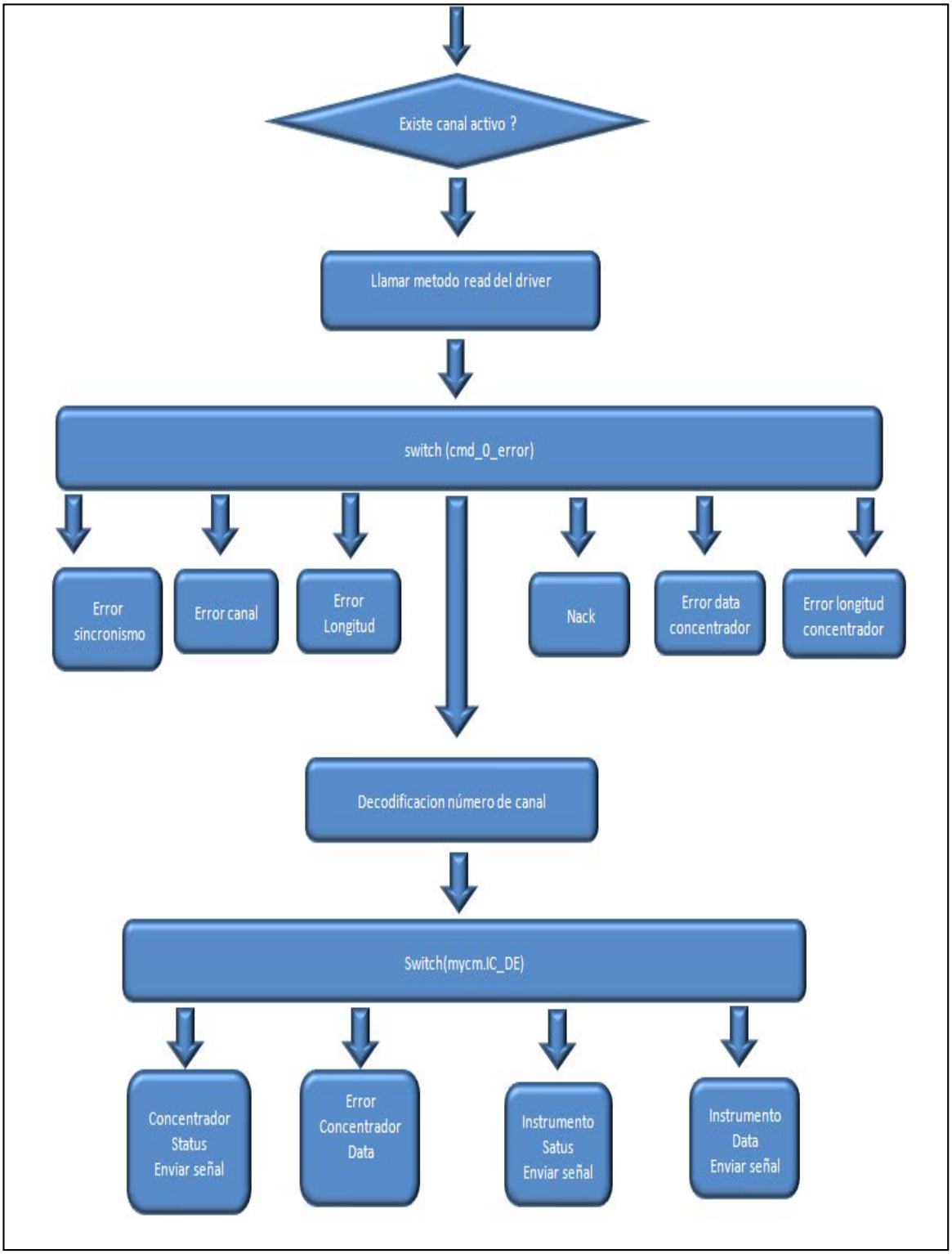


Figura 6.3 Diagrama del manejador sigio del programa comunicaciones (Fuente Propia)

concentrador no envía dicha información, el programa de comunicaciones enviará un comando de reset para el concentrador y este responderá con un estado el cual posteriormente llegará al programa gráfico de instrumento. El programa de comunicaciones lo podemos observar en el **Anexo** C.

6.4 Programa del concentrador.

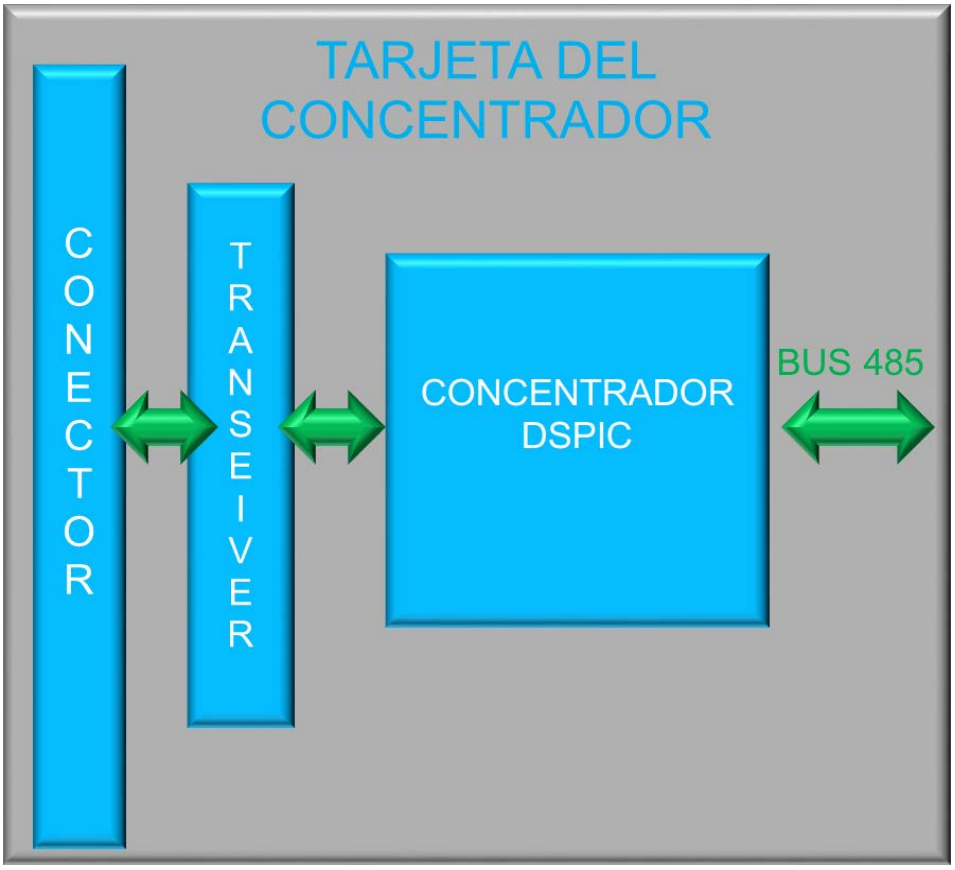


Figura 6.4 Transferencia datos desde el programa instrumentos y la OTM (Fuente Propia)

El dspic concentrador tendrá la función de comunicar los instrumentos con el sistema embebido.

Los instrumentos enviarán la información solicitada a través del bus 485, el concentrador almacenará dicha información y se la transmitirá al programa gráfico de instrumento pertinente usando la interfaz paralela de 16 bits.

De la figura **6 .4** podemos observar que usaremos un transceiver bidireccional de 16 bits para comunicar el bus del sistema embebido a través de su conector CON5 con el dspic concentrador.

De la figura **6 .5** podemos ver el esquemático del concentrador, dentro del cual podemos observar el uso de los integrados ISL83485 para la comunicación 485 y el transceiver 74LCX16245.

El pin 2,3 del ISL83485 habilitará la recepción y transmisión de datos respectivamente. En el diseño siempre habilitaremos la recepción de datos por lo que los dspic de los instrumentos y el propio concentrador siempre estarán escuchando incluso cuando ellos mismos envíen información. Esto lo hacemos para que los instrumentos en todo momento

sean capaces de ser direccionados por el concentrador.

El uart se configurará para comunicación a 9 bits. Con el bit ADDEN seteado el receptor ignorará data cuando el noveno bit de la data sea 0, esto es la base para que el concentrador direccione en un determinado momento a un instrumento.

El 74LCX16245 será usado para aislar el bus del sistema con el concentrador. Los pines 1,24 (TR1,TR2) unidos determinará la dirección de flujo, mientras que los pines 48,25 (OE1,OE2) unidos habilitará la transferencia de datos a nivel bajo.

El chip select del banco 5 nGC5 está conectado a OE1, OE2, esto lo hacemos para que cuando exista una lectura o escritura en este banco se habilite la transferencia de datos. El estado por defecto del nGC5 es a uno por lo que nuestro 74LCX16245 estará en alta impedancia aislando el bus del sistema.

Podemos observar la presencia de una compuerta OR, la salida nWait está conectada la señal nWait del sistema embebido mientras que las entradas son nGC5 y EnableW.

El EnableW está por defecto a cero, de tal forma que cuando iniciemos una lectura o escritura en el banco 5 automáticamente habilitaremos el wait. Esto lo hacemos debido a que el acceso a memoria del sistema embebido es mucho más rápido que las operaciones necesarias por parte del dspic.

El EnableW será usado para liberar el estado de Wait colocándose a uno durante 1 ciclo de instrucción del dspic , tiempo suficiente para que nWait se coloque a uno permitiendo que termine el ciclo de acceso al bus.

El programa del concentrador estará constituido por los archivos *.c

Clock.c: Este archivo tiene todas las rutinas y manejadores de timers empleados para acciones a realizar producto de errores en la comunicación 485, además de uno exclusivo configurado cada 2ms para testear a los instrumentos si tiene o no información.

Main.c: Inicialización del uart , interfaz paralela de 16 bits, pll, timers.

El pll es configurado para trabajar a FCY=36.864MHZ.

De la figura **6 .5** observamos que el main.c tiene las funciones Supervisor y Each2Mseg.

La función Supervisor será la encargada de enviar información al sistema embebido a través de la interfaz paralela cuando un canal configurado tenga disponible información a enviar.

La función Each2Mseg cada dos milisegundo preguntará al instrumento previamente programado si tiene datos a transferir.

ProgPin.c: Rutinas para configurar los pines, en dicho archivo configuraremos entre otras

cosas los pines que utilizaremos para manejar la interfaz paralela.

Paralelo.c: Este archivo tendrá la responsabilidad de manejar la comunicación con el sistema embebido.

De la figura **6 .6** observamos el diagrama de flujo de nuestra máquina de estado, cada vez que exista una lectura o escritura en la interfaz paralela se generará una interrupción en el dspic ingresando a un determinado estado. La señal de interrupción hacia el dspic será la señal CS (chip select) la cual va hacia nivel bajo cada vez que el embebido a través de su driver intente una lectura o escritura en el bus.

La máquina de estados constará de 12 estados en total, siendo el estado de reposo kRead0.

KRead0: Espera a recibir un comando.

KRead1: Lectura de los datos asociados al comando recibido.

KRead2: Leer checksum.

KRead3: Enviar Acknoledge.

KWSta0: Enviar byte de cabecera.

KWSta1: Enviar el Comando Estado del canal programado

KWSta2: Enviar datos de Estado del canal seleccionado.

KWDat0: Enviar cabecera.

KWDat1: Enviar comando.

KWDat2: Enviar datos relativos al comando enviado en el paso previo.

KWCom1: Enviar checksum.

KWCom2: Recibe el Acknoledge.

I485.c: Este archivo se encargará de la comunicación 485 con los instrumentos, esta será basada también en una máquina de estado cuyo diagrama la observamos en la figura **6 .7**.

El manejador 485 constará de 5 estados, siendo KRx0 el estado inicial.

KRx0: Espera que comando sea recibido.

KRx1: Recibe comando de respuesta.

KRd0: Recibe # words

KRd1: Recibe LOW de un word

KRd2: Recibe HIGHT de un word

Debido a que nuestra comunicación con el embebido es a 16 bits, necesitamos armar la palabra para lo cual emplearemos los estados Kd1 y Kd2. El programa del concentrador lo podemos ver en el **Anexo** A.

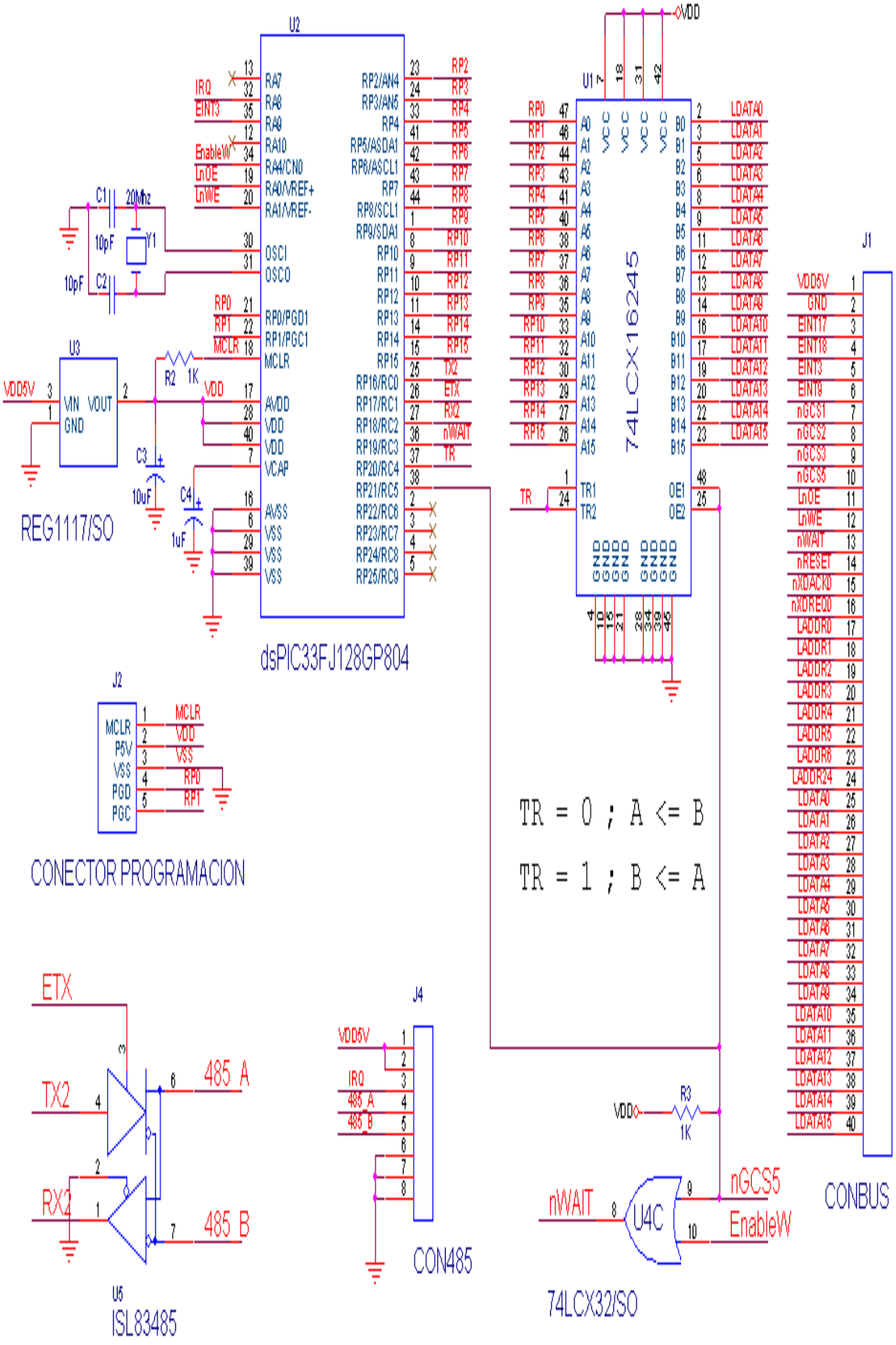


Figura 6.5 Esquemático de la Tarjeta del Concentrador (Fuente Propia)

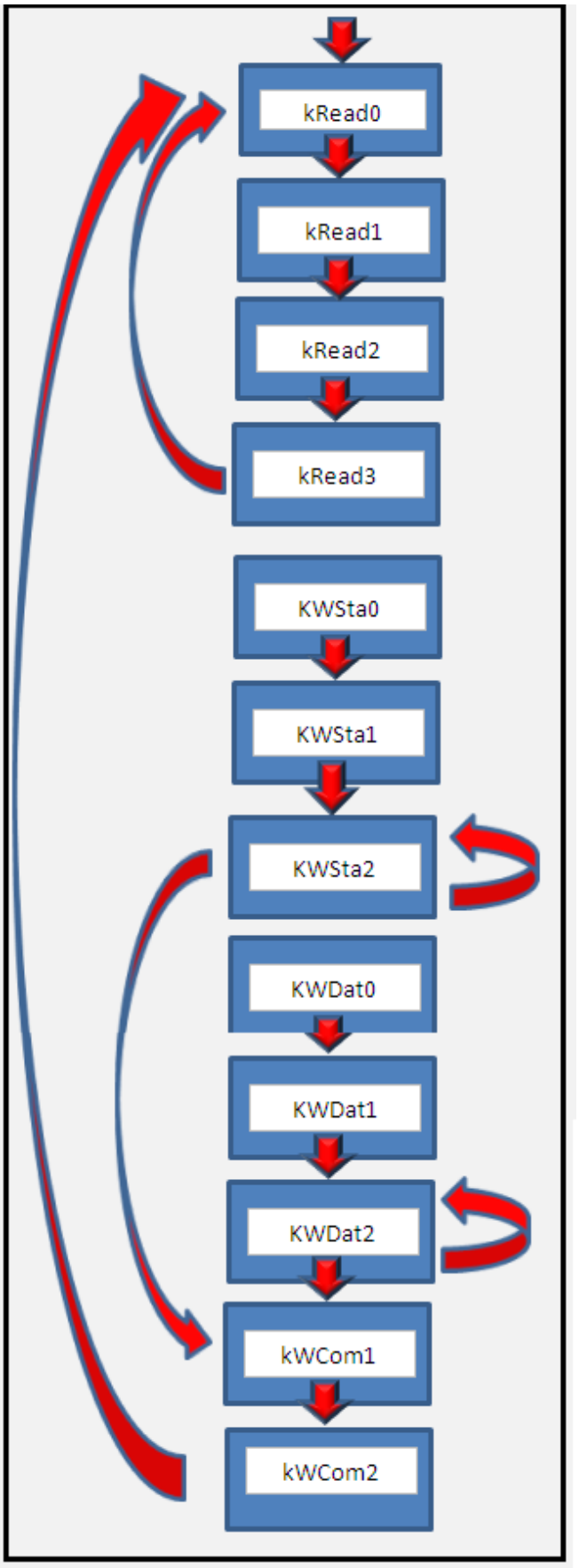


Figura 6.6 Diagrama bloque del Paralelo (Fuente Propia)

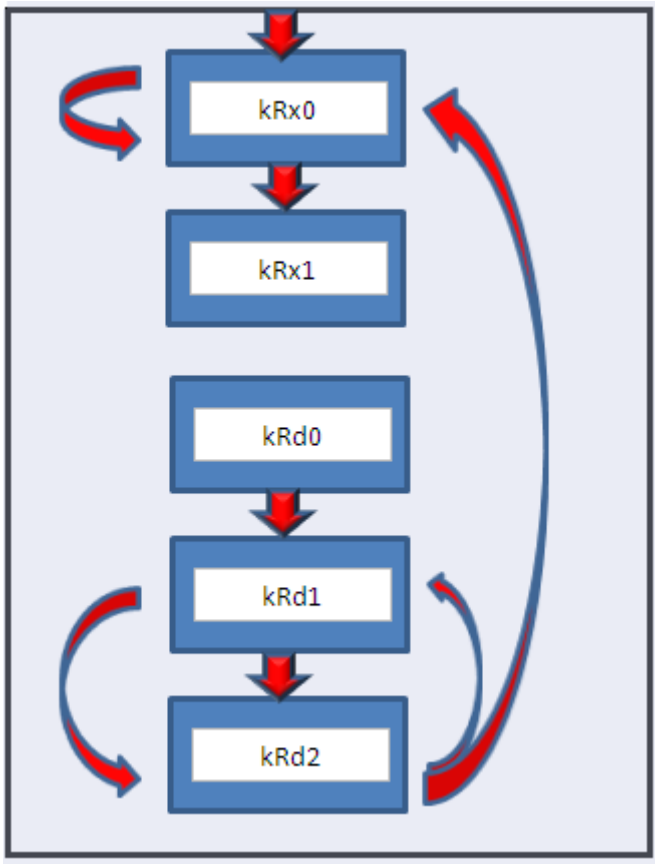


Figura 6.7 Diagrama bloque del manejador 485 (Fuente Propia)

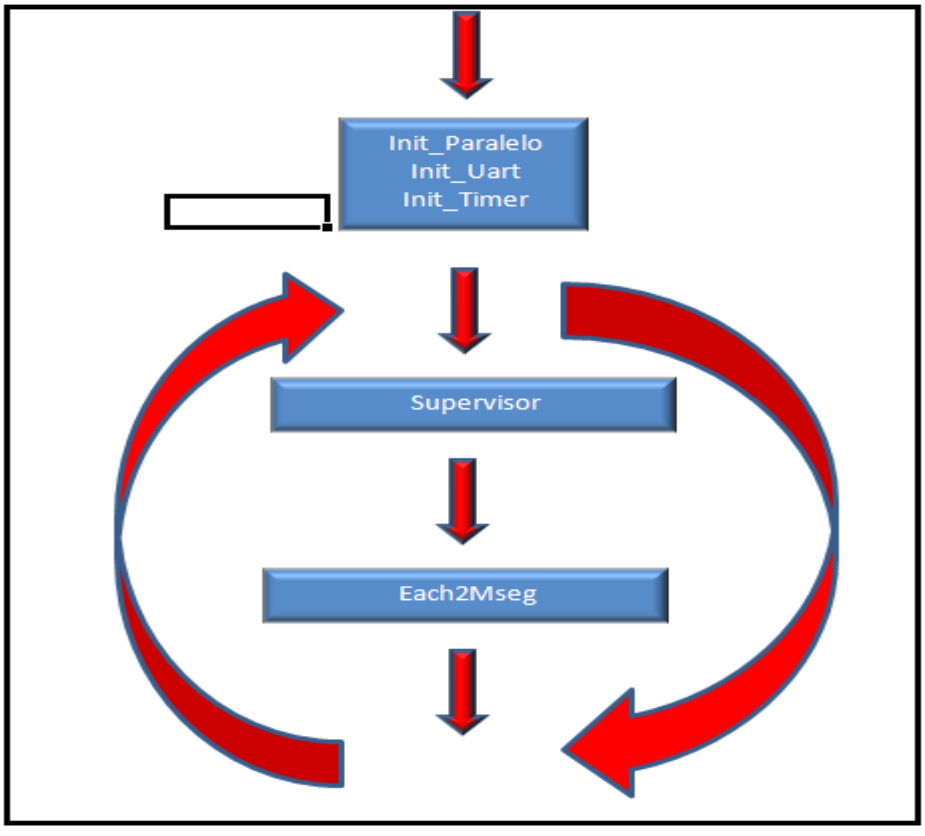


Figura 6.8 Diagrama bloque del main.c (Fuente Propia)

6.5 Programa de instrumentos físicos.

Para poder comprobar nuestro sistema de adquisición de datos programaremos el instrumento TSI visto en el capítulo 5 siguiendo el protocolo implementado para la comunicación concentrador e instrumento establecido en el capítulo 4.

De la figura **6 .9** podemos observar que le programa de instrumentos básicamente tiene dos bloques, uno dedicado a la comunicación 485 con el concentrador y el otro a la comunicación serie con el TSI. Ambos bloques se manejarán en modo de interrupción tanto para la recepción y envío de datos.

Se programará el TSI para obtener 50 muestras cada 10ms por lo que usaremos los siguientes comandos DBFTP0050 y SSR0010. El TSI posee un buffer interno de 50 bytes que permite recordar por lo menos dos comandos. Para poder mantener la transferencia de 50 muestras tenemos que reenviar el comando DBFTP0050. Si deseamos mantener la carencia de 10ms por muestra enviaremos DBFTP0050 dos veces al inicio y una vez que termina el primer grupo de 50 muestras enviamos otra vez el comando.

Después de ser configurado el TSI el instrumento enviará las muestras serialmente a 38400bps.

Para recepcionar los datos utilizaremos una lógica de doble buffering, tenemos dos buffers para recolectar datos y enviarlos al concentrador, si llenamos un buffer y no lo hemos enviado al concentrador comenzaremos a llenar el segundo buffer pero si no hemos transferido el primer buffer y volvemos a recibir muestras entraremos en condición de overrun.

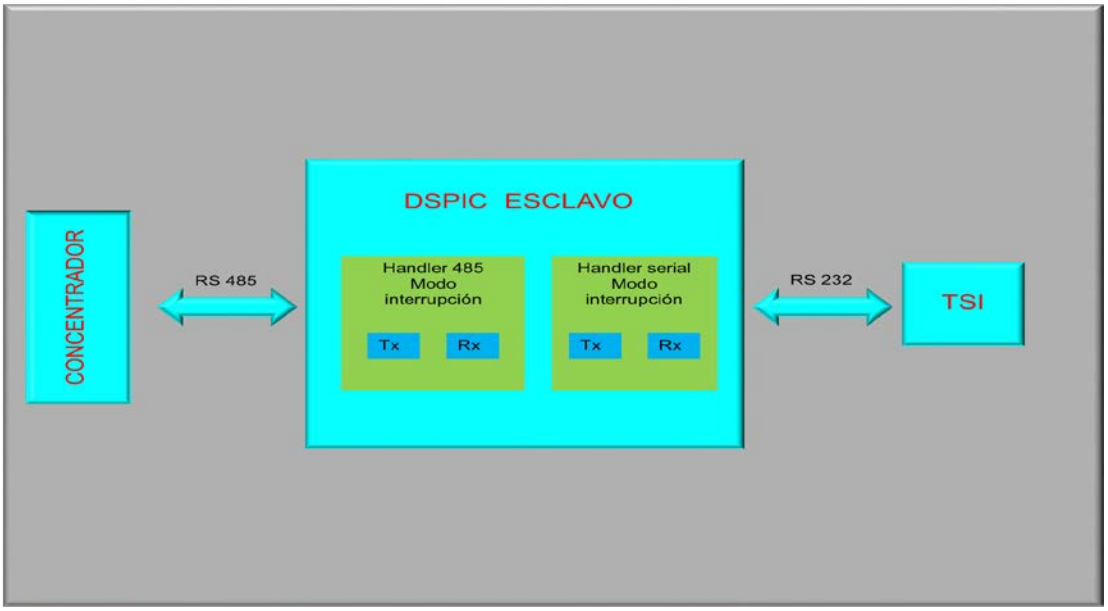


Figura 6.9 Diagrama genérico del programa de instrumento (Fuente Propia)

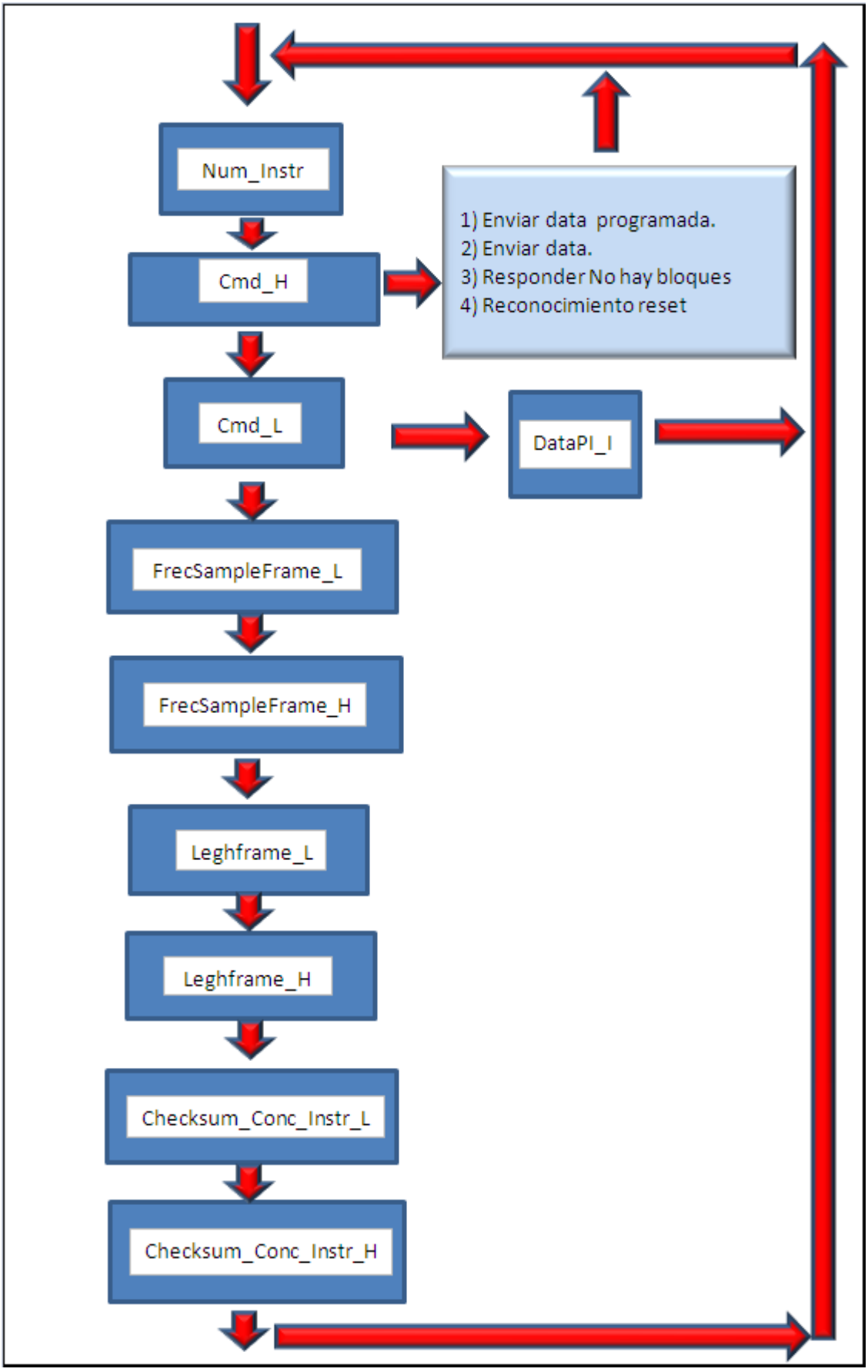


Figura 6.10 Manejador de lectura de comunicación 485 (Fuente Propia)

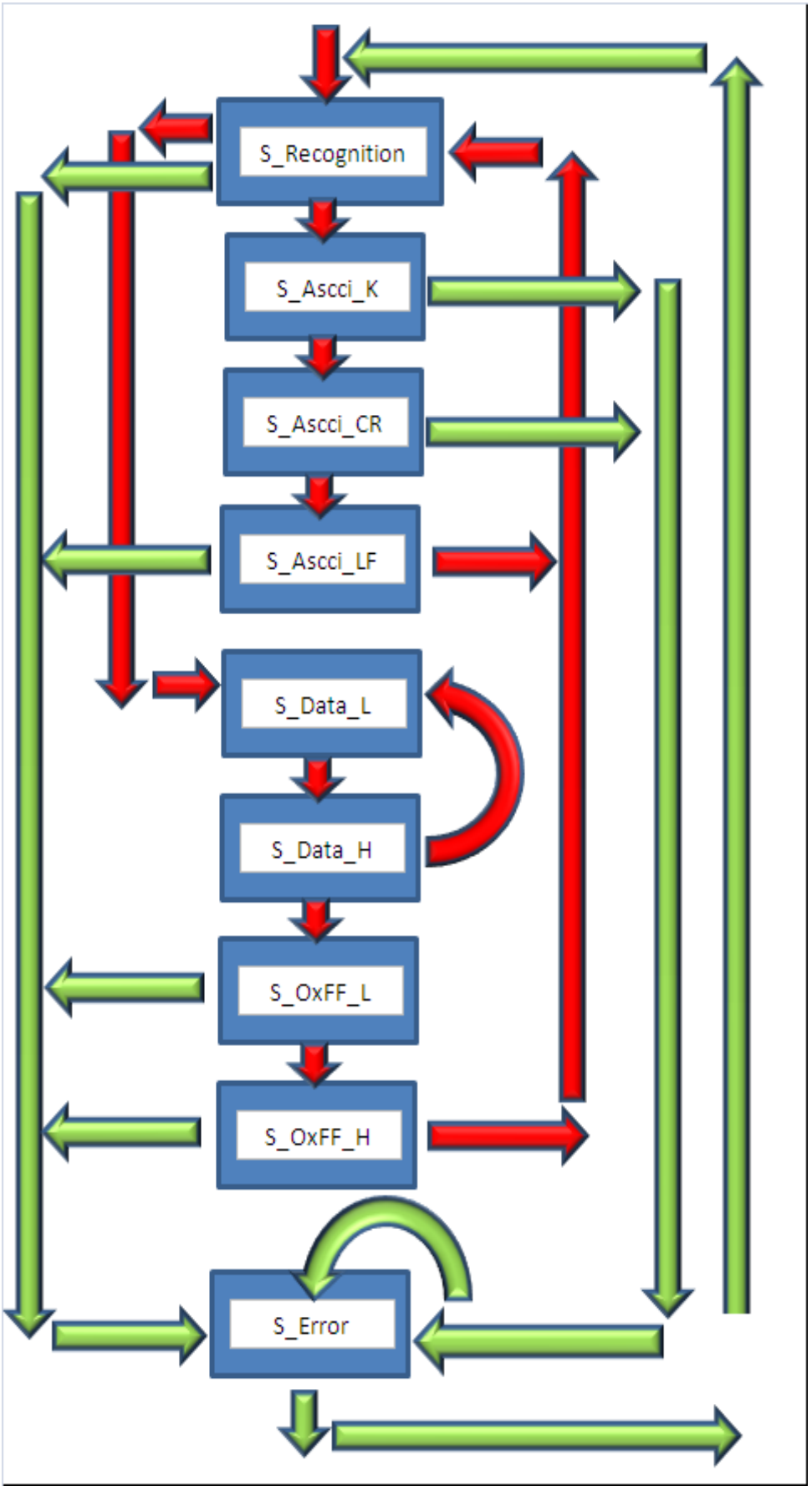


Figura 6.11 Manejador de lectura de comunicación rs232 (Fuente Propia)

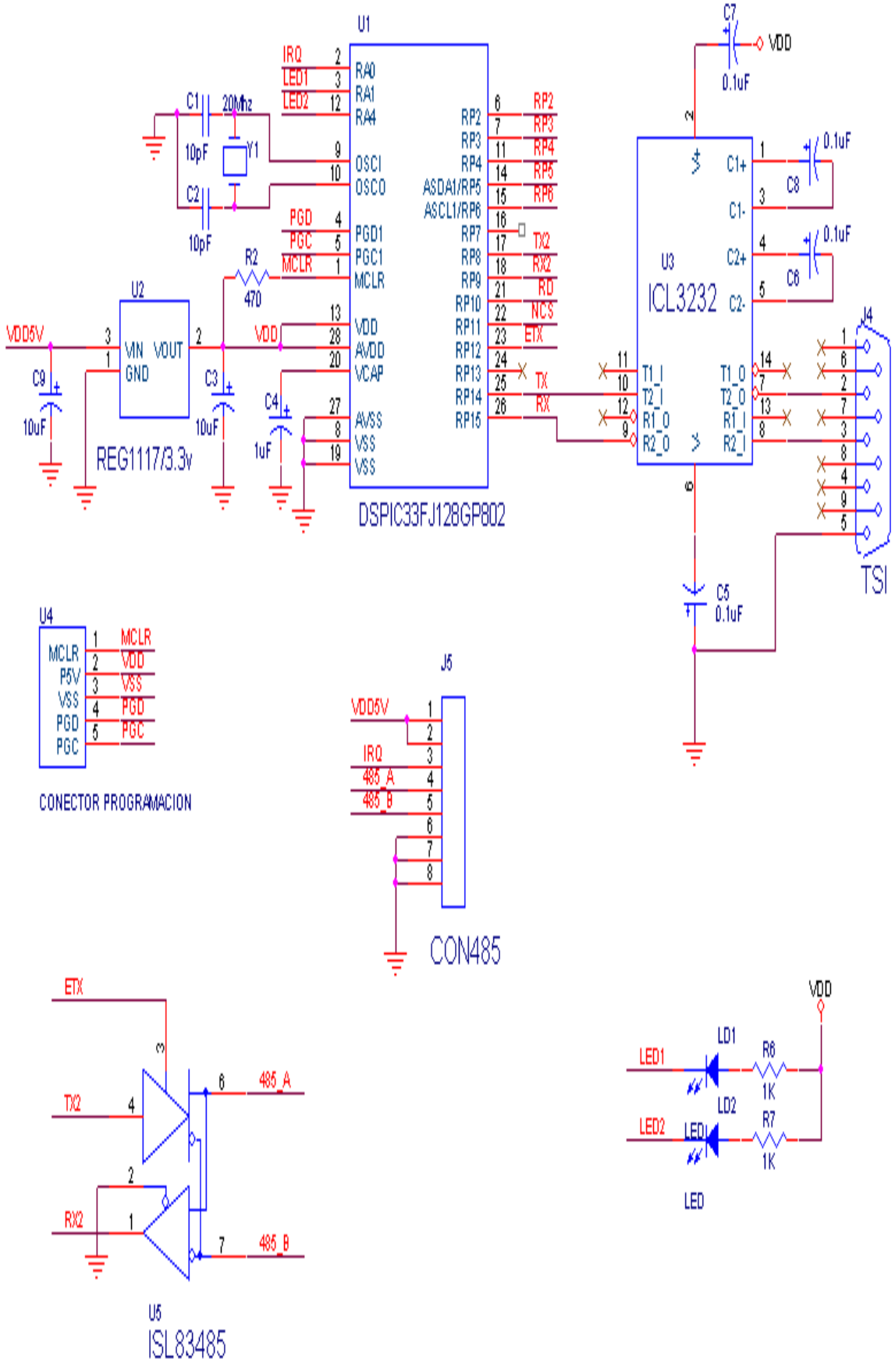


Figura 6.12 Esquemático Instrumento TSI (Fuente Propia)

El programa de instrumentos estará constituido por los archivos *.c

Clock.c: En este archivo tiene todas las rutinas y manejadores de timers empleados para acciones a realizar producto de errores en la comunicación 485.

Main.c: Inicialización de los uarts y pll.

El pll es configurado para trabajar a FCY=36.864MHZ.

Configuración de timers y uarts y pines.

PrigPin.c: Rutinas para configurar los pines.

rs232_Instr.c : Recepción y envío de información desde y hacia el TSI, esto es realizado por interrupciones.

Slave485.c: Recepción y envío de información desde y hacia el concentrador, esto es realizado por interrupciones.

El la figura **6 .10** observamos el diagrama de flujo del manejador de lectura el cual está constituido de varios estados y está basado en el protocolo de comunicación concentrador instrumentos implementado en el capítulo 4.

Num_Instr : Es el estado inicial de todos los instrumentos en el cual se establecerá la conexión con el concentrador, cada instrumento tendrá un código predefinido, por ejemplo el dspic esclavo que observamos en la figura **6 .9** que manejará la comunicación con el TSI tendrá el código 4.

Como comentamos anteriormente todos los instrumentos tendrán el integrado ISL83485 el cual tendrá el pin 2 a cero con lo cual nos aseguramos que todos tengan la capacidad de escuchar o recibir datos, de esta forma si el concentrador desea comunicarse con el instrumento de código 4 (TSI) enviará el código 4 con el bit nueve a 1.

Todos los dspics esclavos tienen por defecto habilitado el bit de detección de dirección en la comunicación de tal forma que todos leerán el código 4 enviado pero solamente el que le corresponda tal código deshabilita el bit de detección de direcciones de tal forma que se inicie la transferencia de datos con este solamente y los demás ignorarán los datos enviados por el concentrador, después de terminar la transferencia de datos el instrumento vuelve a habilitar el bit de detección de direcciones y se coloca en el estado Num_Instr.

Cmd_H : Estado donde se analizará una determinada acción, en caso de que se trate de una lectura de datos o estado según el protocolo implementado , si existen datos a continuación se enviarán al concentrador dichos datos los cuales pueden ser datos programados que se van a graficar o datos producto de un comando de petición previo como por ejemplo datos de tendencia, si no hay datos se responde con un no hay bloques.

Además en este estado también podremos responder con un reconocimiento de reset. La transferencia de información que se hace en este estado desde el instrumento hacia el concentrador para todos los casos mencionados se hará por interrupciones.

Cmd_L: Si llegamos a este estado entramos a dos posibles opciones, o bien estamos ante el caso de que se requiera el envío de un comando del programa instrumento al instrumento o que se requiera configurar el instrumento.

DataPI_I: En este estado almacenaremos información de un comando del programa instrumento al instrumento.

FrecSampleframe_L , FrecSampleframe_H, Lengframe_L, LengthFrame_H, Checksum_Conc_Inst_L, Checksum_Conc_Inst_H : En estos estados se da la configuración del tiempo de muestreo como la longitud del buffer de datos a transferir. Antes de empezar la transferencia de data o estado primero se deberá configurar el instrumento pasando por todo estos estados, en el caso de la prueba del TSI una vez llegado al estado de verificación Checksum_Conc_Inst_H enviamos el comando SSR0002 y dos veces DBFTP0050 a través de una transferencia de datos por interrupciones.

En resumen la secuencia inicial de todo instrumento sería:

El concentrador lo direcciona, posteriormente se establece el tiempo de muestreo y longitud del buffer de datos a transferir, en este punto dependiendo del instrumento específico se programará este para su funcionamiento, después se inicia la recolección de información empleando lógica de doble buffer.

El concentrador preguntará cada 2ms al instrumento si tiene datos disponibles, en caso que los tenga se inicia un transferencia de lo contrario no se envía nada y se termina la comunicación. Esto se repetirá cada 2ms con todos los instrumentos programados en un determinado momento.

En la figura **6 .11** podemos observar el manejador de interrupciones para poder recibir los datos del TSI, al igual que en el caso anterior se trata de una máquina de estados.

S_Recognition:

En este estado se inicia el reconocimiento de la llegada del número 0 o ascci 0 , cualquier otro caso se pasará al estado S_Error.

S_Ascci_K:

Reconocimiento del ascci K caso contrario se pasará al estado S_Error.

S_Ascci_CR:

Reconocimiento del ascci CR caso contrario se pasará al estado S_Error.

S_Ascci_LF:

Reconocimiento del ascci LF caso contrario se pasará al estado S_Error.

S_Data_L; S_Data_H: Se recolectará los datos de flujo, temperatura, presión siendo almacenados alternadamente en buffer A o B. Una vez que se tenga un buffer completo se setea una variable booleana usada para iniciar la transferencia del bloque al concentrador.

S_0xFF_L:

Reconocimiento de OxFF caso contrario se pasará al estado S_Error.

S_0xFF_H:

Reconocimiento de OxFF caso contrario se pasará al estado S_Error.

S_Error:

Cuando llegamos a este estado por algún error en la transferencia de datos se buscará la sincronización buscando la ocurrencia consecutiva de dos llegadas de 0xFF. Si se da esto pasaremos al estado S_Recognition nuevamente. Cabe mencionar que cada vez que se generá un error se dispará un timer de 455ms el cual permitirá enviar periódicamente el comando DBFTP0050 para que no se detenga la transferencia de datos, una vez que nos resincronizamos el timer se detiene. El programa de instrumentos lo podemos ver en el **Anexo** B.

CAPÍTULO VII
EVALUACIÓN DE COSTOS

7.1 Componentes empleados en la elaboración del sistema diseñado.

En este capítulo mencionaremos el costo de los componentes empleados para el diseño del sistema propuesto.

TABLA Nº7.1 COSTOS

COMPONENTE	CANT.	P. UNID	P. TOTAL	CODIGO DIGIKEY
Mini2440 + LCD 7" Innolux	1	$194,95	$194,9500	
Cámara CMOS	1	$8,95	$8,9500	
Carcasa Aluminio	1	$35,00	$35,0000	
ISL83485IBZ-ND	2	$1,55	$3,1000	ISL83485IBZ-ND
Tranceiver Bidireccional 16 bits	1	$0,90	$0,9000	74LCX16245MTD-ND
DSPIC33FJ128GP804	1	$6,54	$6,5400	DSPIC33FJ128GP804-I/PT
Regulador 3.3V 0.25Ma	2	$0,44	$0,8800	MCP1700T3302ETTCT-ND
Gate OR	1	$0,57	$0,5700	74LCX32MXCT-ND
RES SMD 1K	100	$0,0195	$1,9500	RHM1.00KCCT-ND
CAPACITOR 10UF SUPERFICIAL	50	$0,3348	$16,7400	709-1065-1-ND
CAPACITOR 10PF SUPERFICIAL	50	$0,0358	$1,7900	399-1108-1-ND
CAPACITOR 1UF SUPERFICIAL	100	$0,0141	$1,4100	311-1361-6-ND
CAPACITOR 0.1UF SUPERFICIAL	50	$0,0330	$1,6500	339-9159-1-ND
RECEPTACLE 40 POS DUAL 2MM GOLD	1	$4,28	$4,2800	952-1363-5-ND
HEADER 5 POS 2.54MM GOLD	2	$2,00	$4,0000	SAM1111-05-ND
XTAL 20MHZ	10	$0,2890	$2,8900	535-10232-1-ND
ICL3232CBNZ-ND	1	$1,86	$1,8600	ICL3232CBNZ-ND
DSPIC33FJ128GP802	1	$6,16	$6,1600	DSPIC33FJ128GP802-I/SO
LED MONTAJE SUPERFICIAL ROJO	20	$0,36	$7,2000	516-1429-1-ND
LED MONTAJE SUPERFICIAL VERDE	10	$0,36	$3,6000	516-1434-1-ND
LED MONTAJE SUPERFICIAL AMARILLO	10	$0,36	$3,6000	516-1433-1-ND
LAYOUT TARJETA CONCENTRADOR	1	$20,00	$20,0000	
LAYOUT TARJETA INSTRUMENTO	1	$15,00	$15,0000	
		$300,01	$343,02	

La mayoría de los componentes mencionados en la Tabla **7.1** se han comprado en la corporación DIGI-KEY, el sistema embebido en Industrial ARMWorks. Las tarjetas del concentrador e instrumento fueron elaboradas en nuestro país y ambas son de dos capas.

Como podemos observar la Mini2440 + LCD 7" Innolux abarcan aproximadamente el 65% de nuestro costo de producción.

En un futuro para reducir nuestro costo de producción no se comprará la Mini2440 + LCD 7” juntos. En vez de la mini2440 se adquirirá la micro2440 la cual es una tarjeta en la cual solamente se encuentra el procesador, la sdram, la nand y nor flash y tres conectores a través de los cuales tenemos accesibles los pines del procesador. Usando esta tarjeta nos permitirá realizar nuestra propia tarjeta base personalizada.

El LCD 7” que proporciona Industrial ARMWorks cuesta 116$. El precio es elevado por que FriendlyArm para incorporarle la capacidad de touchscreen ha diseñado una tarjeta intermedia para llevar al conector del LCD en el sistema embebido los cuatro hilos de la lámina de touch.

Además FriendlyARM le agregó electrónica para que dicho LCD sea usado por otros sistemas embebidos como la mini6410.

Para reducir el costo se pretende comprar directamente el LCD al fabricante Innolux además obtendremos las láminas para pantalla táctil por separado

CONCLUSIONES Y RECOMENDACIONES

1. El uso del PDA permite que el mantenimiento de los equipos sea más rápido y eficiente.
2. El sistema de adquisición de datos puede transferir hasta 512 words aproximadamente en 1.1 ms.
3. La señal de wait extiende el ciclo de lectura aproximadamente en 2 us.
4. A pesar de que el Proyecto Qtopia en su versión 2.2 ya nos es mantenido por sus desarrolladores nos proporciona una interfaz gráfica estable.
5. El sistema chino mini2440 es apto para realizar proyectos medianos a complejos corroborando con la presente tesis el slogan que tienen por parte de sus fabricantes de ser líder en al área automotriz.
6. Cambiar Qtopia elaborando nuestro propio disparador de aplicaciones usando las librerías Qt5.1 .
7. Utilizar la familia dspic33EP en vez de la dspic33f para trabajar con una mayor frecuencia de ciclo de instrucción de tal forma que se reduzca el tiempo de ciclo de lectura.
8. Reemplazar nuestro concentrador basado en un dspic33f por un FPGA para lograr un sistema de adquisición de datos más rápido.
9. Si se va a utilizar cables para comunicar el conector CON5 del sistema embebido con la tarjeta del concentrador deberemos aislar el cable que conduce la señal de wait de la de datos para evitar que por ruido se libere de forma inesperada un ciclo de lectura ocasionando una lectura incorrecta.
10. Remplazar la mini2440 por la micro2240 la cual contiene CPU, Ram, Flash , Headers , etc para poder diseñar nuestra propia tarjeta de desarrollo.

ANEXO A
PROGRAMA CONCENTRADOR

```
/*****************************************************************************
File: Paralelo.c
*****************************************************************************/
#include <p33FJ128GP804.h>
#include  "MMetro.h"
#include  "Estructuras.h"
#define         K_Ack                   0x2bcd
#define         K_Nack                  0x7234
#define         Cabecera                0x55aa
// Pines que controlan el Wait
#define         T_WAIT                  _TRISC3         // RP19 ve el Wait del BUS.
#define         T_CS                    _TRISC5         // RP21 ve el CS del BUS.
#define         T_EnableW               _TRISA4         // "1" Habilita Wait.
#define         T_Int                   _TRISA9         // Pin que interrumpe.
#define         T_TR                    _TRISC4         // "0" A <= B
#define         TLnOE                   _TRISA0         // Lee el "OE" del BUS.
#define         TLnWE                   _TRISA1          // Lee el "WR" del BUS.
// Pines que controlan el Wait
#define         B_WAIT                  _RC3            // RP19 ve el Wait del BUS.
#define         B_CS                    _RC5            // RP21 ve el CS del BUS.
#define         B_EnableW               _LATA4          // "1" Habilita Wait.
#define         P_Int                   _LATA9          // Pin que interrumpe.
// Pines que controlan              74lcx16245
#define         B_TR                    _LATC4          // "0" A <= B
// Pines que monitorean el BUS
#define         BLnOE                   _RA0            // Lee el "OE" del BUS.
#define         BLnWE                   _RA1            // Lee el "WR" del BUS.
#define         F_RD                    0
#define         F_WR                    1
enum Estado {
KRead0,KRead1,KRead2,KRead3,
KWSta0,KWSta1,KWSta2,
KWDat0,KWDat1,KWDat2,
```

```
KWCom1,KWCom2
};
enum WorkG {ChannIdle,ChannSendData,ChannSendStatus};
enum Work485 {
W485_Loop,W485_WPool,W485_W1Rsp,W485_W2Rsp,W485_WBRsp,W485_WIni,W
485_WRst,W485_WCmd
};
volatile FmtArrEstCon ArrEstCon[4];
volatile FmtArrCmdCon ArrCmdCon[4];
volatile FmtCmd MComando;
volatile FmtCmd MStatus;
volatile FmtInst MCmdInst;
unsigned int BuffDat[4][BUFFSIDE]
unsigned int BuffStat[4][25];
unsigned int BuffCmd[4][25];
unsigned int Tope[4];
unsigned int CntLoad;
unsigned int Numero;
unsigned int Comando;
unsigned int ChkSum;
unsigned int RspAck;
unsigned int Acknoledge;
volatile unsigned char ScanChan;
volatile unsigned char Channel;
volatile unsigned char Semaforo;
volatile unsigned char Ruta;
volatile unsigned char SoD;
volatile unsigned char Free485;
volatile unsigned int S_Divisor;
volatile unsigned int S_DatBlock;
volatile unsigned int S_NumBlock;
volatile unsigned int CntTout;
unsigned int *PntBuf[4];
```

```
// Nuevas Variables.
unsigned char Bus485;                    // Indica si se está utilizando el Bus 485.
volatile FmtBuff StBuffer[4];
volatile unsigned int VrfChk;
volatile unsigned int ChkCal;
// Definición de Funciones.
unsigned char Decode(unsigned char);
void Recibido(unsigned char);
void EndChann(void);
void Clear_Buffer(unsigned char);
void InitBuff(void);
void SetBfFull(void);
void DoCmd(void);
void Init_Paralelo(unsigned char Rst){
  unsigned char j;
  TRISB                    = 0xffff;          // PIC en Lectura.
  B_TR              = 0;                      // BUS en Read.
  B_EnableW  = 1;                             // Disable Wait.
  P_Int             = 1;                      // Pin que interrumpe.
  P_Irq             = 1;
  P_Prb1 = 0; P_Prb2 = 0; P_Prb3 = 0; P_Prb4 = 0; // Pin de prueba.
  T_Prb1 = 0;  T_Prb2 = 0; T_Prb3 = 0; T_Prb4 = 0;
  T_WAIT                   = 1;
  T_CS                     = 0;
  T_EnableW                = 0;
  T_TR                     = 0;
  TLnOE                    = 1;
  TLnWE                    = 1;
  T_Int                    = 0;
  T_Irq                    = 0;
  Semaforo                 = 0;
  Free485 = True;
  ScanChan = 0;
```

```
  IC1CON = 0;
  IC1CONbits.ICM = 2;                 // Captura en flanco de bajada.
  IPC0bits.IC1IP = Prioridad_Input1;
  IFS0bits.IC1IF = 0;                 // Clear IC1 Interrupt Flag.
  IEC0bits.IC1IE       = 1;           // Enable IC1 Interrupt.
  if (Rst == 0){
        InitBuff();
        MComando.EstCmd = 0; MStatus.EstCmd = 0;
  }
  for (j=0;j<4;j++){
        if (Rst == 0) Clear_Buffer(j);
        PntBuf[j] = (unsigned int *)(&BuffDat[j][0]);
  }
   Ruta = KRead0;
   B_EnableW = 0;                     // Enable Wait.
}
// ---------------------------------------------------------------
void Clear_Buffer(unsigned char j){
  unsigned char i;
        for (i=0;i<6;i++){ArrCmdCon[j].Buff[i] = 0;}
        for (i=0;i<6;i++){ArrEstCon[j].Buff[i] = 0;}
}
// ---------------------------------------------------------------
void Init_Buffer(){
  unsigned int i,j;
        for (j=0;j<4;j++){
              for (i=0;i<BUFFSIDE;i++) BuffDat[j][i] = j;
        }
}
// ---------------------------------------------------------------
void FreeWait(void){
  B_EnableW = 1;                          // Disable Wait.
  B_EnableW = 0;                          // Enable Wait.
```

```
}
// ------------------------------------------------------------
unsigned char Choque(unsigned char x){
  unsigned char y;
        y = 1;
        if (x == F_WR){
                if (BLnOE == 0) y=0;                    // Verifica el "OE" del BUS.
        }
        else{
                if (BLnWE == 0) y=0;                    // Verifica el "WR" del BUS.
        }
        if (y == 1) Sout(Ruta+'0');
        return (y);
}
// ------------------------------------------------------------
void __attribute__((interrupt, no_auto_psv)) _IC1Interrupt(void){
  IFS0bits.IC1IF =      0;        // Clear IC1 Interrupt Flag.
  P_Int = 1;                      // Libera la Interrupción.
        switch(Ruta){
                case KRead0:            // Me escriben Comando.
  Parche:
                        if (Choque(F_RD) == 1) break;
                        MComando.EstCmd = PORTB;
                        ChkSum = MComando.EstCmd;
                        Numero = MComando.CmdNum;
                        CntLoad = 0;
                                switch(MComando.CmdChannel){
                                        case 1:
                                                Channel = 3;
                                                break;
                                        case 2:
                                                Channel = 2;
                                                break;
```

```
                    case 4:
                        Channel = 1;
                        break;
                    case 8:
                        Channel = 0;
                        break;
                    default:
                        Channel = 4;
                     break;
                }
            if (Channel > 3) break;
            Ruta = KRead1;
            Sout('@');
            PrHL(MComando.EstCmd);
    break;
    case KRead1:                        // Carga los Datos.
            Choque(F_RD);
            if (MComando.CmdDE == 0)
            ArrCmdCon[Channel].Buff[CntLoad++] = PORTB;
            else BuffCmd[Channel][CntLoad++] = PORTB;
            ChkSum = ChkSum ^ PORTB;
            if (CntLoad >= Numero) Ruta = KRead2;
    break;
    case KRead2:                                    // Lee el CheckSum.
            Choque(F_RD);
            ChkSum = ChkSum ^ PORTB;
            if (ChkSum == 0) Acknoledge = K_Ack;
            else Acknoledge = K_Nack;
            Ruta = KRead3;
    break;
    case KRead3:                                    // Envía el Acknoledge.
            Choque(F_WR);
            LATB = Acknoledge;
```

```
if (MComando.CmdDE == 0){
        ArrEstCon[Channel].EstInst = ArrCmdCon[Channel].ActDact;
        ArrEstCon[Channel].EstCanal = ArrCmdCon[Channel].ActDact;
        ArrEstCon[Channel].V_NumBlk= ArrCmdCon[Channel].NumBlock;
        ArrEstCon[Channel].OverRun = 0;
        if (ArrCmdCon[Channel].ActDact == 0)
          StBuffer[Channel].BfEnable = False;
        else
          StBuffer[Channel].BfInit = True;
        StBuffer[Channel].BfStat = True;
}
 else
StBuffer[ScanChan].BfComd = True;
         Recibido(0);
         B_TR = 1;                          // BUS en Write.
         TRISB = 0;                         // PIC en Write.
         FreeWait();
         TRISB = 0xffff;                    // PIC en Lectura.
         B_TR = 0;                          // BUS en Read.
         return;
break;
case KWSta0:                                // Envía el Status.
         if (Choque(F_WR) == 1){
                 TRISB          = 0xffff;   // PIC en Lectura.
                 B_TR           = 0;        // BUS en Read.
                 goto Parche;
         }
         LATB = Cabecera;
         Ruta = KWSta1;
         B_TR = 1;                          // BUS en Write.
         TRISB = 0;                         // PIC en Write.
break;
case KWSta1:                                // Envía el Status.
```

```
        if (Choque(F_WR) == 1){
                TRISB           = 0xffff;       // PIC en Lectura.
                B_TR            = 0;            // BUS en Read.
                goto Parche;
        }
        Numero = 6;
        MStatus.CmdNum = Numero;
        MStatus.CmdChannel = MComando.CmdChannel;
        MStatus.TrfDE = 0;          // Status.
        MStatus.CmdDE = 0;          // Concentrador.
        LATB = MStatus.EstCmd;
        ChkSum = MStatus.EstCmd;
        CntLoad = 0;
        Ruta = KWSta2;
        B_TR = 1;                           // BUS en Write.
        TRISB = 0;                          // PIC en Write.
break;
case KWSta2:                                // Envía los Datos.
        Choque(F_WR);
        ChkSum = ChkSum ^ ArrEstCon[Channel].Buff[CntLoad];
        LATB = ArrEstCon[Channel].Buff[CntLoad++];
        if (CntLoad >= Numero){
                Ruta = KWCom1;
        }
break;
case KWDat0:                                // Envía  Cabecera.
        if (Choque(F_WR) == 1){
                TRISB = 0xffff;             // PIC en Lectura.
                B_TR = 0;                   // BUS en Read.
                goto Parche;
        }
        LATB = Cabecera;
        Ruta = KWDat1;
```

```
        B_TR = 1;                       // BUS en Write.
        TRISB = 0;                      // PIC en Write.
break;
case KWDat1:                            // Envía  Comando.
        if (Choque(F_WR) == 1){
                TRISB = 0xffff;         // PIC en Lectura.
                B_TR = 0;               // BUS en Read.
                goto Parche;
        }
        Numero = ( ArrCmdCon [Channel] . DatBlock + 3 ) *
        ArrCmdCon[Channel].NumBlock;
        NumEnviado = Numero;
        ArrCmdCon[Channel].DatRcx = 0;
        MStatus.CmdNum = Numero;
        MStatus.CmdChannel = Decode(Channel);
        MStatus.DaPeSp = 0;         // Datos Periódicos.
        MStatus.TrfDE = 1;          // Datos.
        MStatus.CmdDE = 1;          // Instrumento.
        LATB = MStatus.EstCmd;
        ChkSum = MStatus.EstCmd;
        CntLoad = 0;
        Ruta = KWDat2;
        B_TR = 1;                       // BUS en Write.
        TRISB = 0;                      // PIC en Write.
break;
case KWDat2:                            // Envia Datos.
        Choque(F_WR);
        ChkSum = ChkSum ^ BuffDat[Channel][CntLoad];
        LATB = BuffDat[Channel][CntLoad++];
        if (CntLoad >= Numero){
                Ruta = KWCom1;
                StBuffer[Channel].BfTxfr = False;
        }
```

```
                break;
                case KWCom1:                            // Envia el ChkSum.
                        if (Choque(F_WR) == 1){
                                TRISB           = 0xffff;       // PIC en Lectura.
                                B_TR            = 0;            // BUS en Read.
                                goto Parche1;
                        }
                        LATB = ChkSum;
                        Ruta = KWCom2;
                        FreeWait();
                        LATB = 0x5555;
                        TRISB           = 0xffff;       // PIC en Lectura.
                        B_TR            = 0;            // BUS en Read.
                return;
                break;
                case KWCom2:                            // Recibe el Acknoledge.
  Parche1:
                        Choque(F_RD);
                        RspAck = PORTB;
                        Acknoledge = RspAck;
                        if ((RspAck & 0x7fff) == K_Ack){
                                StopTout();
                                if ((RspAck & 0x8000) == 0) Recibido(0);
                                else Recibido(0xff);
                        } else Sout('e');
                break;
        }
  FreeWait();
}
// -------------------------------------------------------------
void Recibido(unsigned char x){
  Ruta = KRead0;
  Semaforo = x;
```

```
}
// ------------------------------------------------------------
unsigned char Decode(unsigned char x){
  unsigned char temp = 0;
        switch(x){
                case 3:
                        temp = 1;
                break;
                case 2:
                        temp = 2;
                break;
                case 1:
                        temp = 4;
                break;
                case 0:
                        temp = 8;
                break;
        }
  return(temp);
}
// ------------------------------------------------------------
void InitBuff(void){
  unsigned char i;
        for (i=0;i<4;i++){
                StBuffer[i].StatBf = 0;
                StBuffer[i].BfFull = False;
                StBuffer[i].BfEmpty = True;
                StBuffer[i].BfSync = False;
                StBuffer[i].BfError = False;
                StBuffer[i].BfOver = False;
                StBuffer[i].BfInit = False;
                StBuffer[i].BfEnable = False;
                StBuffer[i].BfReset = False;
```

```
                StBuffer[i].BfStat = False;
                StBuffer[i].BfIniB = True;
                StBuffer[i].BfTxfr = False;
                StBuffer[i].BfComd = False;
                Idx_Rx[i] = 0;
        }
  Free485 = True;
}
// ------------------------------------------------------------
void Each2Mseg(void){
  unsigned char i;
        if ((Ruta == KRead0) && Free485){
                for (i=0;i<4;i++){
                        if(++ScanChan > 3) ScanChan = 0;
                        if (StBuffer[ScanChan].BfEnable){
                                StBuffer[ScanChan].BfSync = True;
                                DoCmd();
                                break;
                        }
                        if (StBuffer[ScanChan].BfInit){
                                DoCmd();
                                break;
                        }
                }
        }
}
// ------------------------------------------------------------
void DoCmd(void){
  unsigned char i;
        if (StBuffer[ScanChan].BfComd){
                StBuffer[ScanChan].BfComd = False;
                Free485 = False;
                P_Prb3 = 1;
```

```
            CntLoad++;
            Snd485(ArrEstCon[ScanChan].EstInst,0x80,3 + (2*CntLoad));
            ChkSum = (0x80 << 8) + (CntLoad);
            ISout(CntLoad);
            for (i=0;i<(CntLoad-1);i++){
                  ChkSum = ChkSum ^ BuffCmd[ScanChan][i];
                  WSout(BuffCmd[ScanChan][i]);
            }
            WSout(ChkSum);
            ArrEstCon[ScanChan].W485 = W485_WCmd;
            return;
      }
      if (StBuffer[ScanChan].BfInit){
            StBuffer[ScanChan].BfInit = False;
            Free485 = False;
            P_Prb3 = 1;
            Snd485(ArrEstCon[ScanChan].EstInst,0x40,3+(2*3));
            ChkSum = (0x40 << 8) + (3);
            ISout(3);
            ChkSum = ChkSum ^ (ArrCmdCon[ScanChan].Divisor);
            WSout(ArrCmdCon[ScanChan].Divisor);
            ChkSum = ChkSum ^ (ArrCmdCon[ScanChan].DatBlock);
            WSout(ArrCmdCon[ScanChan].DatBlock);
            WSout(ChkSum);
            ArrEstCon[ScanChan].W485 = W485_WIni;
            return;
      }
      if (StBuffer[ScanChan].BfReset){
            StBuffer[ScanChan].BfReset = False;
            Free485 = False;
            P_Prb3 = 1;
            Snd485(ArrEstCon[ScanChan].EstInst,0x10,2+1);
            ArrEstCon[ScanChan].W485 = W485_WRst;
```

```
                return;
        }
        if (StBuffer[ScanChan].BfSync){
                StBuffer[ScanChan].BfSync = False;
                Free485 = False;
                P_Prb3 = 1;
                Snd485(ArrEstCon[ScanChan].EstInst,0x20,2);
                ArrEstCon[ScanChan].W485 = W485_WPool;
                return;
        }
}
// -------------------------------------------------------------
void TipoRsp(void){
        switch(Dato_Rx & 0xf8){
                case 0x08:              // Recibió señal de Reset.
                case 0x10:              // No hay nada.
                default:
                        EndChann();
                        break;
                case 0x20:              // Bloque de Status.
                        ArrEstCon[ScanChan].W485 = W485_W2Rsp;
                        StBuffer[ScanChan].BfDat = False;
                        Put_RutRd0();
                        break;
                case 0x40:              // Bloque de Datos.
                        ArrEstCon[ScanChan].W485 = W485_W2Rsp;
                        StBuffer[ScanChan].BfDat = True;
                        Put_RutRd0();
                        break;
                case 0x80:              // Bloque de Datos Periódicos.
                        ArrEstCon[ScanChan].W485 = W485_WBRsp;
                        StBuffer[ScanChan].BfDat = True;
                        Put_RutRd0();
```

```
                        break;
        }
}
// ------------------------------------------------------------
void Scan485(void){
        switch(ArrEstCon[ScanChan].W485){
                case W485_Loop:                 // Inicia conversación.
                        break;
                case W485_WPool:                // Espera q llegue el Comando de Pool.
                        if(Wait_Snd()) break;
                        ArrEstCon[ScanChan].W485 = W485_W1Rsp;
                        break;
                case W485_W1Rsp:        // Espera el Primer Byte
                        if(Wait_Rsp()) break;
                        if((St_I485.error) == 0){
                                TipoRsp();
                        } else {
                                Sout('^');
                                EndChann();
                        }
                        break;
                case W485_W2Rsp:        // Espera el Resto del Buffer
                        if(Wait_Rsp()) break;
                        if((St_I485.error) == 0) SetBfFull();
                        EndChann();
                        break;
                case W485_WBRsp:        // Espera el Resto Bloques
                        __builtin_btg((unsigned int *) &C_Prb4);
                        if(Wait_Rsp()) break;
                        if((St_I485.error) == 0){
                        if(--ArrEstCon[ScanChan].V_NumBlk == 0){
                         ArrCmdCon[ScanChan].DatRcx = Idx_Rx[ScanChan];
                        Idx_Rx[ScanChan] = 0;
```

```
                    StBuffer[ScanChan].BfIniB = True;
                     ArrEstCon[ScanChan].V_NumBlk =
                     ArrCmdCon[ScanChan].NumBlock;
                    SetBfFull();
                    }
                    } else Sout('*');
                    EndChann();
                    break;
                case W485_WIni:                 // Espera que se envíe el Init.
                    if(Wait_Snd()) break;
                    StBuffer[ScanChan].BfEnable = True;
                    EndChann();
                    break;
                case W485_WRst:                 // Espera que se envíe el Reset.
                    if(Wait_Snd()) break;
                    StBuffer[ScanChan].BfEnable = False;
                    EndChann();
                    break;
                case W485_WCmd:                 // Espera que se envíe el Comando.
                    if(Wait_Snd()) break;
                    EndChann();
                    break;
        }
}
// ------------------------------------------------------------
void EndChann(void){
  Put_RutRx0();
  Free485 = True;
  P_Prb3 = 0;
  ArrEstCon[ScanChan].W485 = W485_Loop;
}
// ------------------------------------------------------------
void SetBfFull(void){
```

```
  if (StBuffer[ScanChan].BfFull) StBuffer[ScanChan].BfOver = True;
  else StBuffer[ScanChan].BfFull = True;
}
// ------------------------------------------------------------
void Supervisor(void){
  unsigned char i;
        if(Free485 && (Semaforo == 0) && (Ruta == KRead0)){
                for (i=0;i<4;i++){
                        if (StBuffer[i].BfStat){
                                StBuffer[i].BfStat = False;
                                CalcChkSum();
                                ChkCal = VrfChk;
                                Ruta = KWSta0;          // Para que envié el Estado.
                                Channel = i;
                                P_Int   = 0;                    // Interrumpe.
                                break;
                        }
                        if (StBuffer[i].BfFull){
                                StBuffer[i].BfFull = False;
                                if (ArrCmdCon[i].ActDact == 0) break;
                                StBuffer[Channel].BfTxfr = True;
                                Ruta = KWDat0;          // Envia Datos.
                                Channel = i;
                                P_Int   = 0;            // Interrumpe.
                                break;
                        }
                }
        }
}
// ------------------------------------------------------------
void IniMan(unsigned char x,unsigned int nist){
  ArrCmdCon[x].ActDact = nist;
  ArrCmdCon[x].Divisor = S_Divisor;
```

```
  ArrCmdCon[x].DatBlock = S_DatBlock;
  ArrCmdCon[x].NumBlock = S_NumBlock;
  ArrEstCon[x].EstInst = ArrCmdCon[x].ActDact;
  ArrEstCon[x].EstCanal = ArrCmdCon[x].ActDact;
  ArrEstCon[x].V_NumBlk = ArrCmdCon[x].NumBlock;
  ArrEstCon[x].OverRun = 0;
  StBuffer[x].BfInit = True;
}
```

ANEXO B

PROGRAMA INSTRUMENTO

```
/*****************************************************************************
 file: rs232_Instr.c
*****************************************************************************/
#include <p33FJ128GP802.h>
#include "TSICMD.h"
#include "rs232_Instr.h"
#include "Estructuras.h"
#include "Tx_Rx_rs232.h"
#include "Clock.h"
#include "Extern_Variables.h"
#define         baudrate        59          // Para 38400 BRGH a 18.432MHz
#define         P_TICK          _LATA0  //Pin Ticks IRQ cada 2ms
//IRQ cada 1ms porque usamos Change Notification
#define         T_P_TICK     _TRISA0
#define         LED_1           LATA,1
#define         LED1            _LATA1
#define         T_LED1          _TRISA1
#define BytesXSample 6          //2bytes ->Flujo   2bytes ->Temp  2bytes ->Presion
#define NumberSampleCmdDBFTP   50
volatile unsigned char tempdata;
volatile unsigned char FrameDataA[BuffSide];
volatile unsigned char FrameDataB[BuffSide];
volatile unsigned char BufferA_Full=0;  // 0 -> Vacío   ---  1-> LLeno
volatile unsigned char BufferB_Full=0;
void Init_Uar1_Instr() {
  PMD1bits.U1MD = 0;                // Uart1 enable !!!
  _TRISB14 = 0;                     // Salida TX.
  _TRISB15 = 1;                     // Entrada RX.
  U1BRG =     baudrate;             // Set Baudrate
  U1MODE = 0;
  U1MODEbits.UARTEN = 1;            // Uart enable.
  U1MODEbits.RTSMD = 1;             // No flow control mode.
  U1MODEbits.PDSEL0 = 0;            // 8 bits sin paridad
```

```
U1MODEbits.PDSEL1 = 0;
U1MODEbits.BRGH =0 ;              //Low speed
U1STA = 0;
U1STAbits.UTXEN = 1;              // Transmit enable.
U1STAbits.ADDEN = 0;              // Address Detect disable.
IFS0bits.U1TXIF = 0;              // Clear Uart Transmit Interrupt.
IFS0bits.U1RXIF = 0;              // Clear Uart Receiv Interrupt.
IEC0bits.U1TXIE = 1;              // Disable Uart Transmit Interrupt.
IEC0bits.U1RXIE = 1;              // Enable Uart Receiv Interrupt.
IPC2bits.U1RXIP2 = 0;              // Set UART RX interrupt priority
IPC2bits.U1RXIP1 = 1;
IPC2bits.U1RXIP0 = 1;
IPC3bits.U1TXIP2=0;
IPC3bits.U1TXIP1=1;
IPC3bits.U1TXIP0=1;
IFS4bits.U1EIF = 0;               // Clear Uart Error Interrupt.
IEC4bits.U1EIE      = 1;          // Enable Uart Error Interrupt.
IPC16bits.U1EIP2    = 1;          // Set UART Error interrupt priority
IPC16bits.U1EIP1    = 0;
IPC16bits.U1EIP0    = 0;
MyState_Instr=S_Recognition;
MyBufferNext=BufferB;  //Estado siguiente en el buffer B para que empiece en A
Ptr_BufferActual=FrameDataA;
Ptr_BufferActual_Full=&BufferA_Full;
MyChangeBuffer.enable=True;
MyCorr.correlativo=0;
MyTime.time=0;
MyOverFlow.Instrument=0;
MyActivity_485.dataProgramada=False;
MyActivity_485.data=False;
MyActivity_485.Status=False;
T_P_TICK=1;  //Input
P_TICK=0;
```

```
  CNEN1bits.CN2IE=1;
  IEC1bits.CNIE=1;
  IFS1bits.CNIF=0;
  IPC4bits.CNIP2=1;
  IPC4bits.CNIP1=0;
  IPC4bits.CNIP0=1;
  MyState_TX_Instr=S_OCIO;
}
// ---------------------------------------------------------------
void __attribute__ ((interrupt, no_auto_psv)) _U1ErrInterrupt(void) {
  IFS4bits.U1EIF = 0;// Clear Uart Error Interrupt.
  U1STAbits.OERR = 0;
  U1STAbits.FERR = 0;
}
// ---------------------------------------------------------------
void __attribute__ ((interrupt, no_auto_psv)) _U1RXInterrupt(void) {
  unsigned char Dato;
  volatile static unsigned int countdata=0;
  volatile static unsigned int countdataframe=0;
  volatile unsigned int checksum=0;
  volatile unsigned int WData;
  unsigned int i;
  IFS0bits.U1RXIF = 0;
  Dato = U1RXREG & 0xff;
    switch(MyState_Instr)
    {
        case S_Recognition:
        if(Dato!= 0){
            if(Dato=='O')
            MyState_Instr=S_Ascci_K ;
            else{
            MyState_Instr=S_Error;
            Enable_Timer3();
```

```
    }
}
else
MyState_Instr=S_Data_L ;
break;
case  S_Ascci_K:
    if(Dato=='K')
   MyState_Instr=S_Ascci_CR ;
    else{
    MyState_Instr=S_Error;
    Enable_Timer3();
    }
break;
case  S_Ascci_CR:
    if(Dato==0x0D)
   MyState_Instr=S_Ascci_LF ;
    else{
    MyState_Instr=S_Error;
    Enable_Timer3();
    }
break;
case  S_Ascci_LF:
    if(Dato==0x0A)
   MyState_Instr=S_Recognition;
    else{
    MyState_Instr=S_Error;
    Enable_Timer3();
    }
break;
case S_Data_L:
    countdata++;
    if( *Ptr_BufferActual_Full == True ){
      MyOverFlow.Instrument=True;
```

```
      //Acción a tomar
      for(i=0;i<MyLenframe.lenght+6;i++)
      //Data + (1W)Tiempo_H + (1W)Tiempo_L + (1W)Corre
      Ptr_BufferActual[i]=0;
      *Ptr_BufferActual_Full=False;
       MyOverFlow.Instrument=False;
       checksum=0;
    }
   Ptr_BufferActual[countdataframe++]=Dato;
   WData=Dato;
   MyState_Instr=S_Data_H;
break;
case S_Data_H:
    countdata++;
    Ptr_BufferActual[countdataframe++]=Dato;
    WData= WData + (Dato<<8);
    checksum = checksum ^ WData;
    if(countdataframe >= MyLenframe.lenght*2  ){
      Ptr_BufferActual[countdataframe++]=MyTime.time0;
      Ptr_BufferActual[countdataframe++]=MyTime.time1;
      Ptr_BufferActual[countdataframe++]=MyTime.time2;
      Ptr_BufferActual[countdataframe++]=MyTime.time3;
       checksum = checksum ^ MyTime.Wtime_L ;
       checksum = checksum ^ MyTime.Wtime_H ;
       MyCorr.correlativo++;
       Ptr_BufferActual[countdataframe++]=MyCorr.correlativo_L;
       Ptr_BufferActual[countdataframe++]=MyCorr.correlativo_H;
       checksum = checksum ^ MyCorr.correlativo ;
       checksum = checksum ^ (  0x8000 + MyLenframe.lenght );
       Ptr_BufferActual[countdataframe++]=checksum;
       countdataframe=0;
       checksum=0;
       *Ptr_BufferActual_Full=True;
```

```
        Ptr_BufferBefore=Ptr_BufferActual;
        Ptr_BufferBefore_Full=Ptr_BufferActual_Full;
        switch(MyBufferNext){
         case BufferA:
              Ptr_BufferActual=FrameDataA;
              Ptr_BufferActual_Full=&BufferA_Full;
              MyBufferNext=BufferB;
              break;
          case BufferB:
              Ptr_BufferActual=FrameDataB;
              Ptr_BufferActual_Full=&BufferB_Full;
              MyBufferNext=BufferA;
              break;
        }
        __builtin_btg((unsigned int *) &LED_1);
         MyActivity_485.dataProgramada=True;
        // MyChangeBuffer.enable=True;
        // while(MyActivity_485.sendingdata==True){}
     }
     if(countdata < BytesXSample*NumberSampleCmdDBFTP )
     // (6 bytes de (2)Flujo , (2)Temperatura , (2)Presion) * 50
     {
       MyState_Instr=S_Data_L;
     }
     else
    {
      countdata=0;
      MyState_Instr=S_OxFF_L;
    }
break;
case S_OxFF_L:
    if(Dato==0xFF)
     MyState_Instr=S_0xFF_H;
```

```
    else{
     MyState_Instr=S_Error;
     Enable_Timer3();
    }
break;
case S_0xFF_H:
   if(Dato==0xFF){
      if( lock485 != 1){
          MyState_TX_Instr=S_DBFTPXXXX;
          LenghCmdTXInstr=10;
          Ptr_BufferCmdTx=DBFTP0050;
          while (U1STAbits.UTXBF == 1) {}
          // 1 -> Por lo menos un caracter puede ser enviado  ?
          U1TXREG = Ptr_BufferCmdTx[0];
          // __builtin_btg((unsigned int *) &LED_TEST);
      }
      MyState_Instr=S_Recognition;
   }
   else
   {
     MyState_Instr=S_Error;
     Enable_Timer3();
   }
break;
case S_Error:
   if(Dato==0xFF)
   count++;
   if(count>=2){
            if(tempdata==0xFF){
            MyState_Instr=S_Recognition;
            count=0;}
            }
            else
```

```
                count=1;
            }
        tempdata=Dato;
        break;
      }
}
// ---------------------------------------------------------------
void __attribute__ ((interrupt,no_auto_psv)) _CNInterrupt(void) {
  static unsigned char count_two=0;
  IFS1bits.CNIF=0;    // Clear CNIF Interrupt.
  count_two++;
  if( count_two>=2 ){
     MyTime.time++;
     count_two=0;
    }
}
void __attribute__ ((interrupt, no_auto_psv)) _U1TXInterrupt(void) {
  volatile static unsigned char coutSSR=0;
  volatile static unsigned char coutDBFTP=0;
  volatile static unsigned char first=False;
  volatile static unsigned char countS_DBFTPXXXX=0;
  IFS0bits.U1TXIF=0;
    switch(MyState_TX_Instr){
          case S_OPEN:
             if ( coutSSR < (LenghCmdTXInstr-1) )
              U1TXREG = Ptr_BufferCmdTx[++coutSSR];
             else
             {
              U1TXREG = DBFTP0050[coutDBFTP++];
                 if(coutDBFTP==10 )
                 {
                    coutDBFTP=0;
                    if(first==True)
```

```
                    {
                     coutSSR=0;
                      coutDBFTP=0;
                     first=False;
                     MyState_TX_Instr=S_OCIO;
                    }
                    else
                    first=True;
                  }
               }
          break;
          case S_DBFTPXXXX:
             if (countS_DBFTPXXXX<LenghCmdTXInstr-1 )
              U1TXREG = Ptr_BufferCmdTx[++countS_DBFTPXXXX];
             else{
              countS_DBFTPXXXX=0;
              MyState_TX_Instr=S_OCIO;
             }
          break;
          case S_SSRXXXX:
          break;
          case S_SUS:
          break;
          case S_CLOSE:
          break;
          case S_OCIO:
          break;
    }
}
void  ClearComunicationInstr(){
  MyActivity_485.dataProgramada=False;
  MyActivity_485.data=False;
  MyActivity_485.Status=False;
```

```
  MyBufferNext=BufferB;  //Estado siguiente en el buffer B para que empieze en A
  Ptr_BufferActual=FrameDataA;
  Ptr_BufferActual_Full=&BufferA_Full;
  BufferA_Full=0;
  BufferB_Full=0;
}
/*****************************************************************************/
```

File:Slave485.c

```
*****************************************************************************/
#include <p33FJ128GP802.h>
#include "Estructuras.h"
#include "Slave485.h"
#include "Tx_Rx_rs485.h"
#include "Clock.h"
#include "rs232_Instr.h"
#include "Extern_Variables.h"
#define         baudrate        9               // Para  APROX 1MBIT BRGH a 18.432MHz
#define         CONN_16_5       LATB,4
#define         CONN16_5        _LATB4
#define         T_CONN16_5      _TRISB4
#define         CONN_16_4       LATB,3
#define         CONN16_4        _LATB3
#define         T_CONN16_4      _TRISB3
#define         LED_2           LATA,4
#define         LED2            _LATA4
#define         T_LED2          _TRISA4
#define         LED_1           LATA,1
#define         LED1            _LATA1
#define         T_LED1          _TRISA1
#define         T_EnableTX_485  _TRISB12
#define EnableTX_485    _LATB12   //habilita TX bus 485 ;  LOW -> Alta impedancia
#define CmdPI_I 8
#define CmdC_I  4
```

```
#define Read_DS 2
#define Reset   1
unsigned char Perif;
unsigned char TPerif;
unsigned int NumberWordTX;
void conversion(unsigned int value,volatile unsigned char * ptr);
// -------------------------------------------------------------
void Init_slave_uart2(unsigned char x) {
  Perif = x;
  PMD1bits.U2MD = 0;                // Uart2 enable !!!
  _TRISB8 = 0;                              // Salida TX.  UART2
  _TRISB9 = 1;                              // Entrada RX. UART2
  T_EnableTX_485 = 0;               // Salida Enable TX 485   // LED3
  T_CONN16_4=0;
  CONN16_4=1;
  EnableTX_485 = 0;           // RS485 en High Z.  //LED3 ON
  U2BRG =     baudrate;             // Set Baudrate
  U2MODE = 0;
  U2MODEbits.UARTEN = 1;        // Uart enable.
  U2MODEbits.RTSMD = 1;// No flow control mode.
  U2MODEbits.PDSEL0 = 1;// 9 bits no parity
  U2MODEbits.PDSEL1 = 1;
  U2MODEbits.BRGH = 1;
  U2STA = 0;
  U2STAbits.UTXEN = 1;              // Transmit enable.
  U2STAbits.ADDEN = 1;              // Address Detect enable.
  IFS1bits.U2TXIF = 0;              // Clear Uart Transmit Interrupt.
  IFS1bits.U2RXIF = 0;              // Clear Uart Receiv Interrupt.
  IEC1bits.U2TXIE = 1;              // Disable Uart Transmit Interrupt.
  IEC1bits.U2RXIE = 1;              // Enable Uart Receiv Interrupt.
  IPC7bits.U2RXIP2 = 1;             // Set UART RX interrupt priority
  IPC7bits.U2RXIP1 = 0;
  IPC7bits.U2RXIP0 = 0;
```

```
  IPC7bits.U2TXIP2 = 1;                // Set UART TX interrupt priority
  IPC7bits.U2TXIP1 = 0;
  IPC7bits.U2TXIP0 = 0;
  IFS4bits.U2EIF = 0;                  // Clear Uart Error Interrupt.
  IEC4bits.U2EIE      = 1;             // Enable Uart Error Interrupt.
  IPC16bits.U2EIP2    = 1;             // Set UART Error interrupt priority
  IPC16bits.U2EIP1    = 0;
  IPC16bits.U2EIP0    = 0;
  MyStateRX = Num_Instr;
  MyActivity_485.sendingdata=False;
  MyStateTX=OCIO;
  NumberWordTX=0;
  lock485=0;
}
// -------------------------------------------------------------
void __attribute__ ((interrupt, no_auto_psv)) _U2ErrInterrupt(void) {
  IFS4bits.U2EIF = 0;// Clear Uart Error Interrupt.
  U2STAbits.OERR = 0;
  U2STAbits.FERR = 0;
}
// -------------------------------------------------------------
void __attribute__ ((interrupt, no_auto_psv)) _U2RXInterrupt(void) {
  volatile static unsigned int countdata=0;
  volatile unsigned int mytimer;
  volatile unsigned char errorbit9;
  unsigned char Dato;
   unsigned int Recieve;
   IFS1bits.U2RXIF = 0;
   Recieve=U2RXREG & 0x1ff;
   Dato = Recieve & 0xff;
              if( MyStateRX != Num_Instr ){
                if( (Recieve&0x100) == 0x100 )
                   errorbit9=True ;
```

```
            else
              errorbit9=False ;
            }
            else{
             if( (Recieve&0x100) == 0x100 )
              errorbit9=False ;
             else
              errorbit9=True ;
            }
if( errorbit9 == False )
{
            switch(MyStateRX) {
             case Num_Instr:
                        TPerif = Dato;
                        if (Perif == (TPerif & 0xfe)){
                          U2STAbits.ADDEN = 0;      // Address Detect disable.
                          CONN16_5 = 1;               // Pin 5 del conector de 16 pines
                          MyStateRX=Cmd_H;
                           Enable_Timer2(0x2B3);      //Enable timer2 a 1.2ms
                        }
              break;
              case Cmd_L:
                        MyCmdR.comando_L = Dato;                                    //
                       //
                        switch(MyCmdR.info){
                             case CmdPI_I:
                             MyStateRX=DataPI_I ;
                             break;
                             case CmdC_I:
                             MyStateRX=FrecSampleFrame_L;
                             break;
                        }
              break;
```

```
case Cmd_H:
  MyCmdR.comando_H = Dato;
  switch(MyCmdR.info){
      case CmdPI_I:
      case CmdC_I:
       MyStateRX=Cmd_L;
       break;
      case Read_DS:
       IEC0bits.T2IE = 0;// Deshabilita la interrupción.
       T2CONbits.TON = 0;// Deshabilita el contador.
       U2STAbits.ADDEN = 1;    //Habilita detección dirección
       EnableTX_485 = 1;   // RS485 enable uart2 TX
       IEC1bits.U2RXIE = 0;              //Disable interrupciones U2
       if(MyActivity_485.dataProgramada==True  && lock485==0 ){
        //Existe un Frame para transmitir
         MyCmdW.numberwords=  (MyLenframe.lenght +3)  & 0x1ff ;
         MyCmdW.indefinido= 0 ;
         MyCmdW.info_5=0x10;
         MyActivity_485.dataProgramada=False;
        CONN16_5 = 0;
         MyStateTX=DataProgramada;
         if( MyCmdW.numberwords != 0)
         mytimer= 26*MyCmdW.numberwords +25;
         Enable_Timer2(mytimer);  //Enable timer2
         ESout(MyCmdW.comando_H);
         ESout(MyCmdW.comando_L);
       }
       else if(MyActivity_485.data==True && lock485==0  ){
                MyCmdW.numberwords= 0x20 & 0x1ff ;   //words
                MyCmdW.indefinido= 0 ;
                MyCmdW.info_5=0x8;
                MyActivity_485.data=False;
                MyStateTX=Data;
```

```
        if( MyCmdW.numberwords != 0)
        mytimer= 26*MyCmdW.numberwords +25;
        Enable_Timer2(mytimer);  //Enable timer2
        ESout(MyCmdW.comando_H);
        ESout(MyCmdW.comando_L);
    }else if(MyActivity_485.Status==True && lock485==0 ){
     if( getProcess_Transducer(&myAvParamTrans)==True  )
     {
      MyCmdW.numberwords=(getNumberWordsToSendTransducer
                                    (&myAvParamTrans) )& 0x1ff     ;
      setPtrStatusData(getArrayInfo_Transducer(&myAvParamTrans)
      clearProcess_Transducer(&myAvParamTrans);
     } else if( getProcess_Respiration(&myAvParamResp)==True  )
         {
        //Resp
         MyCmdW.numberwords=(getNumberWordsToSendRespiration
                                      (&myAvParamResp) ) & 0x1ff;
         setPtrStatusData(getArrayInfo_Respiration(&myAvParamResp));
         //
        clearProcess_Respiration(&myAvParamResp);
         }
     MyCmdW.indefinido= 0 ;
     MyCmdW.info_5=0x4;
     MyStateTX=Status;
  if( MyCmdW.numberwords != 0)
         mytimer= 26*MyCmdW.numberwords +25;
         Enable_Timer2(mytimer);  //Enable timer2
         ESout(MyCmdW.comando_H);
         ESout(MyCmdW.comando_L);
  }else{
              MyCmdW.numberwords= 0 ;
              MyCmdW.indefinido= 0 ;
              MyCmdW.info_5=0x2;
```

```
                MyStateTX=NoBloques;
                Enable_Timer2(0x33);  //Enable timer2
                ESout(MyCmdW.comando_H);
        }
        break;
        case Reset: //Condición de Reinicio
        IEC0bits.T2IE = 0;            // Deshabilita la interrupción.
        T2CONbits.TON = 0;                  // Deshabilita el contador.
        MyCmdW.numberwords= 0 ;
        MyCmdW.indefinido= 0 ;
        MyCmdW.info_5=0x1;
        U2STAbits.ADDEN = 1;              //Habilita detección dirección
        EnableTX_485 = 1;             // RS485 enable uart2 TX
        IEC1bits.U2RXIE = 0;              //Disable interrupciones U2
        MyStateTX=AckReset;
        ESout(MyCmdW.comando_H);
        break;
        default:
        break;
    }  //end switch MyCmdR.info
break;
case FrecSampleFrame_L:
MyFrecSampleframe.frecuencia_L=Dato;
MyStateRX=FrecSampleFrame_H;
break;
case FrecSampleFrame_H:
MyFrecSampleframe.frecuencia_H=Dato;
MyStateRX=Leghframe_L;
break;
case Leghframe_L:
MyLenframe.lenght_L=Dato;
MyStateRX=Leghframe_H;
break;
```

```
 case Leghframe_H:
 MyLenframe.lenght_H=Dato;
MyStateRX=Checksum_Conc_Instr_L;
 break;
 case Checksum_Conc_Instr_L:
 MyChecksum_C_I.checksum_L=Dato;
 MyStateRX=Checksum_Conc_Instr_H;
 break;
 case Checksum_Conc_Instr_H:
 MyChecksum_C_I.checksum_H=Dato;
 IEC0bits.T2IE = 0;          // Desabilita la interrupcion.
 T2CONbits.TON = 0;      // Desabilita el contador.
 if( MyLenframe.lenght==0  &&  MyFrecSampleframe.frecuencia==0 ){
      CONN16_4=0;
      lock485=1;
      break;
 }
// CONN16_4=0;
  ClearComunicationInstr();
  lock485=0;   //Habilitar el 485
  MyStateRX=Num_Instr;
  MyState_TX_Instr=S_OPEN;
  conversion(MyFrecSampleframe.frecuencia*2,&SSRXXXX[3]);
  LenghCmdTXInstr=8;
  Ptr_BufferCmdTx=SSRXXXX;
  while (U1STAbits.UTXBF == 1) {}
  U1TXREG = Ptr_BufferCmdTx[0];
  U2STAbits.ADDEN = 1;                    // Address Detect enable.
  break;
  case DataPI_I:
  DataInstr[countdata]=Dato;
  countdata++;
  if( countdata >= (2*MyCmdR.numberwords +2) ){
```

```
                U2STAbits.ADDEN = 1;                // Address Detect disable.
                MyStateRX=Num_Instr;
                IEC0bits.T2IE = 0;                  // Deshabilita la interrupción.
               T2CONbits.TON = 0;                   // Deshabilita el contador.
               countdata=0;
              }
              break;
             }
  }
}
void ESout(unsigned char x) {
  while (U2STAbits.UTXBF == 1) {}
  U2TXREG =x & 0xff;
}
void conversion(unsigned int value,volatile unsigned char * ptr)
{
  unsigned int residuo;
  unsigned int i=0;
  if(value<10){
      residuo= value%10;
      ptr[0]=0x30;
      ptr[1]=0x30;
      ptr[2]=0x30;
      ptr[3]= residuo+0x30;
  } else if ( value >= 10 && value < 100 ){
              ptr[0]=0x30;
              ptr[1]=0x30;
              for(i=0;i<2;i++){
              residuo= value%10;
              value=(unsigned int)(value/10);
              ptr[3-i]=residuo+0x30;
              }
        }else if (value >= 100 && value < 1000){
```

```
            ptr[0]=0x30;
            for(i=0;i<3;i++){
            residuo= value%10;
            value=(unsigned int)(value/10);
            ptr[3-i]=residuo+0x30;
            }
        } else if (value >= 1000 && value < 10000){
                for(i=0;i<4;i++){
                residuo= value%10;
                value=(unsigned int)(value/10);
                ptr[3-i]=residuo+0x30;
                }
            } else return;
}
```

ANEXO C
PROGRAMA COMUNICACIONES

```
/*****************************************************************************
```

File: main.cpp

```
 *****************************************************************************/
#include <errno.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/time.h>
#include "fsbus.h"
#include <iostream>
using namespace std;
#define MAXNUMCMDINSTR 100
int fd, ints = 0; /* filedescriptor and irq signal handler count */
int ret;
unsigned int num_mod;
int i=0;
int countcmdinstr=0;
fsbus_irq_t irq;
sigset_t mysigset, old_sigset;
void (*old_hdl)(int);
key_t Clave;
int Id_Memoria;
void *Memoria = NULL;
#define readCmd_O_Error
myshreg->cmd_O_error
#define writeCmdCon(value,position,canal) myshreg>ArrCmdCon##canal[position] =
value
#define writelowerCmdCon(value,position,canal)
```

```
myshreg->ArrCmdCon##canal[position] =
((value) | ((myshreg->ArrCmdCon##canal[position])&0xff00))
#define writeupperCmdCon(value,position,canal)
 myshreg->ArrCmdCon##canal[position] =
((value<<8) | ((myshreg->ArrCmdCon##canal[position])&0xff))
#define readCmdCon(position,canal)
 myshreg->ArrCmdCon##canal[position]
#define readlowerCmdCon(position,canal)
 ((myshreg->ArrCmdCon##canal[position])&0xff)
#define readupperCmdCon(position,canal)
 ((myshreg->ArrCmdCon##canal[position])>>8)
#define writeStatus(value,position,canal)
 myshreg->status##canal[position] = value
#define writelowerStatus(value,position,canal)
myshreg->status##canal[position]=((value) | ((myshreg>status##canal[position])&0xff00))
#define writeupperStatus(value,position,canal)
myshreg->status##canal[position]=((value<<8)|((myshreg>status##canal[position])&0xff))
#define readStatus(position,canal)
myshreg->status##canal[position]
#define readlowerStatus(position,canal)
 ((myshreg->status##canal[position] )& 0xff)
#define readupperStatus(position,canal)
((myshreg->status##canal[position])>>8)
#define writef_CmdConct(value)
 myshreg->f_CmdtoConct= value;
#define readf_CmdtoConct(bitpos)
((myshreg->f_CmdtoConct >> bitpos) & 0x1)
#define writef_CmdtoConct(value,bitpos) if(value!=0){ myshreg->f_CmdtoConct |=
(1<<bitpos) ;} else {myshreg->f_CmdtoConct= ((myshreg->f_CmdtoConct) &
(~(1<<bitpos))) ;}
#define writef_CmdInstr(value)
myshreg->f_CmdtoInstr= value;
#define readf_CmdtoInstr(bitpos)
```

```
 ((myshreg->f_CmdtoInstr >> bitpos) & 0x1)
#define writef_CmdtoInstr(value,bitpos)
if(value!=0){ myshreg->f_CmdtoInstr |= (1<<bitpos) ;} else {myshreg->f_CmdtoInstr=
((myshreg->f_CmdtoInstr) & (~(1<<bitpos))) ;}
#define readPid(canal)
myshreg->pid##canal
#define writePid(value,canal)
myshreg->pid##canal=value
#define write_lock_CmdConc(value)
myshreg->lock_CmdtoConc= value;
#define readf_lock_CmdtoConc(bitpos)
((myshreg->lock_CmdtoConc >> bitpos) & 0x1)
#define write_lock_CmdtoConc(value,bitpos)
if(value!=0){ myshreg->lock_CmdtoConc |= (1<<bitpos) ;} else {myshreg-
>lock_CmdtoConc &= (~(1<<bitpos)) ;}
#define readOpt_Chann(canal)
myshreg->opcionchannel##canal
#define writeOpt_Chann(value,canal)
 myshreg->opcionchannel##canal=value
#define readActive_Chann(canal)
 myshreg->activechannel##canal
#define writeActive_Chann(value,canal)
myshreg->activechannel##canal=value
volatile union CMD_R {
        volatile unsigned short int value;
        struct {
           volatile unsigned short int lenght:9;
           volatile unsigned short int ndef:1;       //no definido
           volatile unsigned short int num_canal:4; //numero canal
           volatile unsigned short int IC_DE:2;     //dos bits mas siginificativos
        };
}mycmd;
struct reg{
```

```
volatile unsigned short int cmd_O_error;
volatile unsigned short int data1[MAXNUMBDATA];  //1034 short words
volatile unsigned short int data2[MAXNUMBDATA];
volatile unsigned short int data3[MAXNUMBDATA];
volatile unsigned short int data4[MAXNUMBDATA];
volatile unsigned short int status1[MAXNUMBSTATES]; //11 short words
volatile unsigned short int status2[MAXNUMBSTATES];
volatile unsigned short int status3[MAXNUMBSTATES];
volatile unsigned short int status4[MAXNUMBSTATES];
volatile unsigned short int ArrCmdCon1[MAXNUMCMDCON]; //8 short words
volatile unsigned short int ArrCmdCon2[MAXNUMCMDCON];
volatile unsigned short int ArrCmdCon3[MAXNUMCMDCON];
volatile unsigned short int ArrCmdCon4[MAXNUMCMDCON];
volatile unsigned short int ArrCmdInstr1[MAXNUMCMDINSTR]; //100 short words
volatile unsigned short int ArrCmdInstr2[MAXNUMCMDINSTR];
volatile unsigned short int ArrCmdInstr3[MAXNUMCMDINSTR];
volatile unsigned short int ArrCmdInstr4[MAXNUMCMDINSTR];
volatile unsigned short int f_CmdtoConct;
volatile unsigned short int f_CmdtoInstr;
volatile unsigned short int pid1;
volatile unsigned short int pid2;
volatile unsigned short int pid3;
volatile unsigned short int pid4;
volatile unsigned short int Ack_Nack1;
volatile unsigned short int Ack_Nack2;
volatile unsigned short int Ack_Nack3;
volatile unsigned short int Ack_Nack4;
volatile unsigned short int lock_CmdtoConc;
volatile unsigned short int opcionchannel1;
volatile unsigned short int opcionchannel2;
volatile unsigned short int opcionchannel3;
volatile unsigned short int opcionchannel4;
volatile unsigned short int activechannel1;
```

```
volatile unsigned short int activechannel2;
volatile unsigned short int activechannel3;
volatile unsigned short int activechannel4;
volatile unsigned short int lenghtArrCmdInstr1;
volatile unsigned short int lenghtArrCmdInstr2;
volatile unsigned short int lenghtArrCmdInstr3;
volatile unsigned short int lenghtArrCmdInstr4;
struct debugger {
   volatile unsigned short int array[10];
}mydebugger;
volatile unsigned int short enable_debugger;
}*myshreg;
bool activechannel1=false;
bool activechannel2=false;
bool activechannel3=false;
bool activechannel4=false;
bool repiteCmd_chann1=false;
bool repiteCmd_chann2=false;
bool repiteCmd_chann3=false;
bool repiteCmd_chann4=false;
unsigned char array[500];
volatile bool enableread=false;
volatile bool lockhandler=true;
bool spurea = false;
void initialization();
bool sendCmd();
bool sendCmdInstr();
itimerval mytimer;
void myhandler_timer(int typesignal);
void myhandler_software();
bool ready_read=false;
bool unavez1ercmd=false;
void irq_sigint_handler(int signum)
```

```
{
   myhandler_software();
}
void program_exit (int type);
int main(int argc, char **argv)
{
 int on = 1, edge;
 unsigned int mymodewrite;
 unsigned int mybusywrite;
 //memoria compartida
 Clave = ftok ("/bin/ls", 33);
 if (Clave == -1)
 {
    exit(0);
  }
Id_Memoria = shmget (Clave, sizeof(unsigned char)*12288, 0777 | IPC_CREAT);
if (Id_Memoria == -1)
{
   exit (0);
}
Memoria = shmat (Id_Memoria, (unsigned char *)0, 0);
if (Memoria == NULL)
{
 exit (0);
}
//memoria compartida
myshreg=(struct reg *)Memoria;
//Initialization share memory
initialization();
sleep(2);
//configuracion fsbus
old_hdl = signal(SIGIO, irq_sigint_handler); /* set new and save old handler */
edge=1; //Falling
```

```
/* open device */
fd = open("/dev/fsbus", O_RDWR);
if (fd < 0) {
printf("Can't open /dev/fsbus\n");
signal(SIGIO, old_hdl); /* restore */
exit(1);
}
/* descriptor configuration */
fcntl(fd, F_SETOWN, getpid()); /* set process id that will receive SIGIO */
ioctl(fd, FIOASYNC, &on); /* I/O by signals */
ioctl(fd, FIONBIO, &on); /* nonblocking I/O */
/* set the trigger edge */
if (edge)
   irq.type = 1;
 else
   irq.type = 0;
irq.mask = 1;  //Habilitar la IRQ
/* get IRQ ressource */
if (ioctl(fd, FSBUS_SETIRQ, &irq)) {
printf("Can't activate IRQ\n");
signal(SIGIO, old_hdl);
exit(1);
}
sigemptyset(&mysigset); /* obtain masked signals */
sigprocmask(0, NULL, &mysigset);
sigdelset(&mysigset, SIGIO); /* unmask SIGIO */
sigprocmask(SIG_SETMASK, &mysigset, &old_sigset); /* apply new mask */
//configuracion fsbus
//reconfigurando seÃ±al por default para ctrl-C
if (signal (SIGINT, program_exit) == SIG_ERR)
{
  perror ("No se puede cambiar signal");
}
```

```
//reconfigurando seÃ±al por default para ctrl-C
//configuracion timer//
mytimer.it_interval.tv_sec=10;
mytimer.it_interval.tv_usec=0;
mytimer.it_value.tv_sec=10;
mytimer.it_value.tv_usec=0;
signal(SIGALRM,myhandler_timer);
setitimer(ITIMER_REAL,&mytimer,0);
for (;;)
{
if((readActive_Chann(1)==0)&&(readActive_Chann(2)==0)&&(readActive_Chann(3)==0
) && (readActive_Chann(4)== 0) )
{
        mymodewrite=0;  //Modo normal escribir al bus
                ret = ioctl(fd, FSBUS_MODE_W, &mymodewrite);
                if (ret) {
                        close(fd);
                        exit(1);
                }
                if(  sendCmd()==true ){
                        if ( (readCmdCon(7,1)) != ACK  ){   //Canal 1
                          writef_CmdtoConct(1,0);
                          writeActive_Chann(0,1);
                        }
                        else
                        {
                           mymodewrite=1;  //Modo Write memory
                           ret = ioctl(fd, FSBUS_MODE_W, &mymodewrite);
                           if (ret) {
                            close(fd);
                           exit(1);
                           }
                        }
```

```
                    }
}
else
{
                ret = ioctl(fd, FSBUS_BUSY_W, &mybusywrite);
                if (ret) {
                close(fd);
                exit(1);
                }
                if(mybusywrite==0)  { //libre
                  //Send Comand
                  if(sendCmd() == false)   sendCmdInstr();
                }
}
if(myshreg->enable_debugger != 0){
              if (ioctl(fd, FSBUS_DEBUGGER,  &(myshreg->mydebugger)  ))
               {
                    signal(SIGIO, old_hdl);
                    exit(1);
               }
               myshreg->enable_debugger=0;
}
usleep(50000);
}//for(;;)
 return 0;
}
void initialization()
{
  for(int i=0;i<MAXNUMBSTATES;i++)
  writeStatus(0,i,1);
  for(int i=0;i<MAXNUMBSTATES;i++)
  writeStatus(0,i,2);
  for(int i=0;i<MAXNUMBSTATES;i++)
```

```
  writeStatus(0,i,3);
  for(int i=0;i<MAXNUMBSTATES;i++)
  writeStatus(0,i,4);
  for(int i=0;i<MAXNUMCMDCON;i++)
  writeCmdCon(0,i,1);
  for(int i=0;i<MAXNUMCMDCON;i++)
  writeCmdCon(0,i,2);
  for(int i=0;i<MAXNUMCMDCON;i++)
  writeCmdCon(0,i,3);
  for(int i=0;i<MAXNUMCMDCON;i++)
  writeCmdCon(0,i,4);
  writef_CmdConct(0);
  writef_CmdInstr(0);
  writeOpt_Chann(0,1);
  writeOpt_Chann(0,2);
  writeOpt_Chann(0,3);
  writeOpt_Chann(0,4);
  writePid(-1,1);
  writePid(-1,2);
  writePid(-1,3);
  writePid(-1,4);
  writeActive_Chann(0,1);
  writeActive_Chann(0,2);
  writeActive_Chann(0,3);
  writeActive_Chann(0,4);
  for(int i=0;i<10;i++)
   myshreg->mydebugger.array[i]=0;
   myshreg->enable_debugger=0;
}
void myhandler_timer(int typesignal){
}
void myhandler_software()
{
```

```
fsbus_irq_t irq;
int retread;
volatile unsigned short int mycmd_O_error;
if(readActive_Chann(1) == 0)
    if(readActive_Chann(2) == 0)
        if(readActive_Chann(3) == 0)
            if(readActive_Chann(4) == 0){
             //cout << "Spurea No hay canales" << endl;
             spurea=true;
            }
if(spurea==false){
   retread = read(fd, (void *)(&myshreg->cmd_O_error), 1);
      if (retread < 0) {
         close(fd);
         exit(1);
      }
      mycmd.value=readCmd_O_Error;
      switch( mycmd.value ){
      case E_NACK:
      break;
      case E_CON_L:
      break;
      case E_CON_D:
      break;
      case E_CHANNEL:
      break;
      case E_LENGHT_CH1:
      break;
      case E_LENGHT_CH2:
      break;
      case E_LENGHT_CH3:
      break;
      case E_LENGHT_CH4:
```

```
break;
case E_SINCRO :  //
break;
default:
        switch(mycmd.num_canal){
            case 8:  //canal 1
                switch(mycmd.IC_DE){
                    case 0: //concentrador + transferencia status
                    break;
                    case 1: //concentrador + transferencia de datos
                    break;
                    case 2: //instrumento + transferecia status
                    break;
                    case 3: //instrumento + transferencia datos
                     kill(readPid(1),SIGUSR1);
                    break;
                    default:
                    break;
                }
            break;
            case 4: //canal 2
                switch(mycmd.IC_DE){
                    case 0: //concentrador + transferencia status
                    break;
                    case 1: //concentrador + transferencia de datos
                    break;
                    case 2: //instrumento + transferecia status
                    break;
                    case 3: //instrumento + transferencia datos
                     kill(readPid(2),SIGUSR1);
                    break;
                    default:
                    break;
```

```
        }
    break;
    case 2: //canal 3
        switch(mycmd.IC_DE){
            case 0: //concentrador + transferencia status
            break;
            case 1: //concentrador + transferencia de datos
            break;
            case 2: //instrumento + transferecia status
            break;
            case 3: //instrumento + transferencia datos
             kill(readPid(3),SIGUSR1);
            break;
            default:
            break;
        }
    break;
    case 1: //canal 4
        switch(mycmd.IC_DE){
            case 0: //concentrador + transferencia status
            break;
            case 1: //concentrador + transferencia de datos
            break;
            case 2: //instrumento + transferecia status
            break;
            case 3: //instrumento + transferencia datos
             kill(readPid(4),SIGUSR1);
            break;
            default:
            break;
        }
    break;
    default:
```

```
                    //cout << "error canal" << endl;
                    break;
                }
      break;
      }
 }else
  spurea=false;
}
bool sendCmd()
{
   bool sendCmdNow=false;
   if(readf_CmdtoConct(i)==1){
     switch(i)
     {
        case 0: //canal 1
        num_mod=8; //1000
        ret = ioctl(fd, FSBUS_CC_W, &num_mod);
            if (ret) {
                  cout << "error ioctl num_mod="<<  num_mod << endl;
                  close(fd);
                  exit(1);
            }
        // activechannel1=true;
         writeActive_Chann(1,1);
         ret = write(fd,( void *)(myshreg->ArrCmdCon1), MAXNUMCMDCON-2);
            if (ret < 0) {
                  cout << "error write "<<endl;
                  close(fd);
                  exit(1);
            }
        cout << "canal 1" << endl;
        break;
        case 2://canal 2
```

```
 num_mod=4; //0100
 ret = ioctl(fd, FSBUS_CC_W, &num_mod);
 if (ret) {
            cout << "error ioctl num_mod="<<  num_mod << endl;
            close(fd);
            exit(1);
 }
//activechannel2=true;
 writeActive_Chann(1,2);
 ret = write(fd,( void *)(myshreg->ArrCmdCon2), MAXNUMCMDCON-2);
    if (ret < 0) {
            cout << "error write "<<endl;
            close(fd);
            exit(1);
    }
 cout << "canal 2" << endl;
  break;
case 4://canal 3
num_mod=2; //0010
 ret = ioctl(fd, FSBUS_CC_W, &num_mod);
    if (ret) {
            cout << "error ioctl num_mod="<<  num_mod << endl;
            close(fd);
            exit(1);
    }
// activechannel3=true;
  writeActive_Chann(1,3);
 ret = write(fd,( void *)(myshreg->ArrCmdCon3), MAXNUMCMDCON-2);
    if (ret < 0) {
            cout << "error write "<<endl;
            close(fd);
            exit(1);
    }
```

```
        cout << "canal 3" << endl;
        break;
        case 6://canal 4
         num_mod=1; //0001
         ret = ioctl(fd, FSBUS_CC_W, &num_mod);
            if (ret) {
                  cout << "error ioctl num_mod="<<  num_mod << endl;
                  close(fd);
                  exit(1);
            }
        //  activechannel4=true;
         writeActive_Chann(1,4);
         ret = write(fd,( void *)(myshreg->ArrCmdCon4), MAXNUMCMDCON-2);
            if (ret < 0) {
                  cout << "error write "<<endl;
                  close(fd);
                  exit(1);
            }
       break;
       default:
       break;
     } //switch
   writef_CmdtoConct(0,i);
   sendCmdNow=true;
   } //if
i+=2;
if(i>7) i=0;
return sendCmdNow;
}
bool sendCmdInstr()
{
   bool sendCmdNow=false;
   if(readf_CmdtoInstr(countcmdinstr)==1){
```

```
switch(countcmdinstr)
{
   case 0: //canal 1
   num_mod=8; //1000
   ret = ioctl(fd, FSBUS_IC_W, &num_mod);
       if (ret) {
              cout << "error ioctl num_mod="<<  num_mod << endl;
              close(fd);
              exit(1);
       }
    ret = write(fd,( void *)(myshreg->ArrCmdInstr1),
                                  myshreg->lenghtArrCmdInstr1);
       if (ret < 0) {
              cout << "error write "<<endl;
              close(fd);
              exit(1);
       }
  break;
  case 2://canal 2
  num_mod=4; //0100
  ret = ioctl(fd, FSBUS_IC_W, &num_mod);
       if (ret) {
              cout << "error ioctl num_mod="<<  num_mod << endl;
              close(fd);
              exit(1);
       }
  ret = write(fd,( void *)(myshreg->ArrCmdInstr2),
                               myshreg->lenghtArrCmdInstr2); //
       if (ret < 0) {
              cout << "error write "<<endl;
              close(fd);
              exit(1);
       }
```

```
break;
case 4://canal 3
 num_mod=2; //0010
 ret = ioctl(fd, FSBUS_IC_W, &num_mod);
     if (ret) {
          cout << "error ioctl num_mod="<<  num_mod << endl;
          close(fd);
          exit(1);
     }
ret = write(fd,( void *)(myshreg->ArrCmdInstr3),
                         myshreg->lenghtArrCmdInstr3); //
     if (ret < 0) {
          cout << "error write "<<endl;
          close(fd);
          exit(1);
     }
break;
case 6://canal 4
num_mod=1; //0001
ret = ioctl(fd, FSBUS_IC_W, &num_mod);
     if (ret) {
          cout << "error ioctl num_mod="<<  num_mod << endl;
          close(fd);
          exit(1);
     }
ret = write(fd,( void *)(myshreg->ArrCmdInstr4),
                         myshreg->lenghtArrCmdInstr4); //
     if (ret < 0) {
          cout << "error write "<<endl;
          close(fd);
          exit(1);
     }
break;
```

```
        default:
        break;
      } //switch
   writef_CmdtoInstr(0,countcmdinstr);
   sendCmdNow=true;
   } //if
countcmdinstr+=2;
if(countcmdinstr>7) countcmdinstr=0;
return sendCmdNow;
}
void program_exit (int type)
{
          irq.mask = 0x00;
          if (ioctl(fd, FSBUS_SETIRQ, &irq)) {
                    printf("Warning: Can't deactivate IRQ\n");
          }
          signal(SIGIO, old_hdl);
          close(fd);
          shmdt ((char *)Memoria);
          shmctl (Id_Memoria, IPC_RMID, (struct shmid_ds *)NULL);
          /* Se pone controlador por defecto para ctrl-c */
          signal (SIGINT, SIG_DFL);
          sleep(1);
          exit(EXIT_SUCCESS);
}
```

ANEXO D
PROGRAMA GENERADOR BDSQLITE

```
/*****************************************************************************
  File: main.cpp
*****************************************************************************/
#include <QtCore>
#include <QtXml>
#include <QDebug>
#include <QtSql>
#include <QFileInfo>
int N_Page=0;
int N_Position=-1;
QString PathDirProcedure="-";
bool BuildingProcedure=false;
enum ObjectTypes
{TypeLabel=2300,TypeCheckBox,TypeGroupBox=2302,TypeLineEdit,TypeComboBox,T
ypeHBoxLayout,TypeVBoxLayout, TypeGridLayout, TypeSpacerItem, TypeRadioButton,
TypePlainTextEdit, TypeTableImpl, TypeEditValueIntr, TypeCPushButton };
void ListElement( QDomElement myElement )
{
  int i;
  QString myTagName=myElement.tagName();
  if(myTagName==QString("widget"))
  {
      QString myAttribute=myElement.attribute(QString("class"));
      if( myAttribute==QString("QLabel") )
      {
         int i;
         N_Position++;
         //Variables table TObject
         QString  IdWindows="-";
         QString  IdObject="-";
         QString  Type="-";
         QString  HasInfo="no";
         QString  Name="-";
```

```
QString  Parent="-";
QString  LayoutParent="-";
QString  Strecht="-";
QString  fromRow="-";
QString  toRow="-";
QString  fromCol="-";
QString  toCol="-";
QString  HPolicy="-";
QString  VPolicy="-";
QString  MaxSize="-";
QString  MinSize="-";
//Variables table TLabel
QString  Foreground="rgb(0,0,0)";
QString  Background="inherited";
QString  FontFamily="smoothtimes_440_75.qpf";
QString  FontSize="8";
QString  PixmapImage="-";
QString  Text="-";
QDomNodeList itemsproperty= myElement.childNodes();
for(i=0;i<itemsproperty.count();i++)
{
   QDomNode myproperty= itemsproperty.at(i);
   if(!myproperty.isElement())return;
   else
   {
      QDomElement elemproperty = myproperty.toElement();
      if(elemproperty.tagName()!=QString("property"))return;
      else
      {
        QString nameproperty= elemproperty.attribute(QString("name"),
        QString("no valid"));
        if(nameproperty==QString("no valid")) return;
        else
```

```
{
    if( nameproperty  == QString("pixmap"))
    {
       QDomElement  tagpixmap=
       elemproperty.firstChildElement
       (QString("pixmap"));
       QString mylabelpixmap= tagpixmap.text();
       QStringList list1 =
       mylabelpixmap.split("/");
       mylabelpixmap=list1[list1.count()-1];
       PixmapImage=mylabelpixmap;
    }
    if( nameproperty  == QString("text"))
    {
       QDomElement tagstring=
       elemproperty.firstChildElement
       (QString("string"));
        QString mytextlabel= tagstring.text();
        Text=mytextlabel;
    }
    if( nameproperty  == QString("styleSheet"))
    {
       QDomElement tagstring=
       elemproperty.firstChildElement(QString("string"));
       QString mytextstyle= tagstring.text();
       QStringList list1 = mytextstyle.split(";");
       for(int i=0;i<list1.count()-1;i++)
       {
          list1[i]= list1[i].simplified();
          if(list1[i].startsWith (QString("color")))
          {
          Foreground=list1[i].section(": ",1,1);
          }
```

```
                    if(list1[i].startsWith
                    (QString("background-color")))
                    {
                    Background=list1[i].section(": ",1,1);
                    }
                    if(list1[i].startsWith
                    (QString("font")))
                    {
                    FontSize=list1[i].section(" ",1,1);
                    FontSize=FontSize.section("p",0,0);
                    }
                }
            }
          }
        }
    }
}
Name=myElement.attribute( QString("name"),QString("label without name") );
QDomNode nodeparent=  myElement.parentNode();
QDomElement elementparent;
//looking for Parent
for(;;){
    elementparent=nodeparent.toElement();
    if(elementparent.tagName()==QString("widget"))
    {
     QString nameclass=elementparent.attribute(     QString("class"),
     QString("class without name") );
       if(nameclass==QString("QWidget"))
       Parent="this";
       else
       {
        QString myParent=elementparent.attribute( QString("name"),
        QString("widget parent without name") );
```

```
        QSqlQuery query;
        query.exec("select IdObject from TObject where Name=\"" +
        myParent+ "\""+" and IdWindow="+
        QString::number(N_Page));
          if(query.next())
          {
             myParent= QString::number(N_Page);
             myParent+=",";
             myParent+=QString::number(query.value(0).toInt());
             Parent=myParent;
          }
        }
       break;
     }
 nodeparent=  nodeparent.parentNode();
}
//looking layout parent
nodeparent=  myElement.parentNode();
for(;;){
    elementparent=nodeparent.toElement();
    if(elementparent.tagName()==QString("item") )
    {
       bool islayoutgrid =false;
       QDomElement itemparent= elementparent;
       nodeparent=  nodeparent.parentNode();
       elementparent=nodeparent.toElement();
       LayoutParent=elementparent.attribute(QString("name"),
       QString("widget  parent without name") );
       QSqlQuery query;
    }
    query.exec("select IdObject from TObject where
    Name=\"" + LayoutParent+ "\""+" and IdWindow="+
    QString::number(N_Page));
```

```
if(query.next())
{
   LayoutParent=QString::number(N_Page);
   LayoutParent+=",";
   LayoutParent+=QString::number(query.value(0).toInt());
}
if( itemparent.attribute( QString("row"),QString("-") ) != QString("-") )
{
   fromRow=itemparent.attribute( QString("row") );
   islayoutgrid=true;
}
if( itemparent.attribute( QString("column"),QString("-") ) != QString("-") )
{
fromCol=itemparent.attribute( QString("column"));
}
if( itemparent.attribute( QString("rowspan"),QString("-") ) != QString("-"))
{
 toRow=itemparent.attribute( QString("rowspan"));
 toRow=QString::number
 (fromRow.toInt()+toRow.toInt()-1);
}
else
{
   if(islayoutgrid)
   {
      toRow=fromRow;
   }
}
if( itemparent.attribute(  QString("colspan"), QString("-") ) != QString("-"))
{
   toCol=itemparent.attribute( QString("colspan"));
   toCol=QString::number(fromCol.toInt()+toCol.toInt()-1);
}
```

```
                else
                {
                    if(islayoutgrid)
                    {
                        toCol=fromCol;
                    }
                }
                break;
            }
          nodeparent=  nodeparent.parentNode();
        }
IdWindows=QString::number(N_Page);
IdObject=QString::number(N_Position);
Type=QString::number(TypeLabel);
QSqlQuery query;
QString stringquery="insert into TObject values(";
stringquery+=IdWindows;
stringquery+=",";
stringquery+=IdObject;
stringquery+=",";
stringquery+="\'";
stringquery+=Type;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=HasInfo;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=Name;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
```

```
stringquery+=Parent;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=LayoutParent;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=Strecht;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=fromRow;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=toRow;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=fromCol;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=toCol;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=HPolicy;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=VPolicy;
```

```
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=MaxSize;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=MinSize;
stringquery+="\'";
stringquery+=")";
query.exec(stringquery);
stringquery="insert into TLabel values(";
stringquery+=IdWindows;
stringquery+=",";
stringquery+=IdObject;
stringquery+=",";
stringquery+="\'";
stringquery+=Foreground;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=Background;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=FontFamily;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=FontSize;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
```

```
stringquery+=PixmapImage;
stringquery+="\'";
stringquery+=",";
stringquery+="\'";
stringquery+=Text;
stringquery+="\'";
stringquery+=")";
query.exec(stringquery);
return;
}
int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
db.setDatabaseName("BdSqlite");
if (!db.open()) {
        qDebug()<< "Error open BdSqlite";
        exit(1);
}
 QSqlQuery query;
 query.exec("create table TObject (IdWindow INTEGER , IdObject INTEGER, Type TEXT, HasInfo TEXT, Name TEXT, Parent TEXT, LayoutParent TEXT, Stretch TEXT , fromRow TEXT , toRow TEXT ,fromCol TEXT, toCol TEXT ,HPolicy TEXT, VPolicy TEXT , MaxSize TEXT, MinSize TEXT , PRIMARY KEY "
 "(IdWindow,IdObject) )");
query.exec("create table TLabel (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT,Background TEXT, FontFamily TEXT, FontSize TEXT, PixmapImage TEXT, Text TEXT, "PRIMARY KEY (IdWindow,IdObject) )");
query.exec("create table TGroupBox (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT,Title TEXT  ,PRIMARY KEY (IdWindow,IdObject) ) ");
query.exec("create table TLineEdit (IdWindow INTEGER , IdObject INTEGER, Text TEXT,PRIMARY KEY (IdWindow,IdObject) )");
```

```
query.exec("create table TComboBox (IdWindow INTEGER , IdObject INTEGER, Item1 TEXT, Item2 TEXT,Item3 TEXT, Item4 TEXT, Item5 TEXT, Item6 TEXT, Item7 TEXT, Item8 TEXT, Item9 TEXT,Item10 TEXT  ,PRIMARY KEY (IdWindow,IdObject) )");
query.exec("create table TSpacer (IdWindow INTEGER , IdObject INTEGER, Orientation TEXT,Width TEXT  , Heigth TEXT, PRIMARY KEY (IdWindow,IdObject) ) ");
query.exec("create table TCheckBox (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT, Background TEXT, FontFamily TEXT, FontSize TEXT, Enable TEXT, Text TEXT, PRIMARY KEY (IdWindow,IdObject) )");
query.exec("create table TRadioButton (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT, Background TEXT, FontFamily TEXT, FontSize TEXT, Checked TEXT, Text TEXT, PRIMARY KEY (IdWindow,IdObject) )");
query.exec("create table TMultiLineEdit (IdWindow INTEGER , IdObject INTEGER, Foreground TEXT, Background TEXT, FontFamily TEXT, FontSize TEXT, ReadOnly TEXT, Text TEXT, PRIMARY KEY (IdWindow,IdObject) )");
query.exec("create table TTableImpl (IdWindow INTEGER , IdObject INTEGER, Num_Rows TEXT, Num_Cols TEXT, Proced TEXT, Proportion TEXT, UnidMinCharacters TEXT, ""PRIMARY KEY (IdWindow,IdObject) )");
query.exec("create table THeadersTableImpl (IdWindow INTEGER , IdObject INTEGER, Position TEXT, Label TEXT ,PRIMARY KEY (IdWindow,IdObject,Position) )");
query.exec("create table TCellTableImpl (IdWindow INTEGER , IdObject INTEGER, CellRow TEXT, CellColumn TEXT ,value TEXT , PRIMARY KEY (IdWindow,IdObject,CellRow,CellColumn) )");
query.exec("create table TEditValueInstr (IdWindow INTEGER , IdObject INTEGER, Text TEXT, InstrUnit TEXT, PRIMARY KEY (IdWindow,IdObject) )");
query.exec("create table TCPushButton (IdWindow INTEGER , IdObject INTEGER, isAProcess TEXT, nameProcess TEXT, isAProcedure TEXT, N_PageProcedure TEXT, PRIMARY KEY (IdWindow,IdObject) )");
QDomDocument document;
QString myPath="D:/ejemplos_qt4.7/ReadDom/";
QString mypage;
QFileInfo fi;
QFile file;
int countPageOTM=1;
```

```
int countPageProcedure=1;
qDebug() << "Begin scan xml";
    for(/*int i=1;;i++*/;;)
    {
       if(!BuildingProcedure) //Paginas OTM
       {
         mypage=myPath;
         mypage+="page";
         mypage+=QString::number(countPageOTM);
         mypage+=".xml";
         countPageOTM++;
       }
       else                //Procedimiento
       {
         mypage=PathDirProcedure;
         mypage+="/proc";
         mypage+=QString::number(countPageProcedure);
         mypage+=".xml";
         countPageProcedure++;
       }
       fi.setFile(mypage);
       if(!fi.exists())
       {
          if(!BuildingProcedure)
          break;
          else
          {
             BuildingProcedure=false;
             countPageProcedure=1;
             continue;
           }
       }
       qDebug()<<"Page"<<mypage;
```

```
        file.setFileName( mypage);
        if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
        {
             qDebug() << "Filed open file" ;
              return -1;
         }
        else
        {
          if(!document.setContent(&file))
          {
               qDebug()<< "Failed to load document";
               return -1;
           }
          file.close();
        }
        QDomElement root= document.firstChildElement();
        QDomNodeList myWidgetContainer=
        root.elementsByTagName(QString("widget"));
        QDomNode rootElement =myWidgetContainer.at(0);
        // qDebug()<< myWidgetContainer.count();
         ListElement(rootElement.toElement());
         N_Page++;
         N_Position=-1;
    }
qDebug() << "End scan xml";
db.close();
return a.exec();
}
```

ANEXO E
INTERFAZ PARALELA

```
/*****************************************************************************
 File:Driver.c
*****************************************************************************/
static ssize_t fsbus_read(struct file *filp, char *buff, size_t count, loff_t *offp)
{
int i;
volatile unsigned long flags = 0;
spin_lock_irqsave(&fspario_lock, flags);
FS_ParInput();
if(F_E_SINCR == 1){
F_E_SINCR=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_SINCRO,(unsigned short int *)buff);
return 1;
}
if( F_E_CHANNEL == 1){
F_E_CHANNEL=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_CHANNEL,(unsigned short int *)buff);
return 1;
}
if( F_E_LENGH_CH1 == 1){
F_E_LENGH_CH1=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_LENGHT_CH1,(unsigned short int *)buff);
return 1;
}
if( F_E_LENGH_CH2 == 1){
F_E_LENGH_CH2=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_LENGHT_CH2,(unsigned short int *)buff);
return 1;
}
```

```
if( F_E_LENGH_CH3 == 1){
F_E_LENGH_CH3=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_LENGHT_CH3,(unsigned short int *)buff);
return 1;
}
if( F_E_LENGH_CH4 == 1){
F_E_LENGH_CH4=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_LENGHT_CH4,(unsigned short int *)buff);
return 1;
}
if( F_E_LENGH_CH0 == 1){
F_E_LENGH_CH0=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_LENGHT_CH0,(unsigned short int *)buff);
return 1;
}
if( F_E_STATUSCONC_CH0 == 1){
F_E_STATUSCONC_CH0=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_STATUSCONC_CH0,(unsigned short int *)buff);
return 1;
}
if( F_E_CON_L == 1){
F_E_CON_L=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
put_user( E_CON_L,(unsigned short int *)buff);
return 1;
}
if( F_E_CON_D == 1){
F_E_CON_D=0;
//s3c2410_gpio_setpin(S3C2410_GPB(5), 1);  //Apagado
```

```
put_user( E_CON_D,(unsigned short int *)buff);
return 1;
}
if( BufferRecp[mycmdR.lenght+1] == NACK){
printk("mycmdR.value=%x\n",mycmdR.value);
for(i=0;i< mycmdR.lenght+2;i++)   //Datos + chksum + ACK/NACK
printk("BufferRecp[%d]=%x\n",i,BufferRecp[i]);
put_user( E_NACK,(unsigned short int *)buff);
return mycmdR.lenght;
printk("-----------------------------------\n");
}else{
switch(mycmdR.I_C){
case 0:  //Concentrador
if(mycmdR.D_S==0){ //Status
put_user( mycmdR.value,(unsigned short int *)buff);
switch(mycmdR.num_canal)
{
case 8: //Primer canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 2) , (void *)BufferRecp ,
mycmdR.lenght*2+4);              //datastatus+ cksum + Ack/Nack
break;
case 4: //segundo canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 2*MAXNUMBSTATES +2) ,
(void *)BufferRecp , mycmdR.lenght*2+4);// datastatus +cksum + Ack/Nack
break;
case 2: //tercer canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 4*MAXNUMBSTATES +2) ,
(void *)BufferRecp , mycmdR.lenght*2+4);//datastatus +cksum + Ack/Nack
break;
case 1: //cuarto canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 6*MAXNUMBSTATES +2) ,
(void *)BufferRecp , mycmdR.lenght*2+4);//datastatus +cksum + Ack/Nack
break;
```

```
case 0xe: //Canal Debugger Concentrador
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES +
8*MAXNUMCMDCON+ 8*MAXNUMCMDINSTR + 8*MAXNUMSTATUSINSTR
+8*MAXNUMDATANOPROG +2) , \
(void *)BufferRecp , mycmdR.lenght*2+4); //Datos + chksum + Ack/Nack
break;
default:
//printk("num de channel=%d ???\n",mycmdR.num_canal);
break;
}
}
else{
count_CON_D++;
printk("ERROR CONCENTRADOR + DATA NUMBER=%d\n",count_CON_D++);
put_user( E_CON_D,(unsigned short int *)buff);
return mycmdR.lenght;
// return E_CON_D;
}
break;
case 1://Instrumento
put_user( mycmdR.value,(unsigned short int *)buff); //Comando
if(mycmdR.D_S==1){ //Data
if(mycmdR.dat_datProgram==0){ //Data Programada
switch(mycmdR.num_canal)
{
case 8: //Primer canal
copy_to_user( (void *)(buff+2) , (void *)BufferRecp , mycmdR.lenght*2+4); //Datos +
chksum + Ack/Nack
break;
case 4: //segundo canal
copy_to_user( (void *)(buff + 2*MAXNUMBDATA +2) , (void *)BufferRecp ,
mycmdR.lenght*2+4); //Datos + chksum + Ack/Nack
break;
```

```
case 2: //tercer canal
copy_to_user( (void *)(buff + 4*MAXNUMBDATA +2) , (void *)BufferRecp , mycmdR.lenght*2+4); //Datos + chksum + Ack/Nack
break;
case 1: //cuarto canal
copy_to_user( (void *)(buff + 6*MAXNUMBDATA +2) , (void *)BufferRecp , mycmdR.lenght*2+4); //Datos +chksum + Ack/Nack
break;
 case 0: //Canal 0 caso especial
//Caso especial
break;
default:
printk("numero channel Data Prog =%d  ???\n",mycmdR.num_canal);
break;
}
}
else
{ //Data No programada
switch(mycmdR.num_canal)
{
case 8: //Primer canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES + 8*MAXNUMCMDCON+ 8*MAXNUMCMDINSTR + 8*MAXNUMSTATUSINSTR +2) , (void *)BufferRecp , mycmdR.lenght*2+4); //Datos + chksum + Ack/Nack
break;
case 4: //segundo canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES + 8*MAXNUMCMDCON+ 8*MAXNUMCMDINSTR + 8*MAXNUMSTATUSINSTR +2*MAXNUMDATANOPROG+2) ,  (void *)BufferRecp , mycmdR.lenght*2+4); //Datos //+ chksum + Ack/Nack
break;
case 2: //tercer canal
```

```
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES + 8*MAXNUMCMDCON+ 8*MAXNUMCMDINSTR + 8*MAXNUMSTATUSINSTR +4*MAXNUMDATANOPROG+2) , (void *)BufferRecp , mycmdR.lenght*2+4); //Datos //+ chksum + Ack/Nack
break;
case 1: //cuarto canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES + 8*MAXNUMCMDCON+ 8*MAXNUMCMDINSTR + 8*MAXNUMSTATUSINSTR +6*MAXNUMDATANOPROG +2) , (void *)BufferRecp , mycmdR.lenght*2+4); //Datos //+ chksum + Ack/Nack
break;
default:
printk("numero channel DataNoProg =%d ???\n",mycmdR.num_canal);
break;
}
}
}
else { //Status instrumento
switch(mycmdR.num_canal)
{
case 8: //Primer canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES + 8*MAXNUMCMDCON+ 8*MAXNUMCMDINSTR +2) , (void *)BufferRecp , mycmdR.lenght*2+4); //Datos + chksum + Ack/Nack
break;
case 4: //segundo canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES + 8*MAXNUMCMDCON+ 8*MAXNUMCMDINSTR+ 2*MAXNUMSTATUSINSTR +2) , (void *)BufferRecp , mycmdR.lenght*2+4); //Datos + chksum + Ack/Nack
break;
case 2: //tercer canal
copy_to_user( (void *)( buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES + 8* MAXNUMCMDCON+ 8*MAXNUMCMDINSTR +4*MAXNUMSTATUSINSTR+2)
```

```
,  (void *)BufferRecp , mycmdR.lenght*2+4); //Datos + chksum + Ack/Nack
break;
case 1: //cuarto canal
copy_to_user( (void *)(buff+ 8*MAXNUMBDATA + 8*MAXNUMBSTATES +
8*MAXNUMCMDCON+ 8*MAXNUMCMDINSTR +6*MAXNUMSTATUSINSTR +2)
, (void *)BufferRecp , mycmdR.lenght*2+4); //Datos + chksum + Ack/Nack
break;
default:
printk("numero channel DataNoProg =%d  ???\n",mycmdR.num_canal);
break;
}
}
break;
default:
printk("error cmd startus read\n");
break;
} //switch(mycmdR.I_C)
}// BufferRecp[mycmdR.lenght+2] == NACK
//printk("salio\n");
*offp = 0;                      /* we have no file position */
//return cmd;                         /* always 1 byte read */
spin_unlock_irqrestore(&fspario_lock, flags);
return mycmdR.lenght;  //Numero de words leidos
}
unsigned short int FS_ParInput(void)
{
//volatile unsigned long flags = 0;
int i;
unsigned short int chksum=0;
m_irq.count[0]=0;
sincronismo= *(volatile unsigned short int *)fs_par_addr ;
if(sincronismo!=SINCRO){
error_sincro++;
```

```
printk("Error sincro\n");
F_E_SINCR=1;
goto scape;
}
mycmdR.value = *(volatile unsigned short int *)fs_par_addr ;
//Errores Codificacion canal
if( mycmdR.num_canal != 1 &&  mycmdR.num_canal != 2 && mycmdR.num_canal != 4
&&  mycmdR.num_canal != 8  &&  mycmdR.num_canal != 0  && mycmdR.num_canal
!= 0xe){
printk("E_CHANN= %x \n", mycmdR.num_canal);
error_Channel++;
F_E_CHANNEL=1;
goto scape;
}
//Comprobacion de Errores Concentrador
if(mycmdR.I_C==0){
if( mycmdR.D_S == 1){ //Data
error_conce_data++;
printk("E_CON_D\n");
F_E_CON_D=1;
goto scape;
}
else
{  //Status
if(mycmdR.num_canal!=0xe){
if(mycmdR.num_canal!=0x0){ //Asegurarnos ingresen canales programdos 1,2,3,4
if(mycmdR.lenght != 6){
error_lenght_conce++;
printk("E_CON_L\n");
F_E_CON_L=1;
goto scape;
}
}
```

```
else
{
printk("E_STATUSCONC_CH0\n");
F_E_STATUSCONC_CH0=1;
goto scape;
}
}
}
}
//Errores en longitud de canal programado
if(mycmdR.I_C==1){
if( mycmdR.D_S == 1){
if(mycmdR.dat_datProgram==0)
{
switch( mycmdR.num_canal){
case 8:  //Canal 1
if(lenghDataProgCh1!=mycmdR.lenght){
error_lenght_CH1++;
F_E_LENGH_CH1=1;
printk("E_LENG_CH1 =%x\n", mycmdR.lenght);
goto scape;
}
break;
case 4:  //Canal 2
if(lenghDataProgCh2!=mycmdR.lenght){
error_lenght_CH2++;
F_E_LENGH_CH2=1;
printk("E_LENG_CH2 =%x\n" , mycmdR.lenght);
goto scape;
}
break;
case 2:  //Canal 3
if(lenghDataProgCh3!=mycmdR.lenght){
```

```
error_lenght_CH3++;
F_E_LENGH_CH3=1;
printk("E_LENG_CH3 =%x\n" , mycmdR.lenght);
goto scape;
}
break;
case 1:  //Canal 4
if(lenghDataProgCh4!=mycmdR.lenght){
error_lenght_CH4++;
F_E_LENGH_CH4=1;
printk("E_LENG_CH4 =%x\n", mycmdR.lenght);
goto scape;
}
break;
case 0:  //Canal 0 Especial
if(mycmdR.lenght!=0){
F_E_LENGH_CH0=1;
printk("E_LENG_CH0 =%x\n", mycmdR.lenght);
goto scape;
}
break;
} //Switch
}
}
}
//Errores en longitud de canal programado
chksum = chksum ^ mycmdR.value;
for(i=0;i< mycmdR.lenght+1;i++)   //Datos +chksum
{
BufferRecp[i] = *(volatile unsigned short int *)fs_par_addr;
//rmb();
// mb();
chksum = chksum ^ BufferRecp[i];
```

```
m_irq.count[0]++;
}
rmb();
mb();
if(chksum==0)
{
if(isEnableSendCmd==true){
BufferRecp[mycmdR.lenght+1]=ACK | 0x8000;
}
else{
BufferRecp[mycmdR.lenght+1]=ACK;
}
*(volatile unsigned short int *)fs_par_addr = BufferRecp[mycmdR.lenght+1];
wmb(); // write this value now before you do anything else!
mb(); // memory barrier
if( (isEnableSendCmd==true)  )
{
printk("SEND LOADED COMAND\n\n\n");
*(volatile unsigned short int *)fs_par_addr = mycmdW.value;
wmb(); /* write this value now before you do anything else! */
mb(); /* memory barrier */
for(i=0;i<mycmdW.lenght+1;i++)
{
*(volatile unsigned short int *)fs_par_addr = BufferSend[i];
wmb(); /* write this value now before you do anything else! */
mb(); /* memory barrier */
}
udelay(1);
BufferSend[mycmdW.lenght+1] = *(volatile unsigned short int *)fs_par_addr;
rmb();
mb();
if(BufferSend[mycmdW.lenght+1] == ACK){
busywrite=0;   //write desocupado
```

```
isEnableSendCmd=false;
if( mycmdW.I_C==0  &&  mycmdW.D_C==0  ){
switch( mycmdW.num_canal){
case 8:  //Canal 1
lenghDataProgCh1= (BufferSend[2]+3)*BufferSend[3];
break;
case 4:  //Canal 2
lenghDataProgCh2= (BufferSend[2]+3)*BufferSend[3];
break;
case 2:  //Canal 3
lenghDataProgCh3= (BufferSend[2]+3)*BufferSend[3];
break;
case 1:  //Canal 4
lenghDataProgCh4= (BufferSend[2]+3)*BufferSend[3];
break;
}
}
}else{
printk("ERROR SEND LOADED COMAND\n");
error_nack_con_mem++;
}
}
}
else
{
*(volatile unsigned short int *)fs_par_addr = NACK;
wmb(); // write this value now before you do anything else!
mb(); // memory barrier
BufferRecp[mycmdR.lenght+1]=NACK;
//chksum=0;
//count_error++;
error_nacks_read++;
}
```

```
scape:
//Debugger
if(control.array[2] == 0){    //Deshabilitado solo comando
if( control.array[1] != 0){  //Datos y comandos
printk("--------------------\n");
printk("mycmdR.value=%x\n",mycmdR.value);
for(i=0;i< mycmdR.lenght+1;i++)
printk("BufferRecp[%d]=%x\n",i,BufferRecp[i]);
control.array[1]--;
}
}else{
printk("-----------------\n");
printk("mycmdR.value=%x\n",mycmdR.value);
control.array[2]--;
}
if( control.array[0] != 0 ){  //Lista de Errores
printk("Lista Errores\n");
printk("Error Canal No Exclusivo =%x \n", error_Channel);
printk("Error Numero Datos Programados en Ch1 =%x\n", error_lenght_CH1);
printk("Error Numero Datos Programados en Ch2 =%x\n", error_lenght_CH2);
printk("Error Numero Datos Programados en Ch3 =%x\n", error_lenght_CH3);
printk("Error Numero Datos Programados en Ch4 =%x\n", error_lenght_CH4);
printk("Longitud status concentrador diferente 6 =%x\n",error_lenght_conce);
printk("Error envio de Datos al concentrador =%x\n" , error_conce_data);
printk("Contador Nack read =%x\n", error_nacks_read);
printk("Contador Nack Programando directo CH1 =%x\n", error_nack_prog_Ch1);
printk("Contador Nack Concentrador en Memoria =%x\n", error_nack_con_mem);
printk("Error Sincro =%x", error_sincro);
//printk("Error Correlativo=%x\n",error_correlativo);
printk("\n\n\n");
control.array[0]--;
}
return  0;
```

```
}
static ssize_t fsbus_write(struct file *filp, const char *buff, size_t count, loff_t *offp)
{
unsigned short int chksum=0;
int m,i;
copy_from_user( (void *)BufferSend , (void *)buff ,count*2); //Solo datos
if(writeModeMemory==1)
{
mycmdW.lenght=count;
printk("LOAD COMAND MEMORY\n");
printk("mycmd=%x\n",mycmdW.value);
for( m=0;m<count;m++)
printk("BufferSend[%d]=%x\n",m,BufferSend[m]);
chksum = chksum ^ mycmdW.value;
for(i=0;i<count;i++)
chksum = chksum ^ BufferSend[i];
BufferSend[count]=chksum;
printk("cheksum=%x\n",chksum);
 busywrite=1;
 isEnableSendCmd=true;
return 1;
}
// fsbus_output(count, b);
FS_ParOutput(count);
busywrite=0;
isEnableSendCmd=false;
put_user( BufferSend[count+1],(unsigned short int *)(buff+count*2+2));
*offp = 0;
return 1;
}
```

ANEXO F

PRIMERA PÁGINA OTM

/**

Archivo XML

**/

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Widget</class>
 <widget class="QWidget" name="Widget">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>350</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="windowTitle">
   <string>Widget</string>
  </property>
  <property name="styleSheet">
   <string notr="true"/>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout">
   <property name="margin">
    <number>0</number>
   </property>
   <item>
    <layout class="QGridLayout" name="gridLayout">
```

```
<property name="margin">
 <number>11</number>
</property>
<item row="0" column="0">
 <widget class="QLabel" name="label_3">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="text">
   <string/>
  </property>
  <property name="pixmap">
   <pixmap>essalud.png</pixmap>
  </property>
 </widget>
</item>
<item row="0" column="3">
 <widget class="QLabel" name="label">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="font">
   <font>
    <pointsize>8</pointsize>
   </font>
  </property>
  <property name="text">
```

```
   <string># OTM</string>
  </property>
 </widget>
</item>
<item row="0" column="4" colspan="2">
 <widget class="QLineEdit" name="lineEdit">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Maximum" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="maximumSize">
   <size>
    <width>95</width>
    <height>16777215</height>
   </size>
  </property>
  <property name="text">
   <string>0109 - 2011</string>
  </property>
  <property name="cursorPosition">
   <number>11</number>
  </property>
  <property name="readOnly">
   <bool>true</bool>
  </property>
 </widget>
</item>
<item row="1" column="0" colspan="2">
 <widget class="QLabel" name="label_2">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
```

```
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="font">
   <font>
    <pointsize>8</pointsize>
   </font>
  </property>
  <property name="text">
   <string>GERENCIA OFERTA FLEXIBLE</string>
  </property>
 </widget>
</item>
<item row="1" column="3" colspan="2">
 <widget class="QLabel" name="label_4">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="text">
   <string>Fecha de emision</string>
  </property>
 </widget>
</item>
<item row="1" column="5">
 <widget class="QLineEdit" name="lineEdit_2">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Maximum" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
```

```
  </sizepolicy>
 </property>
 <property name="maximumSize">
  <size>
   <width>70</width>
   <height>16777215</height>
  </size>
 </property>
 <property name="text">
  <string>03/12/11</string>
 </property>
 <property name="cursorPosition">
  <number>8</number>
 </property>
 <property name="readOnly">
  <bool>true</bool>
 </property>
</widget>
</item>
<item row="1" column="2">
 <spacer name="horizontalSpacer_5">
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
  <property name="sizeHint" stdset="0">
   <size>
    <width>40</width>
    <height>20</height>
   </size>
  </property>
 </spacer>
</item>
<item row="0" column="1" colspan="2">
```

```
 <spacer name="horizontalSpacer">
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
  <property name="sizeHint" stdset="0">
   <size>
    <width>168</width>
    <height>29</height>
   </size>
  </property>
 </spacer>
</item>
<item row="2" column="0" colspan="6">
 <layout class="QHBoxLayout" name="horizontalLayout">
  <property name="leftMargin">
   <number>0</number>
  </property>
  <item>
   <spacer name="horizontalSpacer_3">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>13</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QLabel" name="label_5">
    <property name="sizePolicy">
```

```
      <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
       <horstretch>0</horstretch>
       <verstretch>0</verstretch>
      </sizepolicy>
     </property>
     <property name="styleSheet">
      <string notr="true">color: rgb(255, 0, 0);
       font: 14pt &amp;quot;MS Shell Dlg 2&amp;quot;;</string>
     </property>
     <property name="text">
      <string>ORDEN DE TRABAJO DE MANTENIMIENTO</string>
     </property>
    </widget>
   </item>
   <item>
    <spacer name="horizontalSpacer_4">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="sizeHint" stdset="0">
      <size>
       <width>13</width>
       <height>17</height>
      </size>
     </property>
    </spacer>
   </item>
  </layout>
 </item>
</layout>
</item>
<item>
 <widget class="QGroupBox" name="groupBox">
```

```
<property name="styleSheet">
 <string notr="true">color: rgb(0, 0, 255);</string>
</property>
<property name="title">
 <string>I. Datos Generales</string>
</property>
<property name="flat">
 <bool>false</bool>
</property>
<property name="checkable">
 <bool>false</bool>
</property>
<layout class="QGridLayout" name="gridLayout_3">
 <item row="0" column="0">
  <widget class="QLabel" name="label_6">
   <property name="styleSheet">
    <string notr="true">color: rgb(0, 0, 0);</string>
   </property>
   <property name="text">
    <string>1. SERVICIO</string>
   </property>
  </widget>
 </item>
 <item row="0" column="1" colspan="2">
  <widget class="QLineEdit" name="lineEdit_3">
   <property name="styleSheet">
    <string notr="true">color: rgb(0, 0, 0);</string>
   </property>
   <property name="text">
    <string>GOF - CEPRIT</string>
   </property>
   <property name="cursorPosition">
    <number>12</number>
```

```
  </property>
  <property name="readOnly">
   <bool>true</bool>
  </property>
 </widget>
</item>
<item row="1" column="0">
 <widget class="QLabel" name="label_7">
  <property name="font">
   <font>
    <family>MS Serif</family>
   </font>
  </property>
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>2. UBICACION</string>
  </property>
 </widget>
</item>
<item row="1" column="1" colspan="2">
 <widget class="QComboBox" name="comboBox">
  <property name="whatsThis">
   <string extracomment="Combo_Ubicacion"/>
  </property>
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <item>
   <property name="text">
    <string>07-E-P-03-302</string>
   </property>
```

```
  </item>
 </widget>
</item>
<item row="2" column="0">
 <widget class="QLabel" name="label_8">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>3. TELEFONO</string>
  </property>
 </widget>
</item>
<item row="2" column="1">
 <widget class="QLineEdit" name="lineEdit_4">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Maximum" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="maximumSize">
   <size>
    <width>100</width>
    <height>16777215</height>
   </size>
  </property>
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>4490592</string>
  </property>
```

```
    <property name="cursorPosition">
     <number>7</number>
    </property>
   </widget>
  </item>
  <item row="2" column="2">
   <spacer name="horizontalSpacer_2">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
 </layout>
</widget>
</item>
<item>
 <widget class="QGroupBox" name="groupBox_2">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 255);</string>
  </property>
  <property name="title">
   <string>II. Datos del Equipo</string>
  </property>
  <layout class="QGridLayout" name="gridLayout_2">
   <item row="0" column="0">
    <widget class="QLabel" name="label_9">
     <property name="styleSheet">
```

```
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>4. NOMBRE</string>
  </property>
 </widget>
</item>
<item row="0" column="1">
 <widget class="QLineEdit" name="lineEdit_5">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>DOSIMETRO DE RUIDO</string>
  </property>
  <property name="cursorPosition">
   <number>18</number>
  </property>
  <property name="readOnly">
   <bool>true</bool>
  </property>
 </widget>
</item>
<item row="0" column="2">
 <widget class="QLabel" name="label_12">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>5. ET. PATRI</string>
  </property>
 </widget>
</item>
```

```
<item row="0" column="3">
 <widget class="QLineEdit" name="ET_PATRI">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>00490628</string>
  </property>
  <property name="cursorPosition">
   <number>8</number>
  </property>
  <property name="readOnly">
   <bool>true</bool>
  </property>
 </widget>
</item>
<item row="1" column="0">
 <widget class="QLabel" name="label_10">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>6. MARCA</string>
  </property>
 </widget>
</item>
<item row="1" column="1">
 <widget class="QLineEdit" name="lineEdit_6">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>QUEST TECHNOLOGIES</string>
```

```
  </property>
  <property name="cursorPosition">
   <number>18</number>
  </property>
  <property name="readOnly">
   <bool>true</bool>
  </property>
 </widget>
</item>
<item row="1" column="2">
 <widget class="QLabel" name="label_13">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>7. MODELO</string>
  </property>
 </widget>
</item>
<item row="1" column="3">
 <widget class="QLineEdit" name="lineEdit_9">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>Q-400</string>
  </property>
  <property name="cursorPosition">
   <number>5</number>
  </property>
  <property name="readOnly">
   <bool>true</bool>
  </property>
```

```
 </widget>
</item>
<item row="2" column="0">
 <widget class="QLabel" name="label_11">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>8. SERIE</string>
  </property>
 </widget>
</item>
<item row="2" column="1" colspan="2">
 <widget class="QLineEdit" name="lineEdit_7">
  <property name="styleSheet">
   <string notr="true">color: rgb(0, 0, 0);</string>
  </property>
  <property name="text">
   <string>QD6050012</string>
  </property>
  <property name="cursorPosition">
   <number>9</number>
  </property>
  <property name="readOnly">
   <bool>true</bool>
  </property>
 </widget>
</item>
<item row="2" column="3">
 <spacer name="horizontalSpacer_6">
  <property name="orientation">
   <enum>Qt::Horizontal</enum>
  </property>
```

```
        <property name="sizeHint" stdset="0">
         <size>
          <width>121</width>
          <height>20</height>
         </size>
        </property>
       </spacer>
      </item>
     </layout>
    </widget>
   </item>
  </layout>
  <zorder>groupBox_2</zorder>
  <zorder>groupBox</zorder>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```

ANEXO G
PROGRAMA GRÁFICO TSI

```
/*************************************************************************
File:ScrollWindowsData.h
*************************************************************************/
#ifndef SCROLLWINDOWSDATA_H
#define SCROLLWINDOWSDATA_H
#include <qscrollview.h>
#include <qwidget.h>
#include "LCDData/LCDDataImpl.h"
#include "WidgetWindowsDataImpl.h"
#include <map>
#include <iostream>
using namespace std;
class QGridLayout;
struct Ids{
int IDMagnitude;
int IDUnit;
};
struct Strings{
QString SMagnitude;
QStringList SLUnits;
};
#define MAX_Num_LCDDataImpl 20
class ScrollWindowsData : public QWidget{
Q_OBJECT
public:
ScrollWindowsData(QWidget* parent = 0, const char* name = 0, WFlags fl = 0);
~ScrollWindowsData();
friend class ConfUnitImpl;
int Num_LCDDataImpl;
QGridLayout *GridLayoutSV; //GridLayout to ScrollView
map <int,int> mapIdPos_IdMag;
map <int,int> mapIdMag_IdPos;  //Sólo de las visibles actualmente dinámico
map <int,int> mapIdMag_IdUnit; //De todas las magnitudes dinámico
```

```
map <int,Strings>  mapIdMag_Strings;
map <QString,int> mapStringMag_IdMag;
LCDDataImpl *ArrayPtrDataImpl[MAX_Num_LCDDataImpl];
WidgetWindowsDataImpl *myWidgetWindowsDataImpl;
QGridLayout *myGridLayout;
QScrollView *myScrollView;
int countIndex;
int rowGridLayoutSWD;
int colGridLayoutSWD;
void setElementLCDData (int Position, QString myMagnitude, QStringList myListUnit,
QString myCurrentUnit);
void LoadConfigFromRam_PosMag(int IdPos,int IdMag);
void LoadConfgFromRam_MagUnit(int IdMag,int IdUnit);
void Load_mapIdMag_Strings(int myIDMag,Strings myStrings);
void setTextLCDDataImpl(int position,float value,int prec=1);
void buildAllWindowsWidgetFromMAP();
public :
signals:
void settingUnitDisplayData(int Magnitude, int Unit);
};
#endif
/*****************************************************************************
```

File:ScrollWindowsData.cpp

```
*****************************************************************************/
#include "SCrollWindowsData.h"
#include <qlayout.h>
#include <qpushbutton.h>
#include <qlabel.h>
ScrollWindowsData::ScrollWindowsData(QWidget* parent , const char* name , WFlags fl
): QWidget( parent, name, fl )
{
  Num_LCDDataImpl=0;
  countIndex=0;
```

```
  rowGridLayoutSWD=0;
  colGridLayoutSWD=0;
  myGridLayout= new QGridLayout(this);
  myScrollView=new QScrollView(this);
  myGridLayout->setSpacing(0);
  myGridLayout->setMargin(0);
  myScrollView->setSizePolicy(QSizePolicy(QSizePolicy::Expanding,
  QSizePolicy::Expanding, myScrollView->sizePolicy().hasHeightForWidth() ) );
  myGridLayout->addWidget(myScrollView,0,0);
  myScrollView->verticalScrollBar()->setLineStep(3);
  myWidgetWindowsDataImpl=new WidgetWindowsDataImpl(myScrollVie>viewport());
  myWidgetWindowsDataImpl->setPtrmyScrollWindowsData(this);
  myScrollView->enableClipper(true);
  myScrollView->addChild(myWidgetWindowsDataImpl);
  ArrayPtrDataImpl[0]=myWidgetWindowsDataImpl->LCDDataImpl0;
  ArrayPtrDataImpl[1]=myWidgetWindowsDataImpl->LCDDataImpl1;
  ArrayPtrDataImpl[2]=myWidgetWindowsDataImpl->LCDDataImpl2;
  ArrayPtrDataImpl[3]=myWidgetWindowsDataImpl->LCDDataImpl3;
  ArrayPtrDataImpl[4]=myWidgetWindowsDataImpl->LCDDataImpl4;
  ArrayPtrDataImpl[5]=myWidgetWindowsDataImpl->LCDDataImpl5;
  ArrayPtrDataImpl[6]=myWidgetWindowsDataImpl->LCDDataImpl6;
  ArrayPtrDataImpl[7]=myWidgetWindowsDataImpl->LCDDataImpl7;
  ArrayPtrDataImpl[8]=myWidgetWindowsDataImpl->LCDDataImpl8;
  ArrayPtrDataImpl[9]=myWidgetWindowsDataImpl->LCDDataImpl9;
  ArrayPtrDataImpl[10]=myWidgetWindowsDataImpl->LCDDataImpl10;
  ArrayPtrDataImpl[11]=myWidgetWindowsDataImpl->LCDDataImpl11;
  ArrayPtrDataImpl[12]=myWidgetWindowsDataImpl->LCDDataImpl12;
}
void ScrollWindowsData::setElementLCDData (int Position, QString myMagnitude,
QStringList myListUnit,QString myCurrentUnit)
{
  ArrayPtrDataImpl[Position]->setTextMagnitude(myMagnitude);
  ArrayPtrDataImpl[Position]->setListUnit(myListUnit);
```

```
  ArrayPtrDataImpl[Position]->setCurrentUnit(myCurrentUnit);
}
void ScrollWindowsData::LoadConfigFromRam_PosMag(int IdPos,int IdMag)
{
 mapIdPos_IdMag.insert(pair<int,int>(IdPos,IdMag));
 mapIdMag_IdPos.insert(pair<int,int>(IdMag,IdPos));
}
void ScrollWindowsData::LoadConfgFromRam_MagUnit(int IdMag,int IdUnit)
{
 mapIdMag_IdUnit.insert(pair<int,int>(IdMag,IdUnit));
}
void ScrollWindowsData::Load_mapIdMag_Strings(int myIDMag,Strings myStrings)
{
 mapIdMag_Strings.insert(pair<int,Strings>(myIDMag,myStrings));
 mapStringMag_IdMag.insert(pair<QString,int>(myStrings.SMagnitude,myIDMag));
}
void ScrollWindowsData::buildAllWindowsWidgetFromMAP()
{
  int ItemsVisibles,ITemsTotal;
  QString CurrentUnit;
  int  IdUnit;
  Strings myStrings;
  int countUnit=0;
  ItemsVisibles=mapIdPos_IdMag.size();
  ITemsTotal=mapIdMag_Strings.size();
  for(int i=0;i<ItemsVisibles;i++)
  {
    myStrings=mapIdMag_Strings[mapIdPos_IdMag[i]];
    IdUnit=mapIdMag_IdUnit[mapIdPos_IdMag[i]];
     for (QStringList::Iterator it = myStrings.SLUnits.begin(); it !=myStrings.SLUnits.end();
     ++it ) {
         if(IdUnit==countUnit)
         {
```

```
          CurrentUnit=(*it);
          break;
        }
      countUnit++;
    }
   setElementLCDData(i,myStrings.SMagnitude,myStrings.SLUnits,CurrentUnit);
   ArrayPtrDataImpl[i]->show();
 }
   for(int i=ItemsVisibles;i<ITemsTotal;i++)
   {
   ArrayPtrDataImpl[i]->hide();
   }
}
void ScrollWindowsData::setTextLCDDataImpl(int position,float value,int prec)
{
   Strings myStrings=mapIdMag_Strings[mapIdPos_IdMag[position]];
   int IdUnit=mapIdMag_IdUnit[mapIdPos_IdMag[position]];
   QString CurrentUnit;
   QString SValue;
   int countUnit=0;
    for ( QStringList::Iterator it = myStrings.SLUnits.begin(); it!=myStrings.SLUnits.end();
    ++it ) {
      if(IdUnit==countUnit)
      {
        CurrentUnit=(*it);
         break;
      }
      countUnit++;
    }
   SValue=QString::number ( value, 'f',prec);
   SValue+="";
   SValue+=CurrentUnit;
   ArrayPtrDataImpl[position]->setTextLineEdit(SValue);
```

```
}
ScrollWindowsData::~ScrollWindowsData(){}
/*****************************************************************************
```

File:Plotter.h

```
*****************************************************************************/
#ifndef Plotter_H
#define Plotter_H
#include "../.ui/release-shared/Plotter_UI.h"
#include <map>
#include <deque>
#include <qdatetime.h>
#include <iostream>
using namespace std;
struct ConfigurationChannel{
bool Enable;
QString CurrentMagnitude;
QString CurrentUnit;
float MaxValue;
float MinValue;
};
class Plotter : public Plotter_UI{
Q_OBJECT
public:
Plotter(QWidget * parent = 0, const char * name = 0, WFlags f = 0);
~Plotter();
map <QString,QStringList> Curves;
int Frecuency,RecommendedPtsPlotting;
QTime SaveL_TimeRight;
void setCurves(QString Magnitude,QStringList Units);
void LoadConfig(QString fileconfig);
void SaveConfig(QString fileconfig);
void LoadOnePtCurveCh1(float myPoint);
void LoadOnePtCurveCh2(float myPoint);
```

```
void LoadPtsCurveCh1(int numberPoints, float *Points);
void LoadPtsCurveCh2(int numberPoints, float *Points);
void LoadCurves(int nPtsCh1,float * PtsCh1,int nPtsCh2,float * PtsCh2,bool
myUpdateDisplay=true);
void ClearCurveCh1();
void ClearCurveCh2();
void updateDisplay();
void clearReplySetScreen();
void ShowAxis_L();
void ShowAxis_R();
void HideAxis_L();
void HideAxis_R();
void clearReplySetLineTime();
void clearDeques();
int SampleFrec;
struct replySetScreen{
bool cameResponse;
bool isCh1Enable;
bool isOkCh1;
bool isCh2Enable;
bool isOkCh2;
}myreplySetScreen;
struct replySetLineTime{
bool cameResponse;
bool isOk;
}myreplySetLineTime;
friend class Screen;
friend class ConfigChannImpl;
friend class LabelRotate;
friend class SetupUnitImpl;
friend class LineTime;
friend class SetTimeGraphImpl;
public slots:
```

```
void hRun_Stop(bool on);
void hchangeVFAxisV_L();
void hchangeVFAxisV_R();
void hreplySetScreen(bool enableCh1,bool ok1,bool enableCh2,bool ok2);
void hPb_Start_Stop(bool start_stop);
void hSlider( int value);
signals:
void settingScreen(bool enableCh1,int indexMagnitude1,int indexUnit1,bool enableCh2,int
indexMagnitude2,int indexUnit2);
void settingFrecAndRecPts(int myFrec,int myRecPts);
private:
struct ConfigCh1{
bool EnableCh1;
QString CurrentMagnitudeCh1;
QString CurrentUnitCh1;
float MaxValueCh1;
float MinValueCh1;
}MyConfigCh1;
struct ConfigCh2{
bool EnableCh2;
QString CurrentMagnitudeCh2;
QString CurrentUnitCh2;
float MaxValueCh2;
float MinValueCh2;
}MyConfigCh2;
struct ConfigLineTime{
QString Type;
int minutes;
int seconds;
}myConfigLineTime;
bool run_stop;
QString NameFileConfig;
deque<unsigned short int> myPtsCurvesCh1;
```

```
deque<unsigned short int> myPtsCurvesCh2;
public:
ConfigurationChannel getConfigCh1();
ConfigurationChannel getConfigCh2();
};
#endif
/*****************************************************************************
```

File:Plotter.cpp

```
*****************************************************************************/
#include "Plotter.h"
#include "Screen.h"
#include "AxisV_L.h"
#include "AxisV_R.h"
#include "LabelRotate.h"
#include "LineTime.h"
#include <qpushbutton.h>
#include <qlineedit.h>
#include <qfile.h>
#include <qlabel.h>
#include <qtextstream.h>
#include <qdatastream.h>
#include <qslider.h>
#include <iostream>
#include <algorithm>
using namespace std;
Plotter::Plotter(QWidget * parent , const char * name , WFlags f)
        :Plotter_UI( parent , name , f)
{
 run_stop=true;
 LoadConfig(QString("boot.csv"));
 cout << "Load config boot.csv"<<endl;
 myAxisV_L->myUnit->SetString(MyConfigCh1.CurrentMagnitudeCh1+"
"+MyConfigCh1.CurrentUnitCh1);
```

```
 myAxisV_L->MaxValue->setText(QString::number(MyConfigCh1.MaxValueCh1,'f',1));
myAxisV_L->MinValue->setText(QString::number(MyConfigCh1.MinValueCh1,'f',1));
myAxisV_L->setVAxis();
myAxisV_R->myUnit->SetString(MyConfigCh2.CurrentMagnitudeCh2+"
"+MyConfigCh2.CurrentUnitCh2);
 myAxisV_R->MaxValue->setText(QString::number(MyConfigCh2.MaxValueCh2,'f',1));
myAxisV_R->MinValue->setText(QString::number(MyConfigCh2.MinValueCh2,'f',1));
myAxisV_R->setVAxis();
connect(myAxisV_L,SIGNAL(updateAxisVertical()),this,SLOT(hchangeVFAxisV_L()));
connect(myAxisV_R,SIGNAL(updateAxisVertical()),this,SLOT(hchangeVFAxisV_R()));
connect(myLineTime->Pb_Start_Stop,SIGNAL(toggled(bool)),this,
SLOT(hPb_Start_Stop(bool) ));
connect(myLineTime->slider,SIGNAL(valueChanged ( int )),this,SLOT(hSlider( int )));
myLineTime->slider->setFocusPolicy(QWidget::ClickFocus);
myLineTime->slider->setDisabled(true);
if(myConfigLineTime.Type==QString("Span"))
 myLineTime->StartLinesTime();
}
void Plotter::setCurves(QString Magnitude,QStringList Units)
{
  Curves[Magnitude]=Units;
}
void Plotter::LoadConfig(QString fileconfig)
{
    QString temp,name;
    float maxval,minval;
    int timevalue;
    QString myFile("/mnt/");
    myFile+=fileconfig;
    QFile myf( myFile );
    myf.open( IO_ReadOnly );
    QDataStream s( &myf );
    s>>temp>>NameFileConfig;
```

```
myf.close();
myFile="/mnt/";
myFile+=NameFileConfig;
myf.setName(myFile );
myf.open( IO_ReadOnly );
s.setDevice( &myf );
while(!s.atEnd ())
{
    s>>temp;
   if(temp==QString("Channel 1:,"))
   {
     s>>temp;
     if(temp==QString("Enabled"))
     {
       MyConfigCh1.EnableCh1=true;
     }
     else
     {
       MyConfigCh1.EnableCh1=false;
     }
     continue;
   }
   if(temp==QString("Magnitude Ch1:,"))
   {
     s>>temp;
     MyConfigCh1.CurrentMagnitudeCh1=temp;
     continue;
   }
   if(temp==QString("Unit Ch1:,"))
   {
     s>>temp;
     MyConfigCh1.CurrentUnitCh1=temp;
     continue;
```

```
}
if(temp==QString("MaxValue Ch1:,"))
{
   s>>maxval;
   MyConfigCh1.MaxValueCh1=maxval;
   continue;
}
if(temp==QString("MinValue Ch1:,"))
{
  s>>minval;
  MyConfigCh1.MinValueCh1=minval;
  continue;
}
//Channel 2
if(temp==QString("Channel 2:,"))
{
   s>>temp;
  if(temp==QString("Enabled"))
  {
     MyConfigCh2.EnableCh2=true;
  }
  else
  {
     MyConfigCh2.EnableCh2=false;
  }
     continue;
}
if(temp==QString("Magnitude Ch2:,"))
{
  s>>temp;
  MyConfigCh2.CurrentMagnitudeCh2=temp;
  continue;
}
```

```
if(temp==QString("Unit Ch2:,"))
{
  s>>temp;
  MyConfigCh2.CurrentUnitCh2=temp;
  continue;
}
if(temp==QString("MaxValue Ch2:,"))
{
  s>>maxval;
  MyConfigCh2.MaxValueCh2=maxval;
  continue;
}
if(temp==QString("MinValue Ch2:,"))
{
 s>>minval;
 MyConfigCh2.MinValueCh2=minval;
 continue;
}
if(temp==QString("Type:,"))
{
  s>>temp;
  myConfigLineTime.Type=temp;
  continue;
}
if(temp==QString("Minutes:,"))
{
  s>>timevalue;
  myConfigLineTime.minutes=timevalue;
  continue;
}
if(temp==QString("Seconds:,"))
{
  s>>timevalue;
```

```
        myConfigLineTime.seconds=timevalue;
        continue;
       }
   }
myf.close();
}
void Plotter::SaveConfig(QString myNameFileConfig)
{
   QString myFile("/mnt/");
   myFile+=myNameFileConfig;
   QFile myf( myFile );
   myf.open( IO_WriteOnly );
   QDataStream s( &myf );
   cout << myFile.latin1();
   s<<QString("Comments:,Config Curves")<<QString("\n");
   s<<QString("Configuration:,*Current Configuration")<<QString("\n");
   s<<QString("Channel 1:,");
   if(MyConfigCh1.EnableCh1==true)
   s<< QString("Enabled");
   else
   s<< QString("Disabled");
   s<<QString("\n");
   s<<QString("MagnitudeCh1:,")<<MyConfigCh1.CurrentMagnitudeCh1
   <<QString("\n");
   s<<QString("Unit Ch1:,")<<MyConfigCh1.CurrentUnitCh1<<QString("\n");
   s<<QString("MaxValue Ch1:,")<<MyConfigCh1.MaxValueCh1<<QString("\n");
   s<<QString("MinValue Ch1:,")<<MyConfigCh1.MinValueCh1<<QString("\n");
   s<<QString("Channel 2:,");
   if(MyConfigCh2.EnableCh2==true)
   s<< QString("Enabled");
   else
   s<< QString("Disabled");
   s<<QString("\n");
```

```
    s<<QString("MagnitudeCh2:,")<<MyConfigCh2.CurrentMagnitudeCh2
    <<QString("\n");
    s<<QString("Unit Ch2:,")<<MyConfigCh2.CurrentUnitCh2<<QString("\n");
    s<<QString("MaxValue Ch2:,")<<MyConfigCh2.MaxValueCh2<<QString("\n");
    s<<QString("MinValue Ch2:,")<<MyConfigCh2.MinValueCh2<<QString("\n");
    s<<QString("Config Line Time:,")<<QString("\n");
    s<<QString("Type:,")<<myConfigLineTime.Type<<QString("\n");
    s<<QString("Minutes:,")<<myConfigLineTime.minutes<<QString("\n");
    s<<QString("Seconds:,")<<myConfigLineTime.seconds<<QString("\n");
    myf.close();
}
void Plotter::LoadOnePtCurveCh1(float myPoint)
{
  if( myPtsCurvesCh1.size()>= 1000)
  {
     for(int i=0;i<100;i++)
     myPtsCurvesCh1.pop_back();
  }
  myPtsCurvesCh1.push_front (myPoint);
}
void Plotter::LoadPtsCurveCh1(int numberPoints, float *Points)
{
  for(int i=0;i<numberPoints;i++)
   LoadOnePtCurveCh1(Points[i]);
}
void Plotter::LoadOnePtCurveCh2(float myPoint)
{
  if( myPtsCurvesCh2.size()>= 1000)
  {
    for(int i=0;i<100;i++)
     myPtsCurvesCh2.pop_back();
  }
  myPtsCurvesCh2.push_front (myPoint);
```

```
}
void Plotter::LoadPtsCurveCh2(int numberPoints, float *Points)
{
  for(int i=0;i<numberPoints;i++)
   LoadOnePtCurveCh2(Points[i]);
}
//
void Plotter::LoadCurves(int nPtsCh1,float * PtsCh1,int nPtsCh2,float * PtsCh2,bool
myUpdateDisplay)
{
 if(!run_stop) return;
 LoadPtsCurveCh1(nPtsCh1, PtsCh1);
 LoadPtsCurveCh2(nPtsCh2, PtsCh2);
 if(myUpdateDisplay)updateDisplay();
}
void Plotter::ClearCurveCh1()
{
 if(myPtsCurvesCh1.size()!=0)
  myPtsCurvesCh1.clear();
}
void Plotter::ClearCurveCh2()
{
 if(myPtsCurvesCh2.size()!=0)
  myPtsCurvesCh2.clear();
}
void Plotter::updateDisplay()
{
 myScreen->update();
}
void Plotter::hRun_Stop(bool on)
{
  run_stop=on;
}
```

```
void Plotter::hchangeVFAxisV_L()
{
  QString valueMax,valueMin;
  valueMax=myAxisV_L->MaxValue->text();
  MyConfigCh1.MaxValueCh1= valueMax.toFloat();
  valueMin=myAxisV_L->MinValue->text();
  MyConfigCh1.MinValueCh1= valueMin.toFloat();
  if(!run_stop)
  {
    updateDisplay();
  }
}
void Plotter::hchangeVFAxisV_R()
{
   QString valueMax,valueMin;
   valueMax=myAxisV_R->MaxValue->text();
   MyConfigCh2.MaxValueCh2= valueMax.toFloat();
   valueMin=myAxisV_R->MinValue->text();
   MyConfigCh2.MinValueCh2= valueMin.toFloat();
   if(!run_stop)
   {
    updateDisplay();
   }
}
void Plotter::hreplySetScreen(bool enableCh1,bool ok1,bool enableCh2,bool ok2)
{
  myreplySetScreen.cameResponse=true;
  myreplySetScreen.isCh1Enable=enableCh1;
  myreplySetScreen.isOkCh1=ok1;
  myreplySetScreen.isCh2Enable=enableCh2;
  myreplySetScreen.isOkCh2=ok2;
}
void Plotter::clearReplySetScreen()
```

```
{
  myreplySetScreen.cameResponse=false;
  myreplySetScreen.isCh1Enable=false;
  myreplySetScreen.isOkCh1=false;
  myreplySetScreen.isCh2Enable=false;
  myreplySetScreen.isOkCh2=false;
}
void Plotter::clearReplySetLineTime()
{
  myreplySetLineTime.cameResponse=false;
  myreplySetLineTime.isOk=false;
}
void Plotter::ShowAxis_L()
{
  myAxisV_L->show();
}
void Plotter::ShowAxis_R()
{
  myAxisV_R->show();
}
void Plotter::HideAxis_L()
{
  myAxisV_L->hide();
}
void Plotter::HideAxis_R()
{
  myAxisV_R->hide();
}
void Plotter::clearDeques()
{
  myPtsCurvesCh1.clear();
  myPtsCurvesCh2.clear();
}
```

```
void Plotter::hPb_Start_Stop(bool start_stop)
{
  if(start_stop)
  {
     cout << "Start" << endl;
     myLineTime->ClearLineTimes();
     myLineTime->Pb_Start_Stop->setText("||");
     myLineTime->slider->setDisabled(true);
     clearDeques();
     myScreen->offset=0;
     run_stop=true;
  }
  else
  {
    cout << "Stop" << endl;
    myLineTime->StopLocalTimer();
    SaveL_TimeRight=myLineTime->myRealTime;
    myLineTime->Pb_Start_Stop->setText(">>");
    run_stop=false;
     if(myScreen->width() < min(myPtsCurvesCh1.size(),myPtsCurvesCh2.size()))
     {
        myLineTime->slider->setMinValue(0);
        myLineTime->slider->setMaxValue(myPtsCurvesCh1.size());
        myLineTime->slider->setValue(myPtsCurvesCh1.size());
        myLineTime->slider->setDisabled(false);
     }
  }
}
void Plotter::hSlider( int value)
{
   myScreen->offset=myPtsCurvesCh1.size() - value;
   if(myConfigLineTime.Type==QString("Real Time"))
   {
```

```
    if(Frecuency!=0)
    {
      int delta_ms;
      QTime myL_TimeRight,myL_TimeLeft;
      delta_ms=myScreen->offset*Frecuency;
      myL_TimeRight=SaveL_TimeRight.addMSecs(-delta_ms);
      myL_TimeLeft=myL_TimeRight.addSecs(- (myConfigLineTime.minutes*60+
      myConfigLineTime.seconds) );
      myLineTime->L_TimeLeft->setText(myL_TimeLeft.toString() );
      myLineTime->L_TimeRight->setText(myL_TimeRight.toString());
      cout << "Slider" <<endl;
    }
  }
 updateDisplay();
}
ConfigurationChannel Plotter::getConfigCh1()
{
  ConfigurationChannel myConfigurationCh1;
  myConfigurationCh1.Enable=MyConfigCh1.EnableCh1;
  myConfigurationCh1.CurrentMagnitude=MyConfigCh1.CurrentMagnitudeCh1;
  myConfigurationCh1.CurrentUnit=MyConfigCh1.CurrentUnitCh1;
  myConfigurationCh1.MaxValue=MyConfigCh1.MaxValueCh1;
  myConfigurationCh1.MinValue=MyConfigCh1.MinValueCh1;
  return myConfigurationCh1;
}
ConfigurationChannel Plotter::getConfigCh2()
{
   ConfigurationChannel myConfigurationCh2;
   myConfigurationCh2.Enable=MyConfigCh2.EnableCh2;
   myConfigurationCh2.CurrentMagnitude=MyConfigCh2.CurrentMagnitudeCh2;
   myConfigurationCh2.CurrentUnit=MyConfigCh2.CurrentUnitCh2;
   myConfigurationCh2.MaxValue=MyConfigCh2.MaxValueCh2;
   myConfigurationCh2.MinValue=MyConfigCh2.MinValueCh2;
```

```
  return myConfigurationCh2;
}
Plotter::~Plotter()
{
 SaveConfig(NameFileConfig);
}
```

ANEXO H

PROGRAMA OTM

```
/*****************************************************************************
```

File:Otm.h

```
*****************************************************************************/
#ifndef _OTM_H
#define _OTM_H
#include <qwidget.h>
class QPushButton;
class QWidgetStack;
class QVBoxLayout;
class QHBoxLayout;
class QButtonGroup;
class QCheckBox;
class ListOtms;
class Otm : public QWidget{
Q_OBJECT
public:
Otm( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
QPushButton *mypb1;
QPushButton *mypb2;
QWidgetStack *myws;
QVBoxLayout *myvb;
QHBoxLayout *myhb;
QHBoxLayout *myhb2;
QButtonGroup* ButtonGroup1;
QCheckBox* CheckBox1;
QCheckBox* CheckBox2;
QVBoxLayout* ButtonGroup1Layout;
ListOtms * myListOtms;
public slots:
void handlermypb2();
};
#endif
/*****************************************************************************
```

```
/*************************************************************************
File:Otm.cpp
*************************************************************************/
#include "otm.h"
#include <qpushbutton.h>
#include <qwidgetstack.h>
#include <qlayout.h>
#include <qstring.h>
#include <qcheckbox.h>
#include <qbuttongroup.h>
#include "ListOtms.h"
#include <TumimedSqlite3.h>
sqlite3 *db;
int num_otm1;
int num_otm2;
QString num;
int callback(void *NotUsed, int argc, char **argv, char **azColName)  {
  num=argv[0];
  return 0;
}
Otm::Otm( QWidget* parent , const char* name , WFlags fl  )
        :QWidget( parent, name, fl )
{
  setCaption("Listas de Otms");
  myvb = new QVBoxLayout(this);
  myws = new QWidgetStack(this);
  myhb2 = new QHBoxLayout(myvb);
  myhb2 ->addWidget(myws);
  mypb1 = new QPushButton(this);
  mypb2 = new QPushButton (this);
  myhb = new QHBoxLayout(myvb);
  myhb->insertStretch(0,4);
  myhb->addWidget(mypb1);
```

```
myhb->addWidget(mypb2);
mypb1->setText("<<");
mypb2->setText(">>");
connect(mypb2,SIGNAL(clicked()),this,SLOT(handlermypb2()));
bool ok;
open_db("/BaseDatos/ListaOtms", &db);
Load_Handler(callback);
query_exe("select COUNT(*) from myOtm where Grupo_OTM=1;");
num_otm1=num.toInt(&ok);
query_exe("select COUNT(*) from myOtm where Grupo_OTM=2;");
num_otm2=num.toInt(&ok);
 if(num_otm2 > 0)
 {
  ButtonGroup1 = new QButtonGroup( this, "ButtonGroup1" );
  ButtonGroup1->setTitle( tr( "Select option" ) );
  ButtonGroup1->setExclusive( TRUE );
  ButtonGroup1->setColumnLayout(0, Qt::Vertical );
  ButtonGroup1->layout()->setSpacing( 6 );
  ButtonGroup1->layout()->setMargin( 11 );
  ButtonGroup1Layout = new QVBoxLayout( ButtonGroup1->layout());
  ButtonGroup1Layout->setAlignment( Qt::AlignTop );
  ButtonGroup1Layout->setSpacing( 6 );
  ButtonGroup1Layout->setMargin( 11 );
  CheckBox1 = new QCheckBox( ButtonGroup1, "CheckBox1" );
  CheckBox1->setText( tr( "Seleccionar Otms actuales1" ) );
  CheckBox1->setChecked(true);
  ButtonGroup1Layout->addWidget( CheckBox1 );
  CheckBox2 = new QCheckBox( ButtonGroup1, "CheckBox2" );
  CheckBox2->setText( tr( "seleccionar Otms pendientes" ) );
  ButtonGroup1Layout->addWidget( CheckBox2 );
  ButtonGroup1->adjustSize();
  myws->addWidget(ButtonGroup1,0);
  myws->raiseWidget(ButtonGroup1);
```

```
   mypb1->hide();
  }
}
void Otm::handlermypb2()
{
  if(CheckBox1->isChecked())
  myListOtms = new ListOtms(this,0,0,1);
  else
  myListOtms = new ListOtms(this,0,0,2);
  myws->addWidget(myListOtms,1);
  myws->raiseWidget(myListOtms);
  mypb2->hide();
  close_db(db);
}
/*****************************************************************************
```

File:buzzer.h

```
*****************************************************************************/
#ifndef BUZZER_H
#define BUZZER_H
#include <qtimer.h>
class Buzzer;
#ifdef ENABLEBUFFER
extern Buzzer *pBuzzer;
#endif
class Buzzer : public QObject{
Q_OBJECT
public:
int fd;
int frequency;
int runTime_ms;
int number_beeps;
int counterBeeper;
int interval_Time;
```

```
QTimer *timerRunTime;
QTimer *timerTriggerBeeper;
Buzzer(QObject * parent = 0, const char * name = 0 );
void setParameters(int myfrecuency=2500,int myrunTime_ms=250,
                   int mynumber_beeps=1,   int myinterval_Time=0);
void open_buzzer(void);
void close_buzzer(void);
void set_buzzer_freq(void);
void stop_buzzer(void);
void startTimers(void);
void beeps(void);
public slots:
void handlerRunTime();
void handlerTriggerBeeper();
};
#endif
/*****************************************************************************
```

File:buzzer.cpp

```
*****************************************************************************/
#include <qapplication.h>
#include <iostream>
#include "buzzer.h"
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
using namespace std;
#define PWM_IOCTL_SET_FREQ      1
#define PWM_IOCTL_STOP          0
static int fdGlobal=-1;
Buzzer *pBuzzer=0;
```

```
static void closebyexit(void)
{
   if (fdGlobal >= 0) {
         ioctl(fdGlobal, PWM_IOCTL_STOP);
         close(fdGlobal);
         fdGlobal=-1;
         }
}
Buzzer::Buzzer(QObject * parent,const char * name):QObject(parent,name)
{
 fd=-1;
 pBuzzer=this;
 setParameters();
 timerRunTime=new QTimer;
 connect(timerRunTime,SIGNAL(timeout()),this,SLOT(handlerRunTime()));
 timerTriggerBeeper=new QTimer;
 connect(timerTriggerBeeper,SIGNAL(timeout()),this,SLOT(handlerTriggerBeeper()));
}
void Buzzer::handlerRunTime()
{
 stop_buzzer();
}
void Buzzer::handlerTriggerBeeper()
{
 set_buzzer_freq();
 timerRunTime->start(runTime_ms,TRUE);
 counterBeeper--;
 if(counterBeeper<=1){ timerTriggerBeeper->stop(); counterBeeper=number_beeps;};
}
void Buzzer::setParameters(int myfrecuency,int myrunTime_ms,int mynumber_beeps,
                           int myinterval_Time)
{
 frequency=myfrecuency;
```

```
 runTime_ms=myrunTime_ms;
 number_beeps=mynumber_beeps;
 counterBeeper=number_beeps;
 interval_Time=myinterval_Time;
}
void Buzzer::open_buzzer(void)
{
      fdGlobal=fd = open("/dev/pwm", O_RDWR);
         if (fd < 0) {
                  perror("open pwm_buzzer device");
                  exit(1);
         }
         atexit(closebyexit);
}
void Buzzer::close_buzzer(void)
{
         if (fd >= 0) {
                  ioctl(fd, PWM_IOCTL_STOP);
                  close(fd);
                  fd = -1;
            fdGlobal=-1;
         }
}
void Buzzer::set_buzzer_freq(void)
{
         int ret = ioctl(fd, PWM_IOCTL_SET_FREQ, frequency);
         if(ret < 0) {
                  perror("set the frequency of the buzzer");
                  exit(1);
         }
}
void Buzzer::startTimers(void)
{
```

```
        timerRunTime->start(runTime_ms,TRUE);
        if(number_beeps!=1)
        timerTriggerBeeper->start(interval_Time);
}
void Buzzer::stop_buzzer(void)
{
          int ret = ioctl(fd, PWM_IOCTL_STOP);
          if(ret < 0) {
                    perror("stop the buzzer");
                    exit(1);
          }
}
void Buzzer::beeps(void)
{
 set_buzzer_freq();
 startTimers();
}
/*****************************************************************************
```

File:editValueInstr.h

```
*****************************************************************************/
#ifndef _EDITVALUEINST_H
#define _EDITVALUEINST_H
#include "customlineedit.h"
#include <sqlite3.h>
class ERROR : public QObject{
Q_OBJECT
public:
ERROR(QObject * parent = 0, const char * name = 0);
public slots:
void soundnotchannel(const QCString & msg, const QByteArray & data);
};
class QCopChannel;
#ifndef  MYCHANNELUNIT
```

```
extern QCopChannel *mychannelunit;
extern sqlite3 *ptr_BdUnits;
#endif
int callback(void *NotUsed, int argc, char **argv, char **azColName);
class EditValueInst : public QWidget{
Q_OBJECT
public:
EditValueInst(QWidget * parent = 0, const char * name = 0, WFlags f = 0);
CustomLineEdit * mycustomlineedit1;
QString *unidad;
bool validatedbyenter;
void paintEvent ( QPaintEvent * mypaintevent);
QSize sizeHint () const ;
public slots:
void myslot(){};
void textchanged1(const QString &);
int handlerchecktext(const QString & );
int handlercheckbyenter(const QString &);
void connect_channel();
void disconnect_channel();
void handlermessageunit(const QCString & msg, const QByteArray & data) ;
signals:
void out_thread();
void in_thread();
};
#endif
/*****************************************************************************
```

File:editValueInstr.cpp

```
*****************************************************************************/
#define MYCHANNELUNIT
#include <qpixmap.h>
#include <qpainter.h>
#include <editValueInstr.h>
```

```
#include <qapplication.h>
#include <qsound.h>
#include <qcopchannel_qws.h>
#include <qdatastream.h>
#include <iostream>
#define ENABLEBUFFER
#include "buzzer.h"
using namespace std;
bool unitfound=false;
QString PseudoMyUnit;
QString MyUnit;
QCopChannel *mychannelunit=NULL;
sqlite3 *ptr_BdUnits=NULL;
ERROR::ERROR(QObject * parent , const char * name  ):QObject(  parent, name  ){}
void ERROR::soundnotchannel(const QCString & msg, const QByteArray & data)
{
 QSound::play("sounds/1.wav");
 pBuzzer->setParameters(2500,250,2,500);
 pBuzzer->beeps();
}
ERROR myerror;
int callback2(void *NotUsed, int argc, char **argv, char **azColName)
{
  if(argc != 0){
   MyUnit  =(argv[0] ? argv[0] : "NULL");
   unitfound=true;
   }
   else
   unitfound=false;
  return 0;
}
EditValueInst::EditValueInst(QWidget * parent , const char * name , WFlags f ):
                        QWidget(  parent, name , f  )
```

```
{
mycustomlineedit1 = new CustomLineEdit (this);
mycustomlineedit1->setGeometry(0,0,70,25);
mycustomlineedit1->setAlignment(Qt::AlignLeft);
connect(mycustomlineedit1,SIGNAL(s_checkinputtext(const QString &)),this,
        SLOT(handlerchecktext(const QString & )));
connect(mycustomlineedit1,SIGNAL(s_checkbyenter(const QString &)),this,
        SLOT(handlercheckbyenter(const QString &)));
connect(mycustomlineedit1,SIGNAL(s_connect_channel()),this,
        SLOT(connect_channel()));
connect(mycustomlineedit1,SIGNAL(s_disconnect_channel()),this,
         SLOT(disconnect_channel()));
unidad =new QString;
*unidad= QString("Intensidad_corriente");
validatedbyenter=false;
setName("editvalueinst");
if(mychannelunit){
connect( mychannelunit, SIGNAL(received(const QCString&,const QByteArray&)),
      &myerror, SLOT( soundnotchannel(const QCString&,const QByteArray&)) );
}
}
void EditValueInst::paintEvent ( QPaintEvent * mypaintevent)
{
QWidget::paintEvent(mypaintevent);

        if(mycustomlineedit1->hasFocus())
        {
        QPixmap *mypixmap1= new QPixmap("/BOTONES/activo.png");
        QPainter p;
        p.begin( this );
        p.drawPixmap( mycustomlineedit1->x() + mycustomlineedit1->width() + 5,
            mycustomlineedit1->y()+ 5 ,*mypixmap1);
        p.end();
```

```
        }
        else
        {
        QPixmap *mypixmap1= new QPixmap("/BOTONES/desactivo.png");
        QPainter p;
        p.begin( this  );
        p.drawPixmap( mycustomlineedit1->x() + mycustomlineedit1->width() + 5,
             mycustomlineedit1->y()+ 5 ,*mypixmap1);
        p.end();
        }
}
void EditValueInst::textchanged1(const QString &){
}
int EditValueInst::handlerchecktext(const QString & mystring )
{
int codeerror=0;
unsigned int i;
bool characterexist=false;
bool ok;
int rc;
char *zErrMsg = 0;
if(validatedbyenter){
validatedbyenter=false;
return codeerror;
}
  int longitud= mystring.length();
  if(longitud != 0){
    QString t=mystring.simplifyWhiteSpace();
    if( t.length() == 0)
      return codeerror;
            for( i=0;i<t.length();i++){
                  if(  t[i] != QChar('.') && !(t[i].isDigit()) ) {
                   characterexist=true;
```

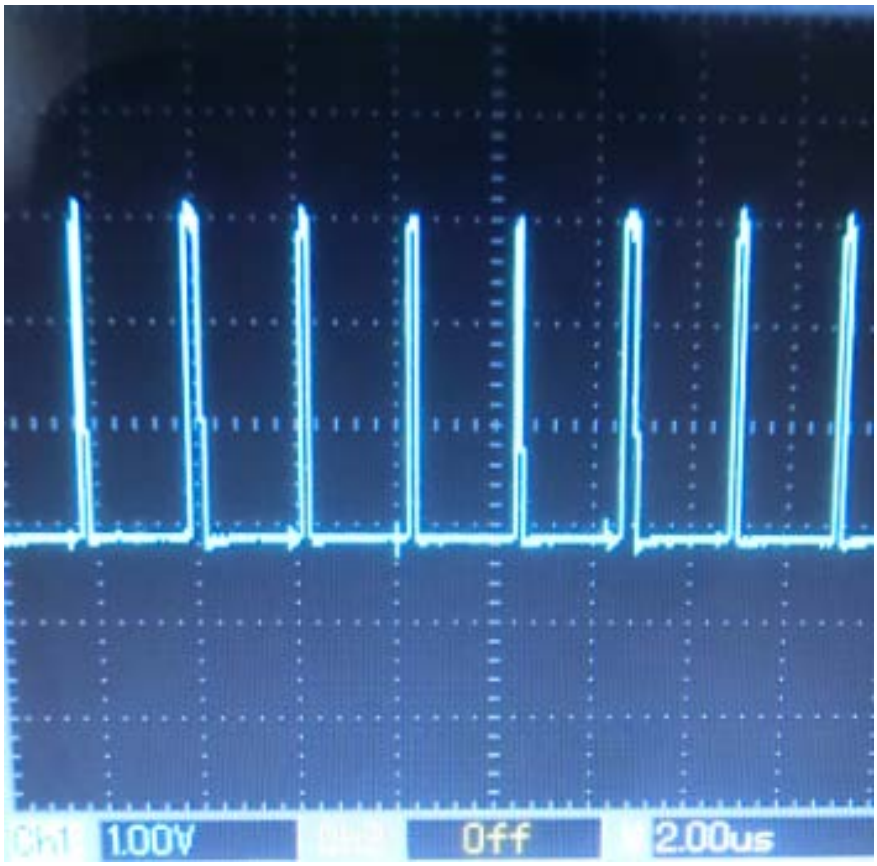
```
                break;
                }
}
if(!characterexist){
                        codeerror=-2;
                        mycustomlineedit1->clear();
   QSound::play("sounds/2.wav");
   pBuzzer->setParameters();
   pBuzzer->beeps();
                        return codeerror;
}
QString myvalue = t.left( i );
myvalue.toDouble(&ok);
if(!ok   || myvalue[myvalue.length()-1]==QChar('.')){
   codeerror=-1;
   mycustomlineedit1->clear();
   QSound::play("sounds/2.wav");
   pBuzzer->setParameters();
   pBuzzer->beeps();
   return codeerror;
}
QString myunit = t.right( t.length() - i );
myunit =  myunit.simplifyWhiteSpace();
QString mysql("select unidad from ");
mysql+= *unidad;
mysql+=" where pseudounidad=\"";
mysql+=myunit;
mysql+="\"";
                rc = sqlite3_exec(ptr_BdUnits, mysql.latin1(), callback2, 0,
                  &zErrMsg);
                if( rc!=SQLITE_OK ){
                fprintf(stderr, "SQL error: %s\n", zErrMsg);
                sqlite3_free(zErrMsg);
```

```
                    }
                if(  !unitfound   )
                {
                 codeerror=-2;
                 mycustomlineedit1->clear();
                 QSound::play("sounds/2.wav");
                 pBuzzer->setParameters();
                 pBuzzer->beeps();
                }
                else
                {
                  int position;
                  QString mystringline(t);
                  if( (position=mystringline.find(myunit)) != -1){

                     mystringline.replace(position,myunit.length(),MyUnit);
                     if(mystringline[position-1] != ' ')
                      mystringline.insert(position,QChar(' '));

                     mycustomlineedit1->setText(mystringline);
                   }
                      }
                 unitfound=false;
                 return codeerror ;
   }
return codeerror;
}
int EditValueInst::handlercheckbyenter(const QString & mystring)
{
        if(handlerchecktext( mystring ) == 0){
        validatedbyenter=true;
        QApplication::sendEvent(mycustomlineedit1,mycustomlineedit1->mykeyEventTab
);
```

```
        }
      return  0;
}
void EditValueInst::connect_channel()
{
if(mychannelunit != NULL)
{
connect( mychannelunit, SIGNAL(received(const QCString&,const QByteArray&)),
this, SLOT( handlermessageunit(const QCString&,const QByteArray&)) );
CustomLineEdit::exist_connection=true;
disconnect( mychannelunit, SIGNAL(received(const QCString&,const QByteArray&)),
&myerror, SLOT( soundnotchannel(const QCString&,const QByteArray&)) );
}
}
void EditValueInst::disconnect_channel()
{
if(mychannelunit != NULL)
{
  disconnect( mychannelunit,SIGNAL(received(const QCString&,const QByteArray&))
                , this, SLOT( handlermessageunit(const QCString&,const QByteArray&)) );
  CustomLineEdit::exist_connection=false;
  if( (parentWidget()->focusWidget())->name()  != QString("customlineedit") ){
   connect( mychannelunit, SIGNAL(received(const QCString&,const QByteArray&)),
            &myerror, SLOT( soundnotchannel(const QCString&,const QByteArray&)) );
  }
}
}
void EditValueInst::handlermessageunit(const QCString & msg, const QByteArray &
                                data)
{
 if(msg.find(unidad->latin1()) == -1)
 {
      QSound::play("sounds/2.wav");
```

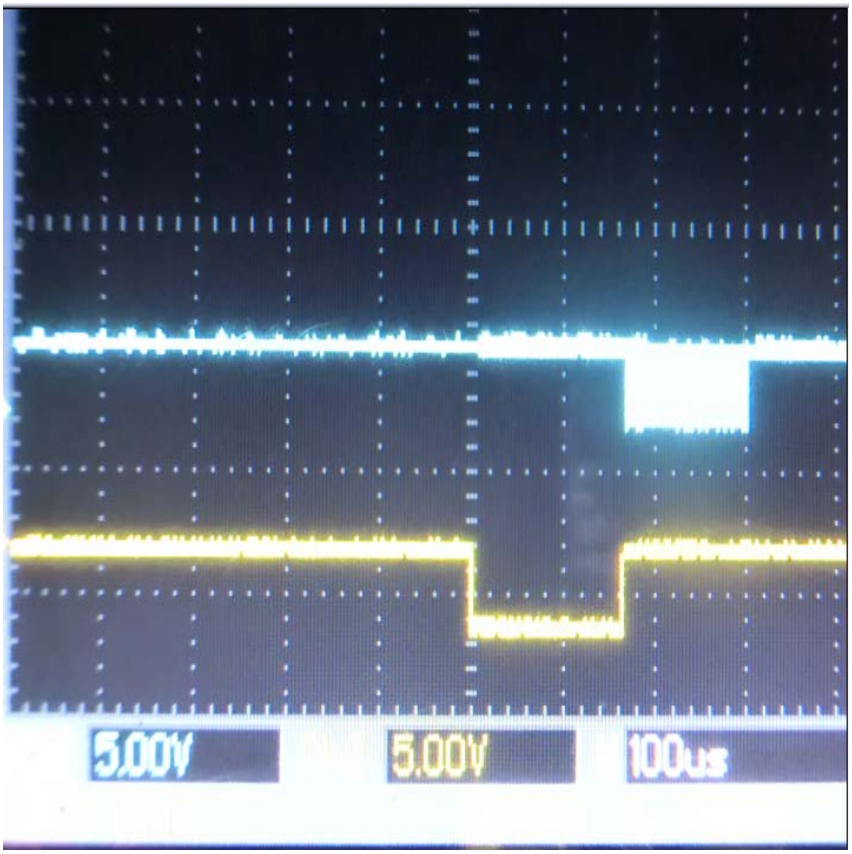
```
        pBuzzer->setParameters();
        pBuzzer->beeps();
  }
  else
  {
          QDataStream in(data,IO_ReadOnly);
          float value;
          QString my_unit;
           QString mytextlineedit;
           QString temp;
            in >> value >> my_unit;

    mytextlineedit+= temp.setNum(value);
    mytextlineedit+=' ';
    mytextlineedit+=my_uni
    mycustomlineedit1->clear();
    mycustomlineedit1->setText(mytextlineedit);
    (this->parentWidget())->setActiveWindow();
    QApplication::sendEvent(mycustomlineedit1,mycustomlineedit1->mykeyEventTab);
  }
}
QSize EditValueInst::sizeHint () const
{
 return QSize(100,25);
}
```

ANEXO I
SEÑALES OSCILOSCOPIO

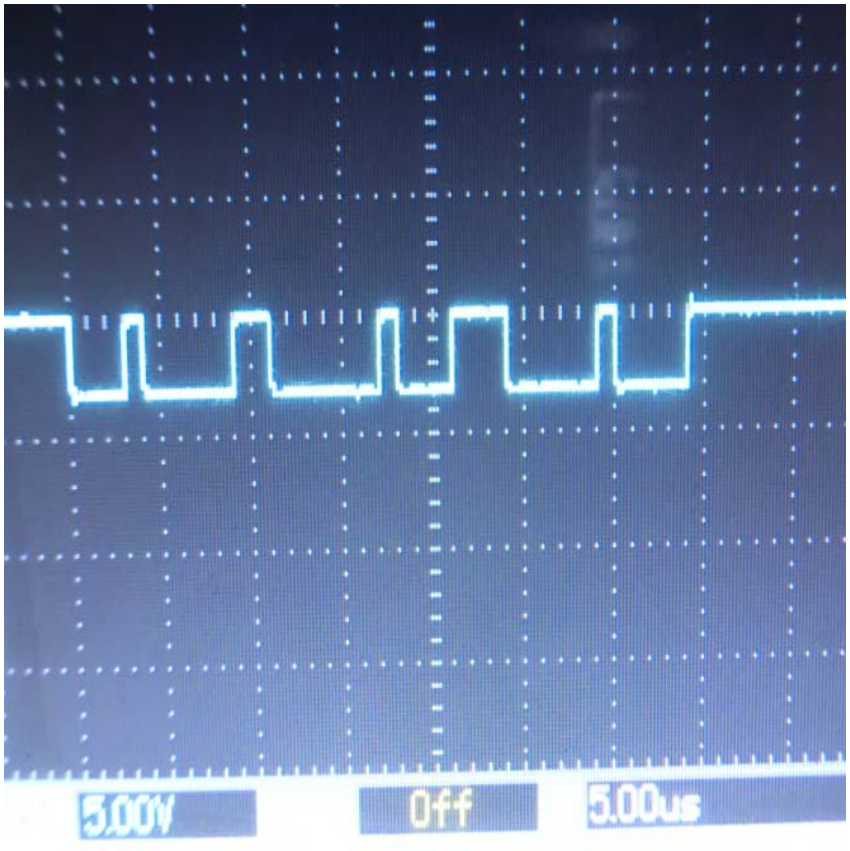


SEÑAL CHIP SELECT 5

SEÑALES CHIP SELECT 5 (CH1) Y RECEPCIÓN DEL ISL83485 (CH2)



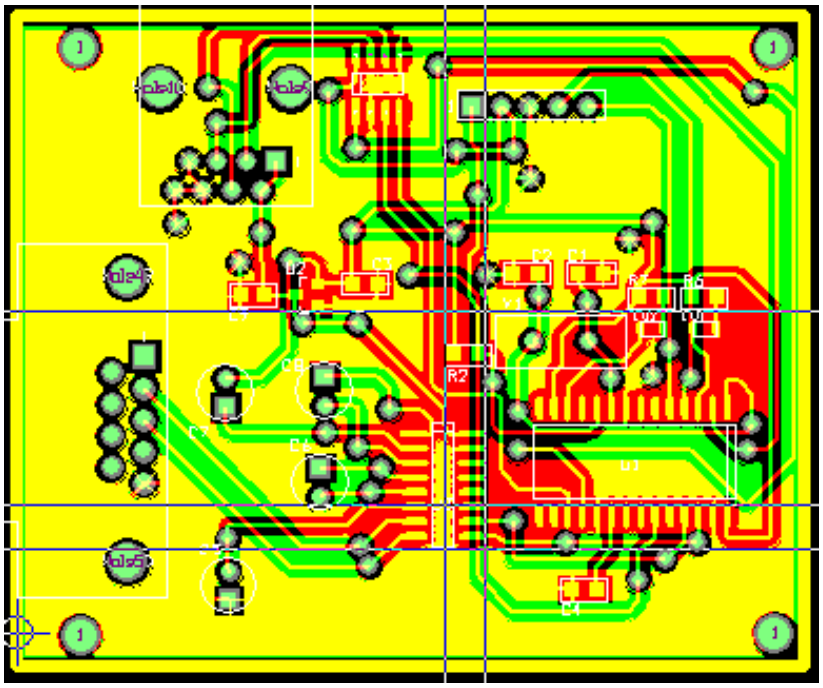
SEÑALES CHIP SELECT 5 (CH1) E INTERRUPCIÓN EINT3 (CH2)



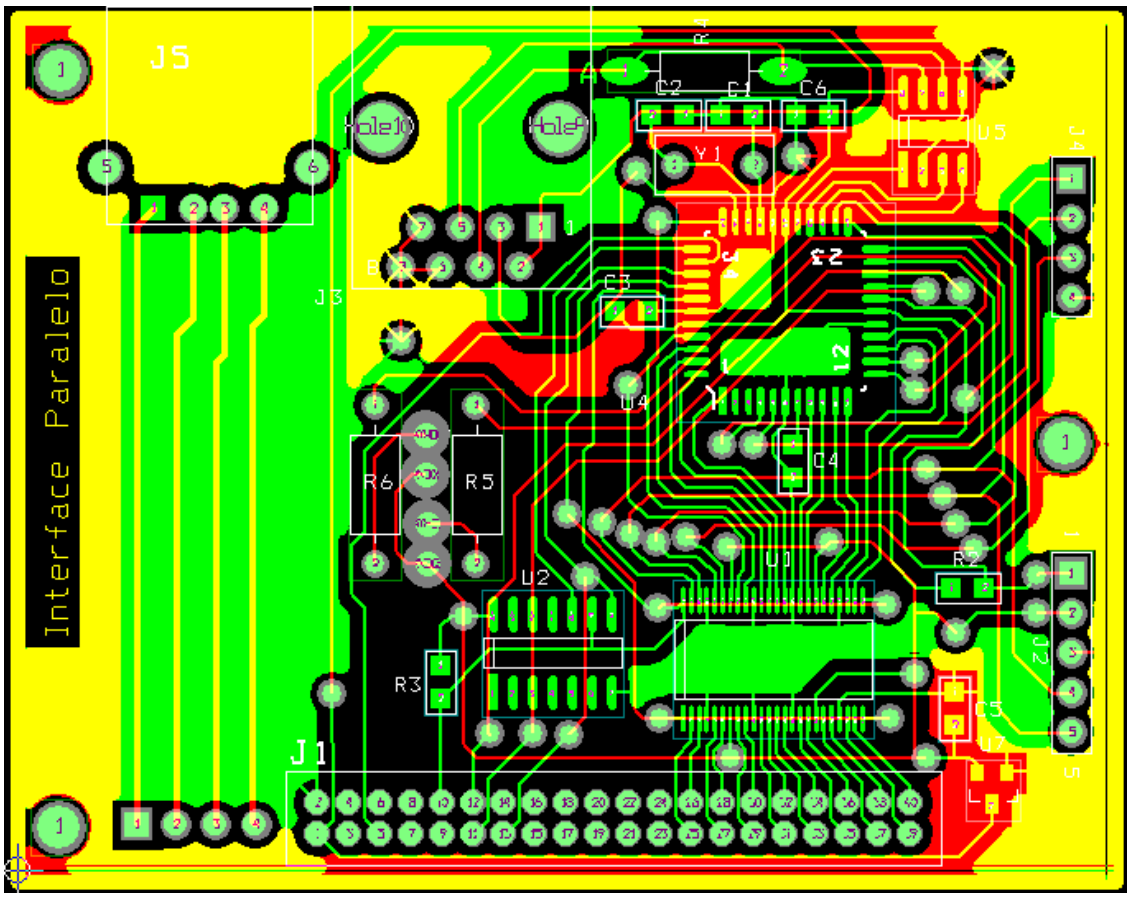
SEÑAL DE RECEPCIÓN DEL ISL83485 A 1MBPS

ANEXO J

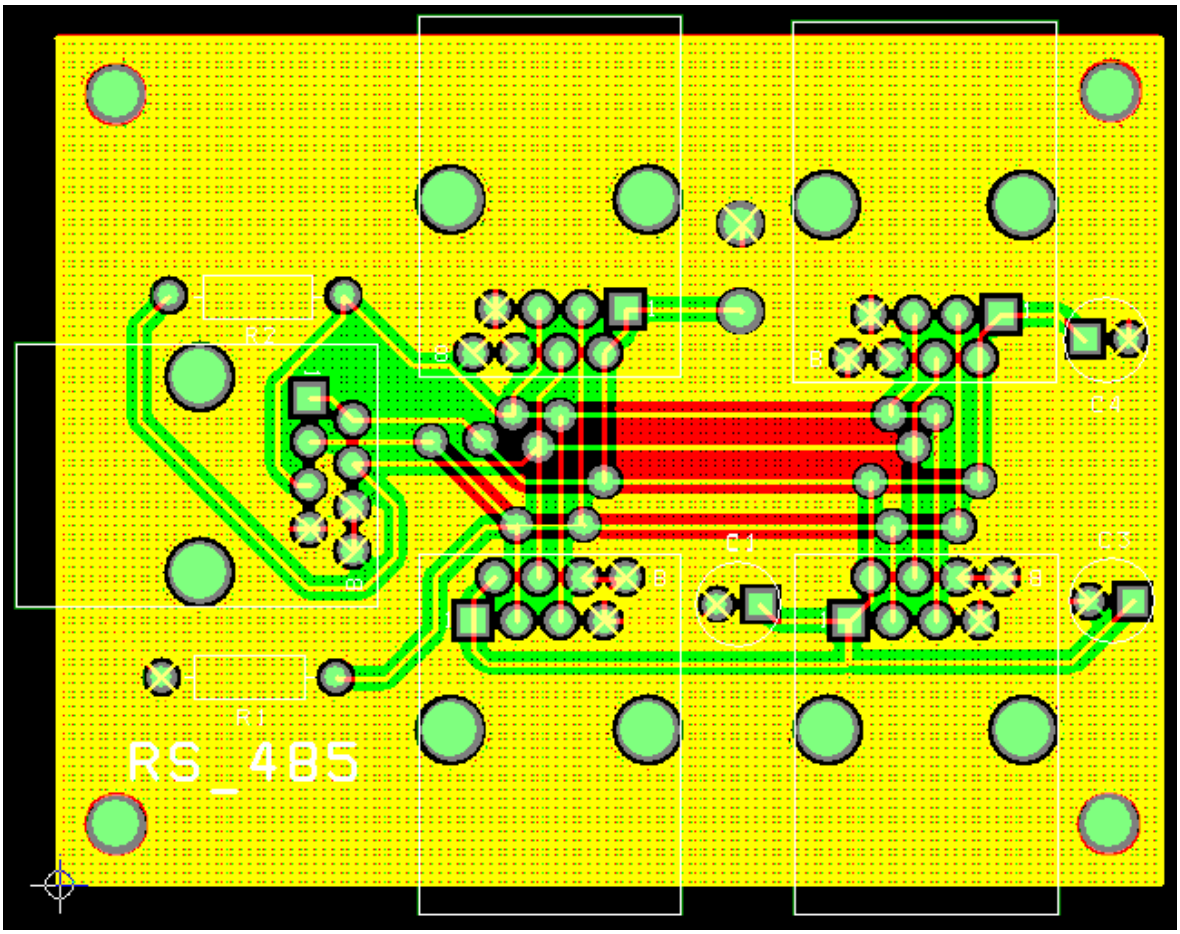
LAYOUT



TARJETA INSTRUMENTO TSI



TARJETA CONCENTRADOR



TARJETA HUB USB

GLOSARIO

ACK: Mensaje que el destino de la comunicación envía al origen de ésta para confirmar la recepción de un mensaje.

Driver: Capa de código que permite interacción con periféricos.

Dspics: Microcontrolador diseñado por la empresa Microchip.

FCY: Frecuencia de ciclo de instrucción de micrcontroladores.

GPL: Licencia Pública General.

GUI: Interfaz Gráfica de Usuario.

IOCTL: Metódo Linux para configurar el modo de operación de un dispositivo

Kernel: Es el núcleo del Sistema Operativo.

Mini2440: Sistema embebidos diseñado por la empresa FriendlyArm

NACK: Mensaje del protocolo que se envía para informar de que en la recepción de una trama de datos ha habido un error

OTM: Orden de Trabajo de Mantenimiento.

PDA: Asistente Digital Personal.

PWM: Modulador por Ancho de pulso

Qtopia: Plataforma gráfica para PDAs que usa las librerías Qt2.2

QT2.2: Librerías Gráficas de código abierto diseñadas por la empresa noruega TrollTech

RS-485: Norma para interconexiones a larga distancia

Sqlite3: Gestor de Base de datos.

S3C2440A: Microcontrolador empleado en la elaboración del la tarjeta Mini2440

SIGIO : Señal de datos disponibles en sistemas Linux.

SIGUSR: Señal de Usuario en sistemas Linux.

TSI: Flujometro de Masa Térmico

Widget: Objetos gráficos empleados en la elaboración de una Interfaz Gráfica de Usuario.

Xml: Lenguaje de Marca Extensible

BIBLIOGRAFÍA

(1) GNU/Embebido Fundamentos Teóricos http://linuxemb.wikidot.com/tesis-c2

(2) Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman "Linux Device Drivers, Third Edition" . O'Reilly Media. 2005

(3) Sreekrisnan Ventateswaran **"**Essential Linux Device Driver".
Prentice Hall. 2007

(4) Interfaz de usuario.
http://www.fismat.umich.mx/~crivera/tesis/node20.html

(5) RS485 http://www.neoteo.com/rs485-domótica-al-alcance-de-tu-mano-15810

(6) Seccions manual reference Dspic.
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2556

(7) Manual, Software, Schematics, Datasheets FriendlyArm
http://www.friendlyarm.net/downloads#schematics

(8) Daniel P. Bovet, Marco Cesati. **"**Understanding the Linux Kernel.".
O'Reilly Media. 2000

(9) Qtopia Reference Documentation.
http://doc.qt.digia.com/qtopia2.2/html/index.htm

(10) TSI General Purpose Thermal Mass Flowmeters

(11) S3C2440A Microcontroller User's Manual