Aplicación en tiempo real para el control de señales en sistemas empotrados

Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Proyecto de Final de Carrera

Autor: Rayco Sánchez García

Tutor: Domingo Benítez Díaz

Tutor de Empresa: Francisco Concepción Concepción

Las Palmas de Gran Canaria, 10 Julio de 2014

Índice

1. Introducción

Hoy en día podemos encontrar tecnología en la mayor parte de aspectos de nuestra vida cotidiana, todo está informatizado. Los ordenadores son un soporte esencial para automatizar procesos y mejorar la productividad. Sin embargo, los sistemas convencionales no son capaces de mantener este nivel de alto rendimiento en entornos en los que el tiempo y la respuesta inmediata a los eventos que se generan son la clave para el buen funcionamiento del sistema.

El sistema operativo en tiempo real es una clase de sistema operativo que surge con el objetivo de funcionar en este tipo de entornos. Se apoya en tres pilares fundamentales; tiempo de respuesta optimizado para las interrupciones hardware, control abierto al programador de los métodos de planificación de selección de procesos en ejecución, y herramientas para incluir las restricciones temporales en el algoritmo.

Un sistema operativo en tiempo real ofrece a los ingenieros grandes facilidades para crear sistemas informatizados totalmente fiables en entornos de alto rendimiento como instalaciones industriales en las que la coordinación entre todos los componentes es vital para la productividad, o incluso entornos en los que un retraso ínfimo en la ejecución de una rutina implica la pérdida de vidas humanas como es el caso de las centralitas de los coches que detectan la posibilidad de colisión, etc.

2. Objetivos del Trabajo

En este trabajo el principal objetivo ha sido crear una aplicación capaz de obtener datos provenientes de señales físicas en un hardware embebido con unas restricciones de tiempo fijadas por el usuario en tiempo de ejecución. Estos datos deberán almacenarse como histórico para el usuario junto a una marca temporal. La aplicación también debe capturar rápidamente los eventos generados en forma de interrupciones hardware para una respuesta temprana a los mismos. Además, la aplicación debe comunicarse vía Ethernet de tal forma que cualquier dispositivo externo pueda conectarse para obtener los datos registrados y configurar parámetros de funcionamiento manteniendo separadas físicamente la parte de interfaz de usuario del resto de la aplicación.

Como objetivo secundario se ha implementado una interfaz gráfica para que el usuario pueda acceder de forma remota para monitorizar las señales.

3. Solución Hardware

Como primer paso para la implementación del sistema es necesario elegir un sistema hardware que nos ayude a satisfacer los requisitos de la aplicación. Los requisitos de tiempo y de atención rápida a eventos hacen necesario el uso de un sistema operativo en tiempo real que nos ayude a satisfacer dichos requisitos, por lo que el hardware elegido deberá ser capaz de ejecutar un sistema operativo de dichas características. A continuación se describen los dispositivos y periféricos que serán necesarios para llevar a cabo el desarrollo de la aplicación.

- Sistema central: Como sistema central encargado de todo el proceso de recogida de datos y captura de eventos se ha elegido al ordenador embebido BeagleBone Black. La BeagleBone Black consiste en un procesador Cortex ARM-A8 de 1GHz capaz de arrancar numerosos sistemas operativos en base Linux, entre ellos uno en tiempo real a través de un BSP (Board Support Project), denominado QNX. Un BSP es un proyecto que permite portar un sistema operativo a una placa programable como BeagleBone o Raspberry PI manteniendo las funciones propias de la placa como los pines de entrada y salida. La BeagleBone Black ofrece soporte para estableces comunicaciones con dispositivos conectados a través de numerosos buses y protocolos de comunicación.

- Dispositivo de soporte para la interfaz gráfica: Para mantener separadas la interfaz de usaurio del resto de la aplicación es necesario hacer uso de un dispositivo distinto a la BeagleBone Black para que ejecute dicha interfaz. Debido a que la interfaz debe ser accesible de forma remota se propone el computador denominado Arduino YUN. El Arduino YUN es una placa programable capaz de generar una red Wi-Fi que hace accesible al dispositivo de forma remota. Esta característica lo hace ideal para el acceso remoto a la interfaz de usuario.

- RTC (Real Time Clock): Debido a que la BeagleBone Black no es capaz de mantener constancia del tiempo en caso de pérdida de la fuente de alimentación, se hará uso de un reloj electrónico con batería interna que procesa la fecha y hora continuamente y la hace accesible a través de una interfaz por I2C.

- Debido a que la comunicación con el dispositivo encargado de la interfaz de usuario no es un procedimiento crítico dentro de la aplicación, el bus de comunicaciones elegido es el Ethernet. El Ethernet es un bus de comunicaciones fácil de manejar con los driver adecuados en C, lenguaje por excelencia para programar este tipo de aplicaciones por su gran rendimiento y eficiencia.

De esta forma, el esquema hardware inicial de la aplicación es el siguiente:

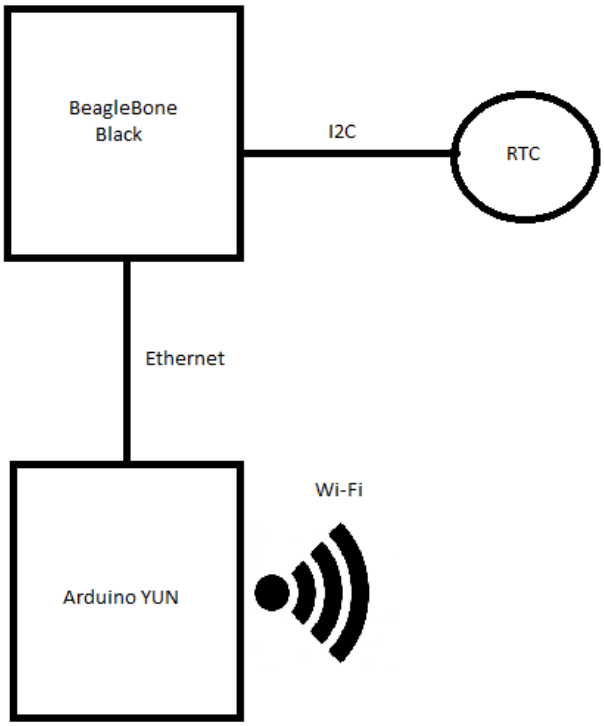


Ilustración 3.1 Esquema Hardware Inicial

Tal y como se muestra en la ilustración 3.1, la BeagleBone representa el sistema central. Ésta se conecta mediante bus I2C al reloj electrónico y mediante Ethernet al sistema encargado de la interfaz de usuario, el Arduino YUN, que genera una red Wi-Fi a la que el usuario puede conectarse para acceder a la interfaz.

3.1 BeagleBone Black

La BeagleBone Black es una placa programable de bajo coste que encuentra su punto fuerte en su gran versatilidad y en su capacidad para convertirse en un ordenador de bajo cunsumo de potencia con un sistema operativo en base Linux.

Su procesador ARM-A8 de 1GHz y su memoria RAM DDR3 de 512 MB la convierten en una unidad de proceso realmente potente en relación a su tamaño. Sin embargo, su mayor potencial se encuentra en su capacidad de conexión gracias a sus 92 pines de entrada/salida, compatibles con buses de comunicación como I2C, UART y SPI, y a sus puertos Ethernet, USB y mini HDMI.

La BeagleBone Black también ofrece una unidad de memoria flash interna de 2GB y la posibilidad de ampliar su capacidad mediante una ranura para tarjetas uSD.

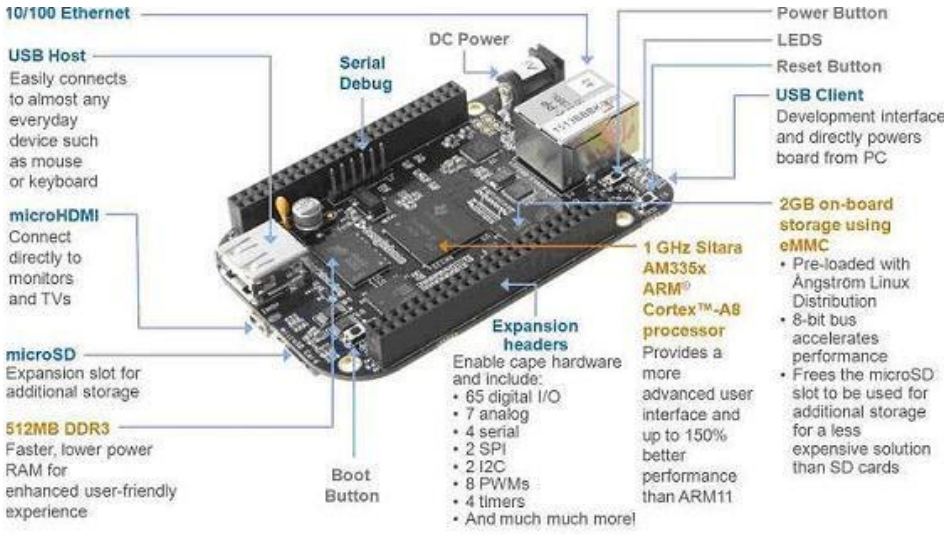


Ilustración 3.2 Descripción de la BeagleBone Black

Como se aprecia en la Ilustración 3.2, la BeagleBone Black también dispone de un puerto serie para depuración (Serial Debug en la imagen). Este puerto permite una conexión directa mediante consola con el dispositivo, lo que es de gran utilidad a la hora de desarrollar, ya que permite comprobar el estado real del dispositivo y qué procedimientos se están ejecutando.

De cara a la aplicación a desarrollar, es necesario identificar cómo acceder a los puertos I2C para la conexión del reloj. Para ello solo es necesario buscar las tablas con la descripción de las señales que transmiten.

Table 11. Expansion Header P9 Pinout

SIGNAL NAME	PIN	CONN		PIN	SIGNAL NAME
	GND	1	2	GND	
	VDD_3V3EXP	3	4	VDD_3V3EXP	
	VDD_5V	5	6	VDD_5V	
	SYS_5V	7	8	SYS_5V	
PWR_BUT*		9	10	A10	SYS_RESETn
UART4_RXD	T17	11	12	U18	GPIO1_28
UART4_TXD	U17	13	14	U14	EHRPWM1A
GPIO1_16	R13	15	16	T14	EHRPWM1B
I2C1_SCL	A16	17	18	B16	I2C1_SDA
I2C2_SCL	D17	19	20	D18	I2C2_SDA
UART2_TXD	B17	21	22	A17	UART2_RXD
GPIO1_17	V14	23	24	D15	UART1_TXD
GPIO3_21	A14	25	26	D16	UART1_RXD
GPIO3_19	C13	27	28	C12	SPI1_CS0
SPI1_D0	B13	29	30	D12	SPI1_D1
SPI1_SCLK	A13	31	32	VDD_ADC	
AIN4	C8	33	34	GNDA_ADC	
AIN6	A5	35	36	A5	AIN5
AIN2	B7	37	38	A7	AIN3
AIN0	B6	39	40	C7	AIN1
CLKOUT2	D14	41	42	C18	GPIO0_7
	GND	43	44	GND	
	GND	45	46	GND	

Ilustración 3.3 Modos de Funcionamiento Principales de los Pines de la Cabecera de Expansión 9 de la Beagle

Como se puede apreciar en la Ilustración 3.3, existen dos puertos I2C a los que el desarrollador tiene acceso a través de los pines 17, 18, 19 y 20 de la cabecera de expansión 9. (Ver Apéndice A: Documentación del Hardware, Anexo 1: Modos de Funcionamiento de los Pines de Entrada/Salida de la BeagleBone Black)

3.2 Arduino YUN

El Arduino YUN es una placa programable basada en dos procesadores, el ATmega32u4 y el Atheros AR9331. El procesador Atheros soporta un sistema operativo en base Linux llamado OpenWrt-Yun, una nueva versión diseñada para ejecutarse en placas Arduino. El procesador ATmega es un procesador Arduino típico usado para controlar todos los pines digitales de entrada/salida, entradas analógicas y señales moduladas por ancho de pulso (PWM).

(Ver Apéndice 1: Documentación del Hardware, Anexo B: Especificación de los Procesadores del Arduino YUN).

Su principal novedad en cuanto al hardware es la incorporación de un módulo capaz de generar una red Wi-Fi permitiendo cargar sketchs y el acceso al sistema operativo de forma remota. Esto permite cargar una aplicación web a la que se puede acceder inalámbricamente a través de la red Wi-Fi. El YUN también ofrece un host para conectar dispositivos USB, puerto Ethernet y una ranura para insertar tarjetas micro SD. El esquema de la distribución de estos módulos hardware puede apreciarse en la ilustración 3.4.

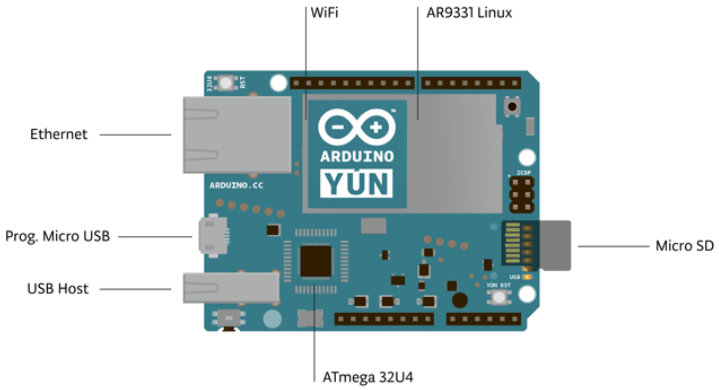


Ilustración 3.4 Diagrama Hardware del Arduino YUN

Al tener dos procesadores en una misma placa, cada uno controlando distintos recursos hardware, es necesario un método de comunicación entre ambos que permita pasar información y controlar de forma “remota” los recursos hardware que controla el otro. Para ello existe un puente entre ambos procesadores que permite una comunicación serie. De esta forma, y junto a las librerías proporcionadas por Arduino, el procesador Atheros puede acceder a los pines de entrada/salida y el ATmega a las funciones de red entre otras cosas. En la ilustración 3.5 puede verse el esquema de conexión del puente entre ambos procesadores.

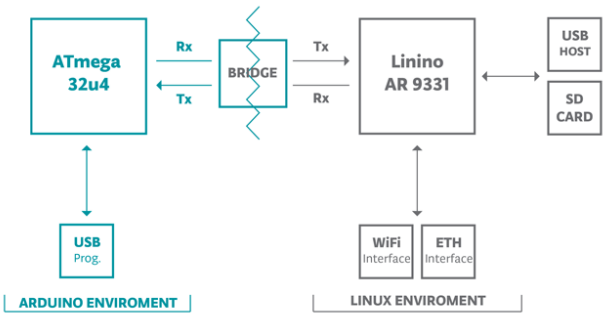


Ilustración 3.5 Esquema del Puente entre los Procesadores del Arduino YUN

Debido a que las necesidades de la interfaz de usuario solo requieren de las funciones de red y del almacenamiento en la tarjeta uSD, el uso del procesador ATmega y las funciones del puente entre procesadores solo son empleadas en la fase de configuración de la placa previa a la carga de la aplicación web.

3.3 RTC

Como dispositivo para controlar el tiempo dentro de la aplicación se ha elegido el reloj en tiempo real Chronodot DS3231, un módulo que, gracias a su batería, permite mantener la referencia de tiempo exacta a pesar de que el sistema entero caiga por cualquier circunstancia. (Ilustración 3.6)

El error de este reloj está por debajo del minuto al año. Esto es debido a que la temperatura del cristal interno, elemento con el que se hacen los cálculos de la fecha actual, está continuamente monitorizada y regulada gracias a unos capacitadores con el fin de mantener la frecuencia del cristal lo más estable posible.

Ilustración 3.6 Chronodot DS3231

Además de la los pines para la comunicación por I2C, el DS3231 tiene tres pines que ofrecen funcionalidades extra a la consulta de la fecha en sí, como el pin SQW, que permite configurar el reloj para que genere una señal cuadrada de 1000, 1024, 4096 o 8192 Hz o configurar una alarma para que genere un pulso cuando se alcance la fecha indicada.

Para la aplicación deberemos configurar el reloj con la fecha actual para que comience a calcular correctamente y lo usaremos para hacer las consultas de fecha a través del puerto I2C (Ver Apéndice A: Documentación Hardware, Anexo 3: DataSheet Chronodot DS3231, Registros Accesibles a Través de I2C)

3.4 Conexiones:

Para el esquema inicial solo son necesarias dos conexiones. En primer lugar está el cable Ethernet, que conectará la BeagleBone Black con el Arduino YUN. Para que la información pueda transmitirse correctamente será necesario configurar las interfaces de red de ambas placas para que se encuentren en la misma red. En segundo lugar es necesario conectar el reloj DS3231 a la BagleBone, para ello usaremos como soporte una protoboard. En este caso debemos conectar cuatro líneas entre el reloj y la BeagleBone; las líneas de datos y reloj (Cables amarillo y azul respectivamente en la Ilustración 3.7) del bus I2C más la línea de alimentación, que permite al reloj determinar el voltaje con el que debe enviar las señales por el bus (3.3V o 5V) y la referencia a tierra (Cables rojo y negro conectados a las referencias de voltaje y tierra de la BeagleBone).

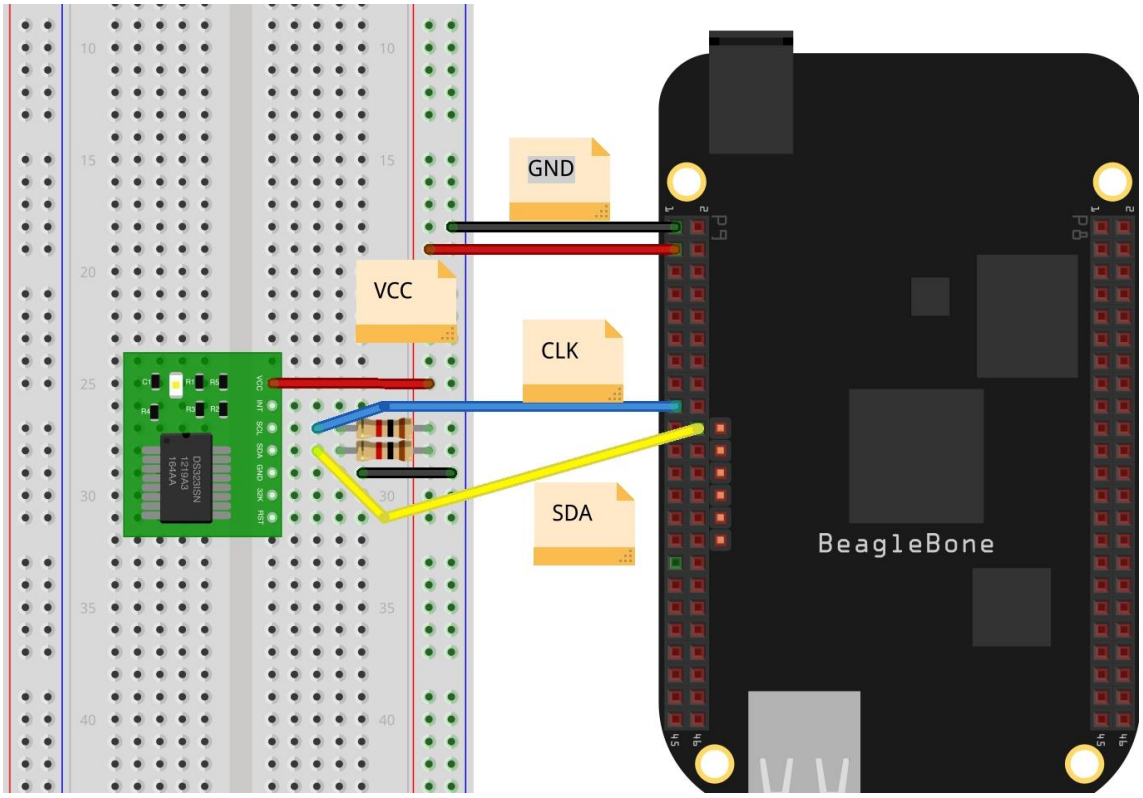


Ilustración 3.7 Diagrama de Conexiones entre el RTC y la BeagleBone BLack

Como se puede apreciar en el diagrama, es necesario incluir unas resistencias de Pull-Up en las conexiones del bus I2C para eliminar el ruido y que la BeagleBone sea capaz de leer la información del bus correctamente.

Una vez realizadas ambas conexiones el resultado del sistema completo quedaría de la siguiente manera como se puede apreciar en la Ilustración 3.8.

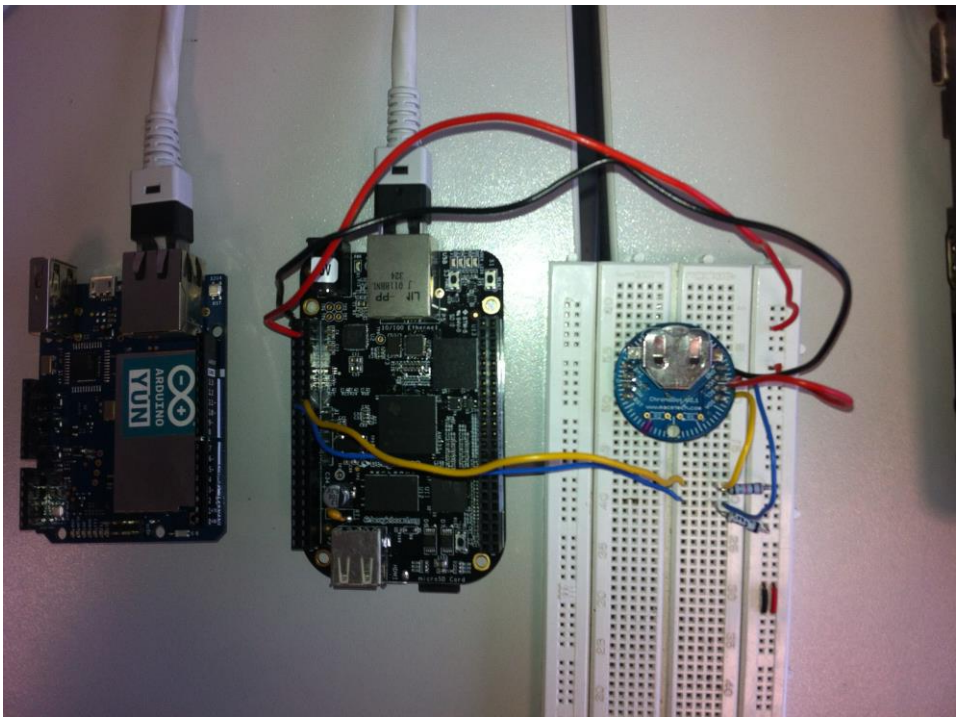


Ilustración 3.8 Conexiones para el Sistema Completo

4. El Sistema Operativo

Debido a las restricciones de tiempo impuestas para la aplicación, se ha optado por hacer uso de un sistema operativo en tiempo real que nos proporcione las herramientas necesarias para controlar los tiempos de ejecución del procedimiento de recogida de datos, mejorar la eficacia en la comunicación entre procedimientos y la detección de interrupciones hardware.

Tras un estudio de los principales sistemas operativos en tiempo real disponibles actualmente, centrando el foco de estudio en la compatibilidad con el hardware, se ha llegado a la conclusión de que el único sistema operativo completo y compatible es QNX.

Junto a QNX se analizaron otros tres sistemas operativos; FreeRTOS, RTLinux y ChibiOS.

- FreeRTOS: Es un sistema operativo en tiempo real de código abierto con una gran comunidad de desarrolladores implicados y compatible con numerosas arquitecturas hardware, entre ellas la arquitectura ARM7 de la BeagleBone Black. Sin embargo, la falta de un BSP que permitiese hacer uso de los recursos hardware de la placa hacía inviable esta alternativa como sistema operativo.

- RTLinux: RTLinux ofrece una aproximación a los sistemas operativos en tiempo real poco común. RTLinux es sólo un pequeño núcleo con un planificador. Sobre este núcleo se ejecuta la distribución de Linux que se desea como un proceso más. Esto permite evitar que el sistema operativo se ejecute debido a una mayor prioridad que las aplicaciones que el desarrollador ejecute.

  Debido a que sólo es compatible con procesadores i386 se descartó como posible sistema operativo para la BeagleBone Black.

- ChibiOS: ChibiOS es un pequeño núcleo creado para dar soporte a aplicaciones en tiempo real en sistemas empotrado. Ofrece todas las herramientas básicas para el desarrollo de aplicaciones en tiempo real y es compatible con la BeagleBone Black.

  La elección de QNX sobre ChibiOS se debió a que la compatibilidad con la placa es muy limitada. Sólo permite el uso de puerto serie e I2C, pero no es posible hacer uso de las funciones de red por puerto Ethernet o el almacenamiento de información en memoria a través de la ranura para tarjetas micro SD.

QNX es un sistema operativo en base Linux orientado para equipos empotrados. Se basa en un núcleo mínimo (microkernel) al que se le añaden distintos servicios que completan el sistema operativo como servicios de red, administrador de dispositivos, servidor web o base datos, etc. Esto permite al usuario crear un sistema operativo a medida y que ocupa poco espacio sin tener que modificar el propio núcleo del sistema.

La arquitectura del microkernel se divide en cuatro partes tal y como se muestra en la ilustración 4.1:

- IPC (Inter Process Communication): Se encarga de dirigir el paso de mensajes entre procesos y captar interrupciones software o enviar señales software a aquellos procesos encargados de manejarlas.

- Interfaz de red (Network Interface): Encargada de las comunicaciones de red a bajo nivel entre distintos nodos.

- Scheduler: Este módulo del microkernel es el encargado de decidir qué proceso debe estar en ejecución. El usuario puede indicar al sistema operativo el tipo de scheduling que desea entre FIFO, Round-Robin exclusivo y Round-Robin no exclusivo.

- Manejador de interrupciones de primer nivel (Interrupt Redirector): Encargado de captar las interrupciones hardware y redirigirlas a aquellos procesos que deban atenderlas.

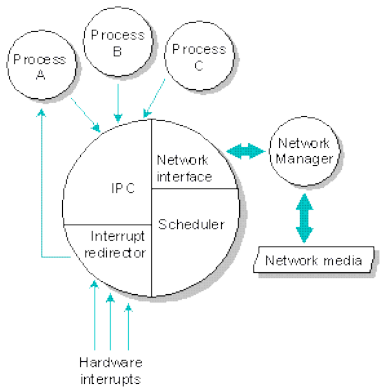


Ilustración 4.1 Esquema del Microkernel de QNX

Una capa por encima del núcleo se encuentran los servicios del sistema operativo ejecutándose como procesos comunes. La distribución de QNX que hemos utilizado en este proyecto tiene los siguientes procesos que se han añadido al núcleo mínimo tal y como se puede ver en la ilustración 4.2:

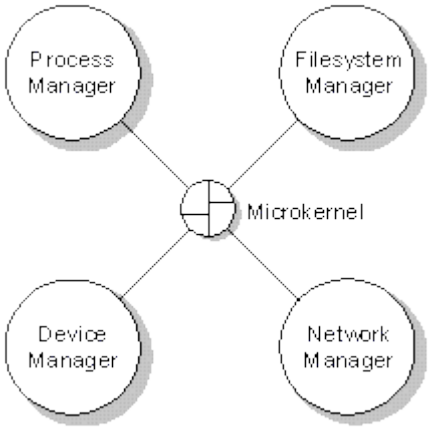


Ilustración 4.2 Esquema de los Servicios Estándar de QNX

- Administrador de Procesos (Process Manager): Este proceso trabaja conjuntamente con el Scheduler. Es el encargado de crear nuevos procesos y manejar los recursos esenciales de éstos.

- Administrador del Sistema de Ficheros (Filesystem Manager): El administrador de ficheros es el encargado de manejar la memoria de usuario permitiendo el manejo de todas las operaciones sobre ficheros.

- Administrador de Red (Network Manager): El administrador de red se comunica directamente con el microkernel y es el encargado de ampliar la potencia del paso de mensajes entre procesos de QNX hacia la red. Gracias a este administrador el desarrollador puede abstraerse de los detalles a bajo nivel de la red, ya que éste muestra los distintos nodos de la red como una extensión del sistema de ficheros. Esto permite acceder directamente a los datos de otros ordenadores QNX conectados a la red e incluso ejecutar aplicaciones de forma remota para trabajar usando programación distribuida.

- Administrador de Dispositivos (Device Manager): El administrador de dispositivos es el encargado de proporcionar una interfaz común entre los procesos del sistema y los dispositivos físicos conectados.

  En la ilustración 4.3 se puede apreciar la estructura de interfaces y el flujo de datos entre un proceso y el dispositivo con el que se desea comunicar tal y como se explica a continuación.

  El administrador de dispositivos (Device Manager) se comunica con los driver de cada uno de los dispositivos a través de una interfaz de drivers (Driver Interface). Esta interfaz se comunica con cada driver a través de zonas de memoria compartida donde se almacenan los datos de entrada y salida del dispositivo. El driver del dispositivo se encarga de trasladar la información contenida en estas zonas de memoria compartida al dispositivo correspondiente. De esta forma un proceso puede comunicarse con cualquier dispositivo conectado a través de un solo intermediario, el administrador de dispositivos.

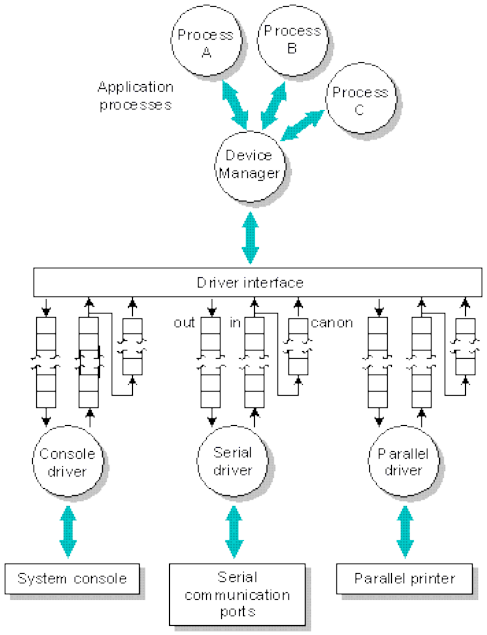


Ilustración 4.3 Esquema de Flujo de Información entre Procesos y Dispositivos

5. Herramientas de Desarrollo

En este apartado se procederá a explicar las herramientas de desarrollo empleadas para llevar a cabo este TFG.

5.1 BeagleBone Black

Para el desarrollo de la aplicación que se ejecuta en la BeagleBone Black se ha usado el IDE Momentics. Momentics IDE es propiedad de QNX y está hecho expresamente para desarrollar aplicaciones para este sistema operativo. Ofrece herramientas para depuración remota, comunicación por terminal con dispositivos conectados por red y manejo de BSP's (Board Support Systems). En la ilustración 5.1 se puede apreciar cómo es este entorno de desarrollo basado en Eclipse.

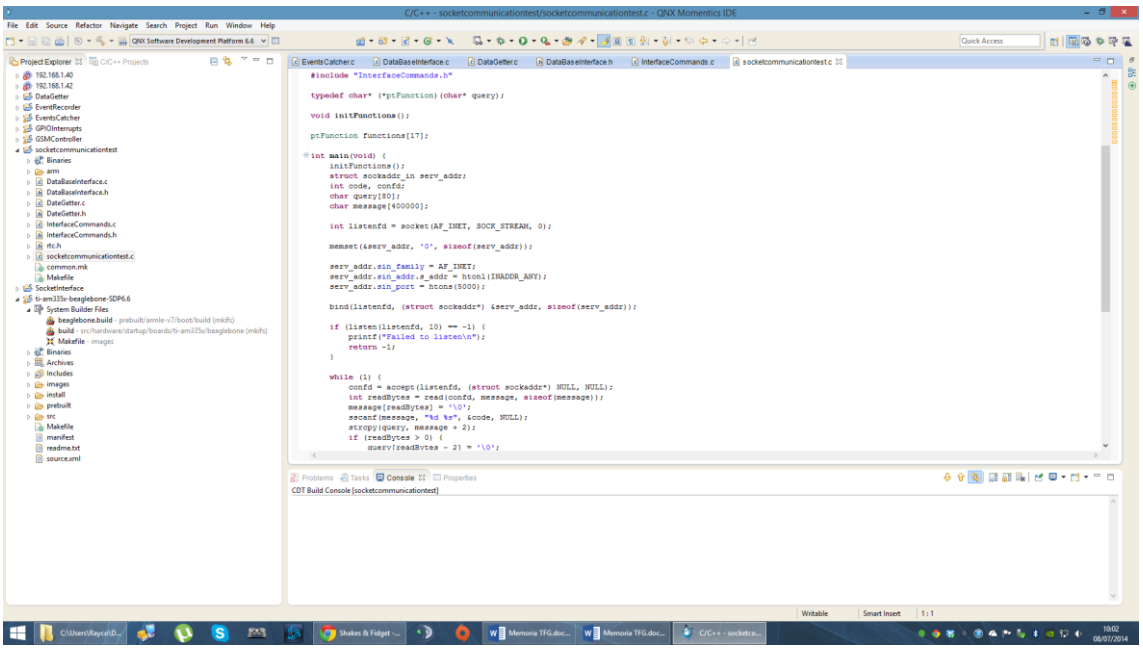


Ilustración 5.1 Momentics IDE

Para que la BeagleBone arranque con el sistema operativo QNX es necesario compilar el BSP desde el IDE Momentics e introducir la imagen del sistema operativo generada en una tarjea micro SD formateada. Cuando iniciamos la BeagleBone con dicha tarjeta insertada, el procesador reconoce que existe una imagen del sistema operativo en ella y ejecuta el sistema operativo. El código fuente del BSP puede verse dentro del directorio del CD "Código Fuente/BeagleBone/BSP"

Si se desea añadir funciones al BSP, como los binarios de la aplicación desarrollada, es necesario modificar el fichero beaglebone.build que se encuentra en el directorio System Builder Files del BSP (Ilustración 5.2)

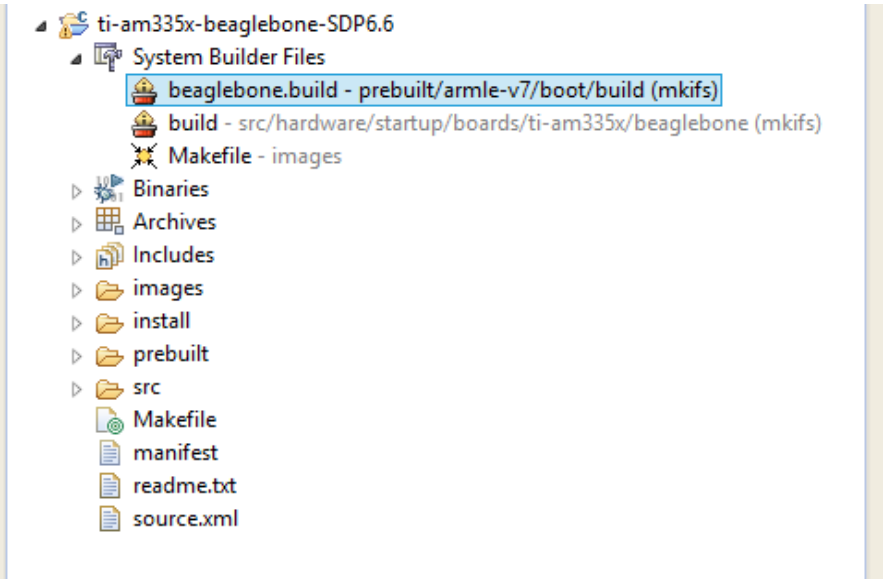


Ilustración 5.2 Sistema de Directorios del BSP

Este fichero contiene la información de qué librerías, binarios y ficheros de configuración se van a cargar en la imagen del sistema operativo que se genere. También contiene un script que se ejecuta al arrancar el sistema identificado con la etiqueta "[+script]". Esto es de gran utilidad, ya que nos permite ejecutar la aplicación automáticamente con el arranque del sistema sin tener que acceder al dispositivo cada vez que éste se enciende. A continuación, en la Ilustración 5.3, se puede observar un ejemplo de cómo se declaran los ficheros que se desea que se copien en la imagen del sistema operativo.

```
#Declaración para copiar ficheros desde el sistema de ficheros del IDE al BSP

[data=copy]

#A la izquierda se pone la ruta en la que se desea que se copie. A la deracha el fihero fuente

/bin/ds = ds #Binario

#Librerías
/lib/libqdb.a = libqdb.a
/lib/libqdb.so = libqdb.so
/lib/libqdb.so.1 = libqdb.so.1

#Binario creado por el desarrollador. WORKSPACE es una variable de entorno que indica la ruta al workspace del IDE
/bin/corrupteddb = WORKSPACE/corrupteddb/arm/o-le-v7/corrupteddb

#Fichero de configuración
[data=uip] /etc/syslog.conf = syslog.conf
```

Ilustración 5. 3 Ejemplo de Adición de Elementos al Fichero beaglebone.build

5.2 Arduino YUN:

Para el desarrollo de la aplicación web se ha decidido emplear el lenguaje PHP. Esto se debe a que al hacer uso de la red Wi-Fi que genera el YUN y que aprovechamos como método de acceso inalámbrico a la interfaz, el mejor método para crear la interfaz de usuario es mediante una aplicación web. PHP también proporciona funciones de comunicación socket para la transmisión de datos con el sistema central.

Como herramienta para el desarrollo de la aplicación web se usa el IDE NetBeans con el plugin de PHP. NetBeans es un IDE desarrollado en Java con herramientas de programación avanzada muy potentes, como funciones de refactorización del código automatizadas. En la siguiente imagen (Ilustración 5.4) se puede ver la interfaz de NetBeans con el proyecto de la aplicación web abierto.

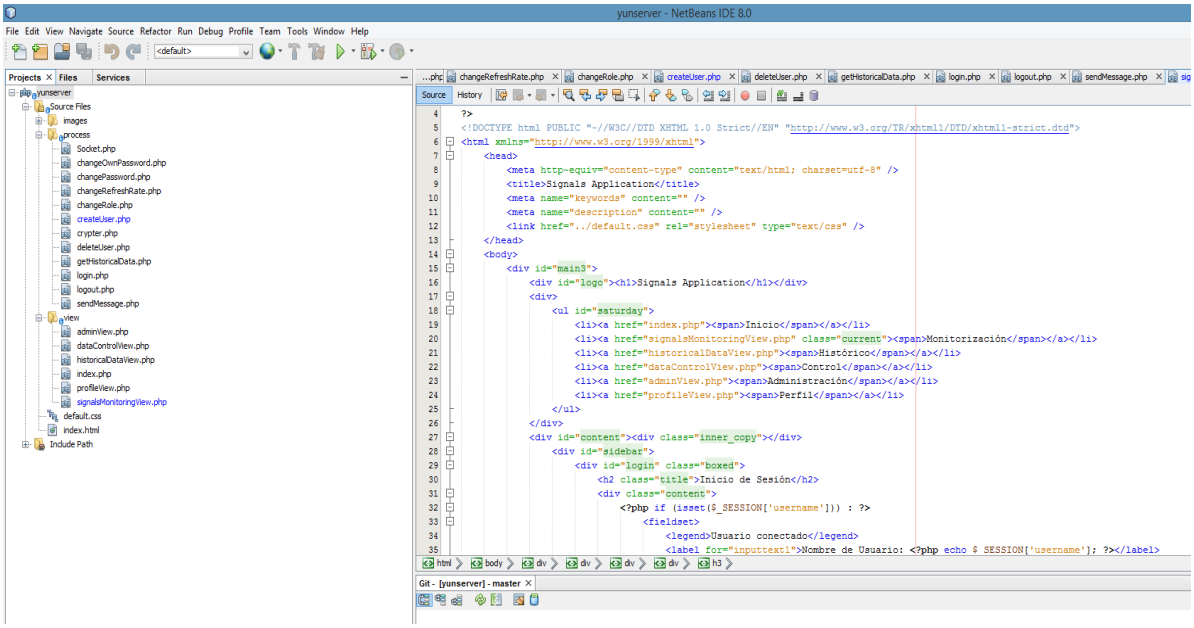


Ilustración 5.4 NetBeans IDE

Para cargar la aplicación web en el Arduino YUN es necesario un segundo IDE, el Arduino IDE 1.5. Este IDE nos permitirá cargar la aplicación en el Arduino YUN de forma inalámbrica a través de su red Wi-Fi.

En primer lugar es necesario abrir el Arduino IDE y crear un nuevo sketch. Dentro del directorio del nuevo sketch se debe crear una carpeta con el nombre “www” que contenga la aplicación web tal y como se muestra en la Ilustración 5.5.

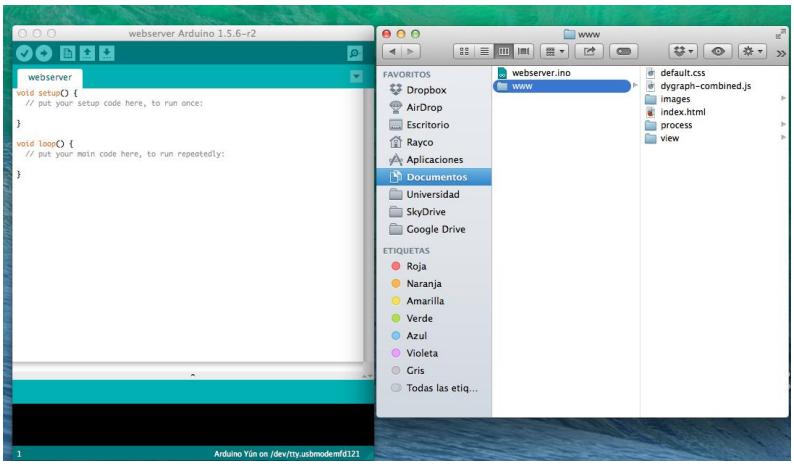


Ilustración 5.5 Sketch con Aplicación Web en Arduino YUN

Una vez hecho esto, y si nuestro ordenador se encuentra conectado a la red Wi-Fi generada por el YUN, podremos cargar la aplicación tan solo mandando cargar el sketch al Arduino mediante el puerto con la dirección Wi-Fi. Como se puede ver en la ilustración 5.6, el IDE ha detectado automáticamente que existe un Arduino YUN conectado a la red Wi-Fi y lo indica en la parte inferior de la pantalla (Recuadro rojo).

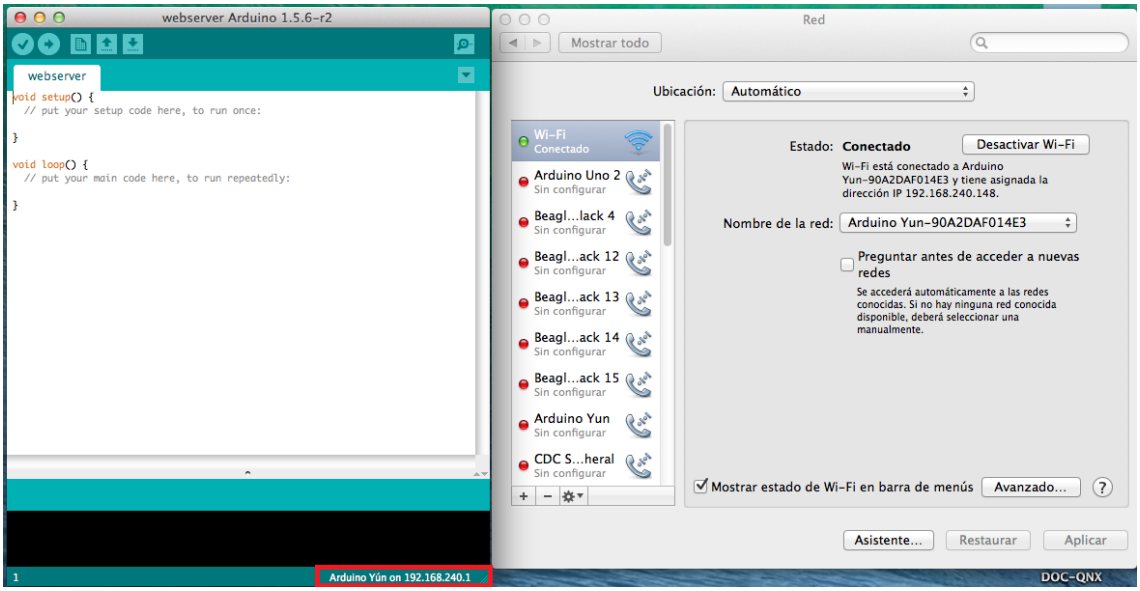


Ilustración 5.6 Carga de la Aplicación Web en Arduino YUN

Sin embargo, para que la aplicación web funcione correctamente en el YUN es necesario realizar una serie de configuraciones previas.

En primer lugar, debido a la poca memoria interna de la placa Arduino YUN, es necesario ejecutar un sketch que no permite ampliar el espacio del sistema de ficheros a través de una tarjeta microSD conectada. Este sketch puede encontrarse en el apartado del CD "Código Fuente/YUN/yunexpander/"

Una vez ejecutado el sketch y seguido las instrucciones que aparecen en pantalla a través del monitor serie podremos disponer del espacio necesario para el resto de configuraciones.

Lo siguiente que debemos hacer es conectarnos por ssh al arduino yun ejecutando el siguiente comando:

ssh root@arduino.local

Si nuestro ordenador se encuentra conectado a la red Wi-Fi del YUN, esta instrucción nos permitirá acceder por consola al sistema operativo Linux del YUN para realizar el resto de configuraciones tal y como se muestra en la ilustración 5.7.

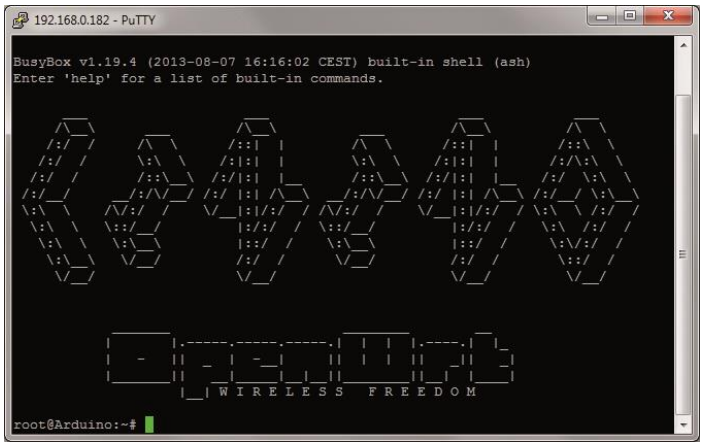


Ilustración 5.7 Acceso por SSH al Arduino YUN

Una vez tenemos acceso al sistema operativo debemos instalar los paquetes de PHP y los módulos de ejecución de código en páginas web y de comunicación socket (php5 y php5-mod-sockets respectivamente) mediante el comando opkg install.

Tras la instalación de los paquetes indicados solo queda configurar el servidor web, uhttpd por defecto, para que ejecute el código PHP de las páginas web que muestre. Para ello es necesario modificar el fichero “uhttpd” ubicado en el directorio “/etc/config/”. El fiechero debe incluir lo siguiente (Ilustración 5.8).

```
        # If no listen_https addresses are given,
        # the key options are ignored.
        option cert             /etc/uhttpd.crt
        option key              /etc/uhttpd.key

        # CGI url prefix, will be searched in docroot.
        # Default is /cgi-bin
        option cgi_prefix       /cgi-bin

        # List of extension->interpreter mappings.
        # Files with an associated interpreter can
        # be called outside of the CGI prefix and do
        # not need to be executable.
        list interpreter        ".php=/usr/bin/php-cgi"
#       list interpreter        ".cgi=/usr/bin/perl"

        # Lua url prefix and handler script.
        # Lua support is disabled if no prefix given.
#       option lua_prefix       /luci
#       option lua_handler      /usr/lib/lua/luci/sgi/uhttpd.lua

        # CGI/Lua timeout, if the called script does not
        # write data within the given amount of seconds,
        # the server will terminate the request with
```

Ilustración 5.8 Fichero de Configuración del Servidor Web

Por último se deben habilitar la extensión de sockets en el fichero de configuración de PHP “/etc/php.ini” tal y como se muestra a continuación (Ilustración 5.9).

```
;extension=hash.so
;extension=iconv.so
;extension=json.so
;extension=ldap.so
;extension=mbstring.so
;extension=mcrypt.so
;extension=mysql.so
;extension=openssl.so
;extension=pcre.so
;extension=pdo.so
;extension=pdo-mysql.so
;extension=pdo-pgsql.so
;extension=pdo_sqlite.so
;extension=pgsql.so
extension=session.so
;extension=soap.so
extension=sockets.so
;extension=sqlite.so
;extension=sqlite3.so
;extension=tokenizer.so
;extension=xml.so
;extension=xmlreader.so
;extension=xmlwriter.so
```

Ilustración 5.9 Fichero de Configuración de PHP

Con esto las configuraciones correspondientes al sistema de ficheros, el servidor web, y los paquetes de PHP del Arduino YUN quedan finalizadas.

6. Metodología de Desarrollo

En este apartado se explicará la metodología de desarrollo empleada durante la realización de este TFG y las motivaciones que llevaron a la elección de la misma.

Debido a la alta complejidad de la aplicación a causa del control en tiempo real de los procesos y las comunicaciones entre dispositivos, se creyó conveniente desarrollar la aplicación en módulos independientes los unos de los otros. Esto permite probar cada parte de la aplicación a medida que se va desarrollando. Además, esta estrategia permite aislar las fuentes de errores por módulos, acortando el tiempo que conlleva la depuración de la aplicación y, por tanto, también el tiempo de desarrollo en general.

Como estrategia de desarrollo de los distintos módulos se busca construir cada uno de ellos mediante un método de desarrollo iterativo e incremental. Esto permite comenzar con un módulo con funcionalidad limitada, pero sencillo de probar. Una vez desarrollada esta primera versión, se puede ir ampliando la funcionalidad en iteraciones con la seguridad de que, en caso de fallo, será fácilmente localizable ya que estará en los cambios de la última iteración realizada.

Una vez establecidas estas dos estrategias de desarrollo, el trabajo que se ha realizado en este TFG ha seguido el ciclo de vida del software que se divides en las siguientes etapas: análisis, diseño, implementación, y verificación (ver Ilustración 6.1). El primer paso para llevar a cabo este método de desarrollo será realizar un análisis de los requisitos de la aplicación, buscando detectar módulos independientes fácilmente verificables. Una vez realizado el análisis, las fases de diseño, implementación y verificación realizan en iteraciones para cada módulo. Al terminar todas las implementaciones, se realizará una verificación final del sistema integrado. Es prueba final será la propia interfaz de usuario.

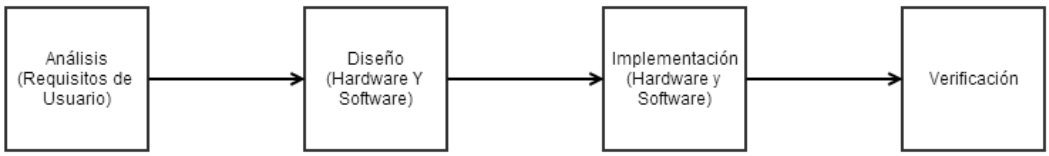


Ilustración 6.1 Fases de Desarrollo del TFG

7. Análisis

En este apartado se muestran agrupados todos los requisitos funcionales de la aplicación descrita en los objetivos funcionales junto a aquellos no funcionales que han ido surgiendo durante el análisis inicial previo del hardware a usar y las herramientas de desarrollo.

Los requisitos funcionales de la aplicación son los siguientes:

- La aplicación debe obtener los datos de las señales físicas del sistema periódicamente y almacenar la información junto a la fecha de obtención.
- La aplicación será capaz de detectar eventos en forma de interrupciones hardware y almacenar dicho evento junto a la fecha en la que se produjo.
- La aplicación debe poder comunicarse con otros dispositivos sockets para recoger los datos registrados y cambiar parámetros de configuración.
- El usuario podrá visualizar los datos a través de una interfaz gráfica
- El usuario tendrá la capacidad de configurar el periodo de obtención de datos
- Se le permitirá al usuario modificar los valores de las señales físicas de salida a través de la interfaz gráfica de usuario.

Como se puede apreciar a partir de los requisitos funcionales de la aplicación, existen cuatro módulos independientes que trabajan conjuntamente para generar la aplicación completa. El primer módulo es el encargado de la obtención y almacenamiento de los datos. El segundo es el encargado de la detección y almacenamiento de los eventos del sistema. El tercero tiene como comedido el ofrecer una interfaz a través de comunicación socket para que otros dispositivos puedan acceder a los datos y configurar los parámetros de funcionamiento. Estos tres módulos son el núcleo de la aplicación y se encuentran alojados en el sistema central, la BeagleBone Black. El último módulo es la interfaz gráfica de usuario, que se ejecuta en el Arduino YUN y se conecta al sistema central a través del puerto Ethernet.

Por último, los requisitos no funcionales son los siguientes:

- La aplicación no debe consumir CPU durante los tiempos de espera entre recogidas de datos.
- El tiempo entre la generación de una interrupción y la detección del evento por parte de la aplicación debe ser mínimo.
- El acceso a la visualización de datos y configuración de los parámetros de funcionamiento desde la interfaz gráfica debe estar restringido por un control de usuario y un sistema de roles.

8. Persistencia de Datos

Debido a la necesidad de almacenar los datos que se generen de forma permanente, se ha optado por hacer uso de la base de datos propia del sistema operativo QNX. Esta base de datos está basada en SQLite3 y nos permite mantener una estructura de datos común para todos los módulos dentro del sistema central.

En primer lugar es necesario añadir al BSP de QNX para la BeagleBone Black las librerías y binarios necesarios para para hacer uso de la base de datos siguiendo el procedimiento descrito en el apartado 5.1. Las librerías y binarios requeridos se muestran a continuación (Ilustración 8.1).

```
[data=copy]

#Binarios para la base de datos
/bin/qdbc = qdbc
/bin/qdb = qdb

#Librerías para la base de datos
/lib/libqdb.a = libqdb.a
/lib/libqdb.so = libqdb.so
/lib/libqdb.so.1 = libqdb.so.1
/lib/libsqlite3.so.1 = libsqlite3.so.1
/lib/libz.so.2 = libz.so.2
/usr/lib/libsqlite3.so = libsqlite3.so
/usr/lib/libsqlite3.so.1 = libsqlite3.so.1
/proc/boot/libcpp.so.4 = libcpp.so.4


#Declaración para copiar ficheros desde el sistema de ficheros del IDE al BSP
```

Ilustración 8.1 Ficheros Necesarios para el Uso de la Base de Datos en QNX

También se añaden una serie de comandos al script que se encuentra en el fichero beaglebone.build para que el servidor de la base de datos arranque con el sistema (ver Ilustración 8.2).

```
#Configuración inicial de la base de datos
TMPDIR=/fs/sd0/bbdd/temp
QDBC_DBNAME=/fs/sd0/bbdd/database
HOME=/
PATH=:/proc/boot:/bin:/usr/bin:/opt/bin:/usr/photon/bin
LD_LIBRARY_PATH=:/proc/boot:/lib:/usr/lib:/lib/dll:/opt/lib

#Arranque del servidor de la base de datos
pps -m /fs/sd0/bbdd/ &
qdb -A -c /fs/sd0/bbdd/dbconfigfile.cfg -D -I full -n /fs/sd0/bbdd -N mymsdcontrol
```

Ilustración 8.2 Comandos para el Inicio del Servidor de la Base de Datos

La primera parte del script es una configuración inicial de las variables de entorno que utilizará la base de datos. Las variables de entorno específicas para su funcionamiento son TMPDIR, que indica a la base de datos el directorio habilitado para almacenar ficheros temporales. La otra variable de entorno importante es QDBC_DBNAME, que indica la localización del fichero que debe emplearse para acceder a la base de datos.

A continuación, en los Apartado 8.1, 8.2, 8.3 y 8.4, se muestra el diseño Entidad-Interrelación empleado para generar la base de datos. La explicación del medelo se ha dividido en cuatro subapartados para mejorar la visibilidad y comprensión de los mismos. La división se ha realizado por función dentro de la aplicación, siendo el apartado 8.1 (Mapeo de Señales Físicas) el diseño del modelo de datos relevante para la captura de datos. El apartado 8.2, Eventos, muestra la estructura de datos diseñada para usar en el módulo de detección de eventos. El apartado 8.3, Registro de Usuarios, permite ver cómo se almacenarán los datos de los usuarios registrados en el sistema. Por último, el apartado 8.4, Periodo entre Lecturas, muestra el diseño que permite almacenar los datos de configuración del módulo de captura de datos.

También puede consultarse el guion SQL de la base de datos en el directorio del CD “Código Fuente/Apéndice 1”.

8.1 Mapeo de Señales Físicas:

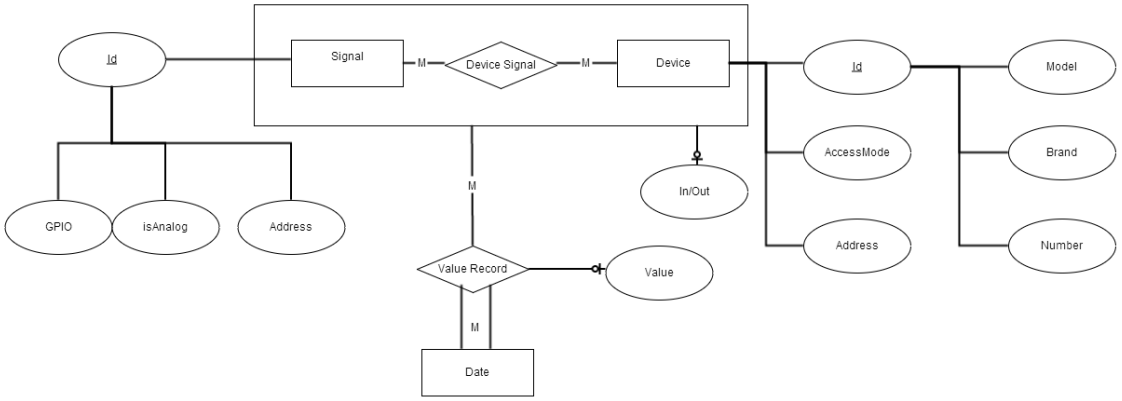


Ilustración 8.3 Modelo Entidad-Interrelación del Mapeo de las Señales Físicas

Con el objetivo de crear una arquitectura software potente capaz de ofrecer mayor funcionalidad en el futuro se ha optado por usar una serie de entidades para identificar los dispositivos y señales físicas de la siguiente forma. (Ver ilustración 8.3)

- Dispositivo (Device): Un dispositivo es cualquier periférico hardware conectado al sistema central del que se pueden obtener datos. El dispositivo contiene un identificador, el método de acceso al mismo (I2C, ModBus,…), y la dirección para llegar hasta él.
  Dentro de esta entidad la BeagleBone Black es otro dispositivo al que se puede acceder para obtener datos. La única diferencia es que el modo de acceso será local y la dirección no se usará. Éste será el único dispositivo del que obtendremos datos en la aplicación inicial.

- Señal física (Signal): La señal es el mapeo en la base de datos de la información para obtener los valores de una señal física. Contiene información que indica si es una señal digital o analógica y si es de entrada o salida. Además, la señal contiene un último campo que indica su dirección dentro de un dispositivo. Es dirección se usa junto a la dirección del dispositivo para acceder a su valor.
  Por ejemplo, si tuviésemos un dispositivo I2C conectado del que queremos obtener el valor de una de sus señales, la dirección del dispositivo nos diría la dirección del dispositivo I2C al que le tendríamos que enviar la orden de lectura. La dirección indicada por la señal sería la que nos indicaría en qué posición de memoria accesible por I2C se encuentra mapeado el valor de la señal.

- Histórico de Valores (Value Record): Por último cuando obtenemos el valor de una señal se almacena en esta tabla, donde se identifica al dispositivo y señal del que se leyó el valor y la fecha de la lectura.

8.2 Eventos:

Los eventos son sucesos anómalos que son notificados por un dispositivo externo al sistema central o por un procedimiento del propio sistema. Cada evento se distingue por su identificador y tiene un mensaje descriptivo del evento.

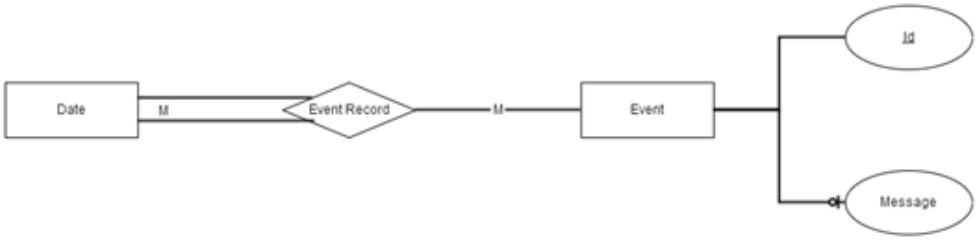


Ilustración 8.4 Modelo Entidad-Interrelación de los Eventos

Cuando un evento se produce se guarda en la base de datos junto a la fecha en la que se produjo para mantener un histórico de los eventos que se han generado a lo largo del tiempo. Ver Ilustración 8.4.

8.3 Registro de Usuarios:

La última parte de la base de datos es el registro de usuarios. Cada usuario con acceso a la interfaz gráfica estará registrado aquí con un nombre de usuario, su contraseña encriptada y el rol que posee (Monitorización, Control o Administración). El diagrama con el modelo para el registro de usuarios puede en la ilustración 8.5.

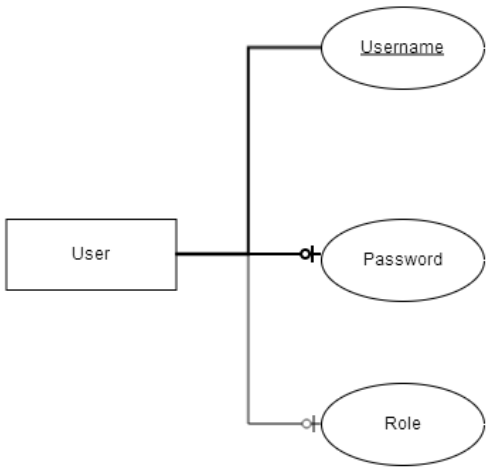


Ilustración 8.5 Modelo Entidad-Interrelación del Registro de Usuarios

8.4 Periodo entre Lecturas:

El periodo entre lecturas es el tiempo que la aplicación debe esperar cada vez que realiza la consulta de valores a las señales físicas. Se guarda en la base de datos para que la aplicación pueda saber el periodo con el que debe trabajar al inicio de la ejecución. El diseño es el que se muestra en la Ilustración 8.6.

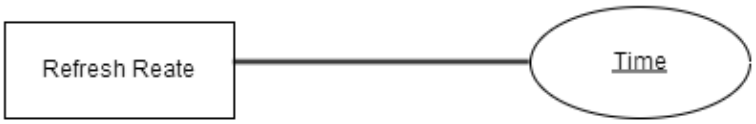


Ilustración 8.6 Modelo Entidad-Interrelación del Parámetro de Configuración del Módulo de Captura de Datos

9. Procedimientos del Sistema Central

En la fase de análisis se establecieron tres módulos de la aplicación que residían en el sistema central y que trabajaban conjuntamente para ofrecer al usuario los datos obtenidos y una interfaz para configurar el funcionamiento de la aplicación. En este apartado se mostrará el diseño e implementación de estos módulos en forma de procedimientos separados junto a su verificación.

9.1 Obtención de Datos de Forma Periódica:

Este procedimiento es el encargado de obtener los datos a partir de las distintas señales físicas y almacenar dichos datos con una marca temporal.

Además, este procedimiento deberá ejecutar la obtención de datos periódicamente con un tiempo fijado por el usuario y no consumir la CPU durante las esperas. También deberá hacer uso de la base de datos para conocer las señales físicas que debe consultar y almacenar los valores obtenidos de las mismas.

- **Diseño:**

Este procedimiento debe seguir el siguiente orden de ejecución, tal y como se especifica en la ilustración 9.1.

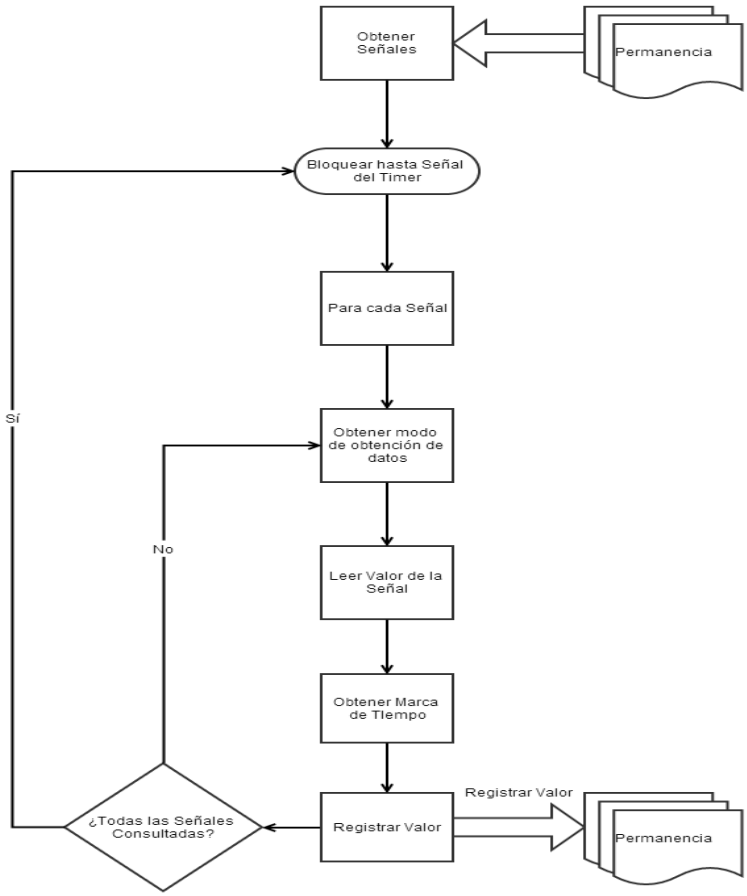


Ilustración 9.1 Diagrama de Flujo del Procedimiento de Obtención de Datos

- Obtener señales: En este paso el procedimiento debe consultar a la base de datos la información sobre las señales físicas que hay disponibles para consultar.
- Bloquear hasta Señal del Timer: El procedimiento debe bloquearse sin consumir CPU hasta que sea la hora de obtener los datos de las señales nuevamente.
- Bucle “Para Cada Señal”: Se entra en un bucle en el que en cada iteración tendremos la información de una señal física distinta.
- Obtener modo de obtención de datos: En este paso el procedimiento deberá consultar a qué dispositivo debe acceder para obtener el valor de la señal física, el método de acceso al dispositivo (Local, I2C, ModBus,…), la dirección en la que se encuentra y la dirección en la que se encuentra el valor actual de la señal física dentro del dispositivo.
- Leer Valor de la Señal: Con toda la información obtenida en el paso anterior, el procedimiento procede a consultar el valor de la señal. En el caso de la aplicación inicial, el único método de lectura de señales físicas será el de lectura de los propios pines digitales de la BeagleBone Black.

- Obtener Marca de Tiempo: El procedimiento debe consultar al reloj electrónico (RTC) la fecha actual y pasarla a un formato reconocible por el sistema.
- Registrar Valor: El procedimiento debe almacenar en la base de datos el valor de la señal física leída junto a la marca de tiempo.
- ¿Todas las Señales Consultadas?: EL procedimiento se mantiene en un bucle hasta que haya leído el valor de todas las señales físicas. Cuando termina, se bloquea hasta el próximo momento en el que sea necesaria la lectura.

- **Implementación:**

Para controlar el periodo de tiempo entre lecturas del procedimiento se hará uso del timer de QNX. Este timer es un recurso del sistema operativo que permite al programador fijar el momento en que desea que el timer avise a un procedimiento, e incluso si desea que lo haga de forma periódica. El sistema operativo envía una señal software asíncrona cuando el timer lo indica al procedimiento que lo activó. De esta manera nuestro procedimiento puede bloquearse cuando no tenga que consumir CPU y puede volver a la ejecución cuando sea necesario, mejorando así el rendimiento general de la aplicación.

Si se desea hacer uso de los timers del sistema operativo QNX, es necesario rellenar una estructura de datos que indica al timer dentro de cuánto tiempo en nanosegundos debe avisar al procedimiento y, una vez avisado la primera vez y de manera opcional, con qué frecuencia lo volverá a hacer. En nuestro caso ambos valores para el timer son el mismo, el periodo que el usuario nos indique. Una vez hemos inicializada la estructura solo debemos activar el timer indicando la señal software que debe mandarnos el sistema operativo cuando el timer lo indique.

Las señales software que maneja QNX son las mismas que el estándar POSIX, y la forma de asociar un manejador de señal software a nuestro procedimiento también sigue este estándar.

Con el objetivo de mejorar la modularidad del procedimiento se ha separado en una función de librería el método para la obtención de la fecha a través del reloj DS3231, lo que permite reutilizar la función en otros procedimientos que lo requieran. (Ver en CD: “Código Fuente/Apéndice 2”)

El código fuente de este procedimiento puede consultarse en el CD dentro del directorio “Código Fuente/BeagleBone/Obtención de Datos”

- **Criterio de Verificación:**

Como criterio de verificación se utiliza la información almacenada en la base de datos. Si el procedimiento está generando en la base de datos el histórico de las señales físicas con el valor y la fecha correcta, la iteración dedicada a la implementación del procedimiento puede darse por finalizada.

9.2 Detección de Eventos:

Este procedimiento se encarga de detectar rápidamente eventos ocurridos en los dispositivos conectados al sistema central. Pare ello se hará uso de una línea de interrupciones dedicada para cada dispositivo. Cuando una interrupción se genera, el procedimiento pude distinguir por la línea de interrupción activada qué dispositivo ha generado el evento y almacenar la información.

- **Diseño:**

El procedimiento de detección de eventos debe seguir el siguiente esquema de comportamiento (El esquema descrito puede encontrarse en la ilustración 9.2).

- Configurar Interrupción: Cuando se inicia el procedimiento, éste debe configurar la línea de interrupción a la que va a atender para que se active cuando un dispositivo acitva el pin de entrada asociado a la interrupción.
- Marcar Procedimiento como Manejador de la Línea de Interrupción: Una vez configurada la línea de interrupción, el procedimiento debe indicar al sistema operativo que debe avisarlo cuando esta línea de interrupción se active.
- Bloquear hasta Interrupción: El procedmiento debe quedarse en estado de bloqueo sin cunsumir CPU hasta que el sistema opartivo avise de que la línea de interrupción se ha activado.
- Obtener Marca de Tiempo: Una vez el procedimiento se despierta debe atender la interrupción. Para ello lo primero que debe hacer es obtener la fecha actual del reloj electrónico (RTC).
- Registrar Evento: Por último, el procedimiento almacena en la base de datos el código de evento relacionado con la interrupción activada y la fecha obtenida. Una vez almacenada la información, el procedimiento vuelve bloquearse hasta la próxima interrupción.

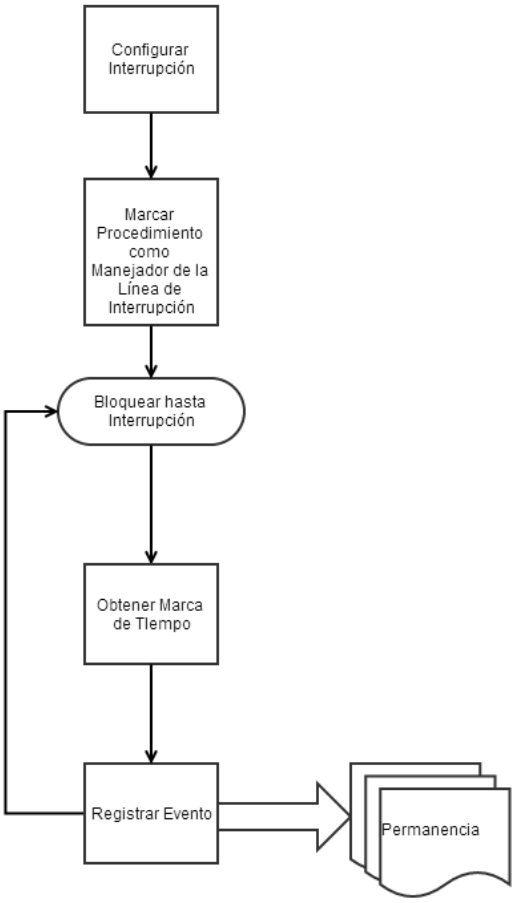


Ilustración 9.2 Diagrama de Flujo del Procedimiento de Detección de Eventos

- **Implementación:**

A nivel de implementación este procedimiento entraña una dificultad elevada debido a la necesidad de comprender cómo la BeagleBone maneja las interrupciones provenientes de los GPIO y cómo fluye la información desde el momento en el que el GPIO genera una interrupción hasta que el procedimiento de la aplicación es notificado de ello.

Debido a las limitaciones de los drivers del BSP de QNX para la BeagleBone Black, el control de los GPIO, la generación de interrupciones y el enmascaramiento de las mismas debe realizarse utilizando el mapeo de la entradas/salida en memoria directamente.

En el esquema de la ilustración 9.3 se muestra cómo es el flujo de información y los registros implicados desde que se produce un cambio en un GPIO hasta que su línea de petición de interrupción asociada se activa.

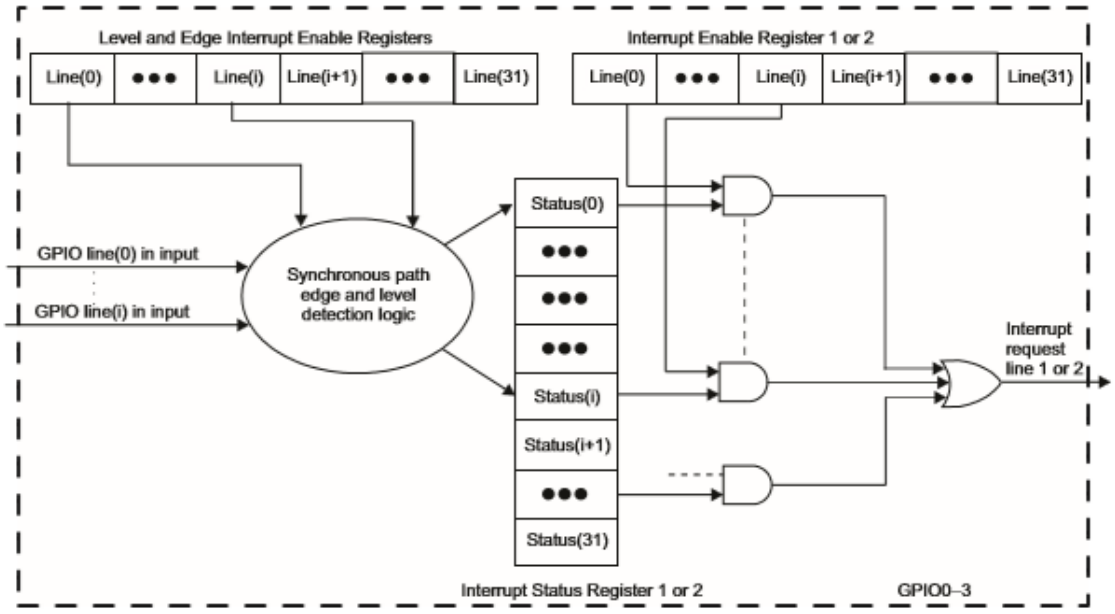


Ilustración 9.3 Esquema de Interacción de las Zonas de Memoria relacionadas con las Interrupciones GPIO

El primer paso es el de configurar el pin de digital (GPIO) que deseamos para que genere un tipo de interrupción, ya sea por nivel alto/bajo o por flanco de subida/bajada ("Level and Edge Interrupt Enable Registers" en la ilustración). El procesador usa una lógica interna donde usa los valores de los GPIO (GPIO line in input) y el valor de las direcciones de memoria donde se encuentran mapeados estos registros para determinar si un GPIO ha generado una interrupción. El resultado lo almacena en otra dirección de memoria asociada al registro de estado de interrupciones (Interrupt Status Register).

Posteriormente, el mencionado registro de estado de interrupciones se analiza para comprobar si alguna de las interrupciones que se ha activado está habilitada. Para ello el procesador consulta la zona de memoria donde se almacena el registro de habilitación de interrupciones (Interrupt Enable Register). Si alguna de las interrupciones que se activaron está habilitada, se activa una línea de petición de interrupción.

Para que la línea de petición de interrupción se convierta en una interrupción hardware visible para el sistema operativo se debe pasar por un último control, el enmascaramiento de las interrupciones. Así que para que nuestra interrupción llegue al sistema operativo es necesario modificar la zona de memoria donde están mapeadas las máscaras de las distintas líneas de interrupción para desenmascarar aquellas que deseemos.

Añadiendo todos estos detalles de implementación al diseño, el esquema de comportamiento quedaría tal y como se muestra en la ilustración 9.4.

- Configurar el GPIO en Pull-Up: El sistema de detección de interrupciones en la BeagleBone Black funciona mediante lógica positiva, por lo que un uno lógico implica que hay tensión en el circuito y un cero lógico que no existe tensión en el mismo. Por ello debemos configurar el GPIO para que funcione de la misma forma.
- Habilitar la Petición de Interrupciones para el GPIO: Para la interrupción llegue a la zona de enmascaramiento de interrupciones es necesario que el bit asociado con el GPIO de lo zona de memoria denominada “Interrupt Enable Register” esté activo. De esta manera, cuando el bit de nuestro GPIO en el “Interrupt Status Register” esté activo, la petición de interrupción se activará.
- Configuración del Modo de Activación de la Interrupción: Cada GPIO tiene asociadas zonas de memoria donde se configura en qué condiciones dicho GPIO generará una interrupción hardware. Estas zonas de memoria se denominan Low/High Level Interrupt Enable Register y Rising/Falling Interrupt Enable Register. En este caso controlaremos los eventos por flanco de subida. De este modo, cada vez que el GPIO pase de cero a uno, generará una interrupción.
- Desenmascarar la Línea de Interrupción Asociada al GPIO: Para que el sistema operativo se entere de que existe una interrupción hardware pendiente de ser atendida es necesario que dicha interrupción esté desenmascarada. Para ello solo es necesario poner a 0 el bit asociado a la línea de interrupción que queremos desenmascarar en la zona de memoria que se identifica como “Mask Interrupt Register”
- Marcar Procedimiento como Manejador de la Línea de Interrupción: Una vez configurada la línea de interrupción, el procedimiento debe indicar al sistema operativo que debe avisarlo cuando esta línea de interrupción se active. Para ello se hace uso de las funciones especificas para menjo de interrupciones hardware de QNX.
- Bloquear hasta Interrupción: El procedmiento debe quedarse en estado de bloqueo sin cunsumir CPU hasta que el sistema opartivo avise de que la línea de interrupción se ha activado.
- Limpiar la Fuente de Interrupción: Para que no se genere una interrupción contínua, se debe volver a poner a cero el bit del GPIO que generó la interrupción en el “Interrupt Status Register”.
- Desenmascarar la Línea de Interrupciones: Cada vez que se genera una interrupción, el sistema operativo la enmascara. Es responsabilidad del manejador de dicha interrupción desenmascararla una vez ha sido atendida.
- Obtener Marca de Tiempo: Una vez el procedimiento se despierta debe atender la interrupción. Para ello lo primero que debe hacer es obtener la fecha actual del reloj electrónico (RTC).
- Registrar Evento: Por último, el procedimiento almacena en la base de datos el código de evento relacionado con la interrupción activada y la fecha obtenida. Una vez almacenada la información, el procedimiento vuelve bloquearse hasta la próxima interrupción.

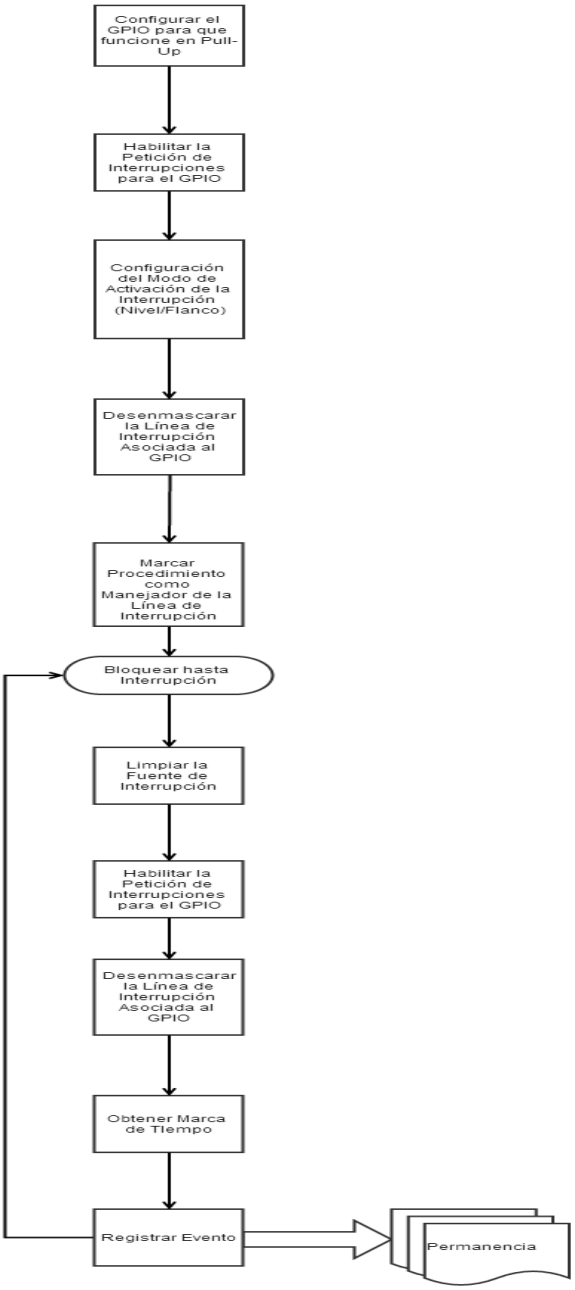


Ilustración 9.4 Diagrama de Flujo Final del Procedimiento de Detección de Eventos

El acceso al mapeo de las entradas y salidas en memoria es restringido por QNX a través de unas funciones especiales para direccionar las distintas zonas de memoria y leer y escribir en ellas. Esto permite un manejo más seguro y controlado del hardware.

Para la captura de interrupciones, QNX también tiene sus propias funciones que permiten asociar un manejador de interrupción, una función en el caso de C, a una línea de interrupción hardware. El microkernel se encarga de ejecutar el manejador siempre que dicha interrupción se produzca. El código fuente de este procedimiento puede consultarse en el directorio del CD "/Código Fuente/BeagleBone/Detector de Eventos".

- **Criterio de Verificación:**

El criterio de verificación de este procedimiento es el de generar un registro en la base de datos cuando se activa un GPIO asociado a una línea de interrupción para la detección de eventos. Una vez cumplido esto, la iteración se da por concluida.

9.3 Comunicación Socket:

El último procedimiento que conforma el núcleo de la aplicación es el encargado de ofrecer una interfaz socket a través del puerto Ethernet a dispositivos externos.

El procedimiento debe atender las peticiones que lleguen y realizar las acciones pertinentes, ya sea el cambio de un parámetro de configuración o el envío de vuelta de los datos recogidos para la visualización de los datos obtenidos por el sistema.

El protocolo es muy simple, consta de un código que identifica el comando deseado junto a los parámetros necesarios separados por un carácter especial. Del mismo modo, el procedimiento devuelve, si procede, los datos pertinentes con un carácter de separación entre grupos de información y otro entre los distintos parámetros del grupo.

El formato de los comandos de entrada es el siguiente:

ENTRADA = “COMANDO[PARÁMETRO]*”

Lo que significa que los comandos que llegan por socket deben empezar con el código numérico del comando que se desea ejecutar seguido de los parámetros que se requieran separados por espacio. El espacio como separador se ha elegido debido a que las funciones de C para tratamiento de cadenas de caracteres con son muy potentes, y el carácter que mejor reconoce como separador en sus funciones estándar es este.

En el caso de las salidas se codificarán de la siguiente manera.

SALIDA = “OBJETO[?OBJETO]*”

Siendo objeto

OBJETO = ATRIBUTO[&ATRIBUTO]*

Así podemos enviar como respuesta un grupo de señales con sus valores si se ejecuta una monitorización del sistema o el histórico de valores de una señal usando una sola comunicación, lo que reduce el tiempo de establecimiento de conexión con respecto a si se enviase en múltiples salidas.

Los comandos que se pueden enviar al procedimiento son los indicados en la ilustración 9.5:

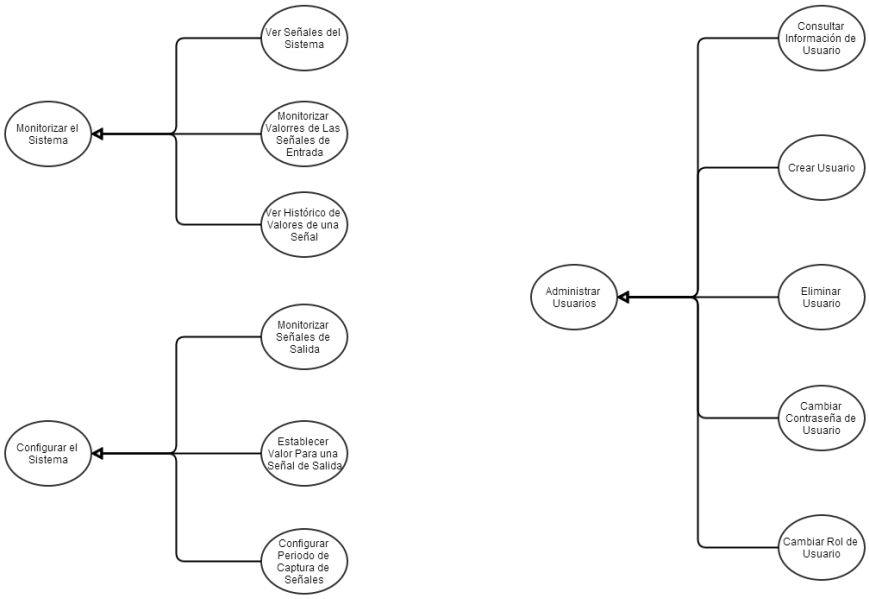


Ilustración 9.5 Comandos del Procedimiento de Comunicación Socket

Tabla 9.1 Comando Ver Señales del Sistema

Nombre	Ver Señales del Sistema
Precondiciones	-
Postcondiciones	-
Flujo Principal	1. Se accede a la base de datos y se obtienen las señales registradas 2. Se codifican las señales para generar la respuesta 3. Se devuelve la respuesta al proceso de comunicaciones socket
Formato de Mensaje de Entrada	-
Formato de Respuesta	Respuesta = “Señal?Señal?...” Señal = “Marca&Modelo&Número&Tipo&Dirección&Entrada/Salida”

Tabla 9.2 Comando Monitorizar Valores de las Señales de Entrada

Nombre	Monitorizar Valores de Las Señales de Entrada
Precondiciones	-
Postcondiciones	-
Flujo Principal	1. Se accede a la base de datos y se obtienen las señales de entrada registradas junto a su último valor obtenido 2. Se codifican las señales para generar la respuesta 3. Se devuelve la respuesta al proceso de comunicaciones socket
Formato de Mensaje de Entrada	-
Formato de Respuesta	Respuesta = “Señal&Valor?Señal&Valor?...” Señal = “Marca&Modelo&Número&Tipo&Dirección”

Tabla 9.3 Ver Histórico de Valores de una Señal

Nombre	Ver Histórico de Valores de una Señal
Precondiciones	-
Postcondiciones	-
Flujo Principal	1. Se decodifica el mensaje de entrada para obtener la señal que se desea monitorizar 2. Se accede a la base de datos para y so obtienen todos los registro de la señal 3. Se codifican los registros para generar la respuesta 4. Se devuelve la respuesta al proceso de comunicaciones socket
Formato de Mensaje de Entrada	Mensaje = “Señal” Señal = “Marca Modelo Número Tipo Dirección Entrada/Salida”
Formato de Respuesta	Respuesta = “Fecha&Valor?Fecha&Valor?...” Fecha = “yyyy-mm-ddThh:mm:ss”

Tabla 9.4 Comando Monitorizar Valores de las Señales de Salida

Nombre	Monitorizar Valores de las Señales de Salida
Precondiciones	-
Postcondiciones	-
Flujo Principal	1. Se accede a la base de datos y se obtienen las señales de salida registradas junto a su último valor obtenido 2. Se codifican las señales para generar la respuesta 3. Se devuelve la respuesta al proceso de comunicaciones socket
Formato de Mensaje de Entrada	-
Formato de Respuesta	Respuesta = “Señal&Valor?Señal&Valor?...” Señal = “Marca&Modelo&Número&Tipo&Dirección”

Tabla 9.5 Comando Establecer Valor para una Señal de Salida

Nombre	Establecer Valor para una Señal de Salida
Precondiciones	-
Postcondiciones	Las salidas del dispositivo se modifican
Flujo Principal	1. Se decodifica el mensaje para obtener la señal cuyo valor se desea cambiar y el valor en sí 2. Se cambia a nivel hardware el valor de la señal 3. Se accede a la base de datos y se registra el nuevo valor de la señal
Formato de Mensaje de Entrada	Mensaje = “Señal Valor” Señal = “Marca Modelo Número Tipo Dirección”
Formato de Respuesta	-

Tabla 9.6 Comando Configurar Periodo de Captura de Datos

Nombre	Configurar Periodo de Captura de Datos
Precondiciones	-
Postcondiciones	Periodo de configuración modificado
Flujo Principal	1. Se decodifica el mensaje para obtener el periodo 2. Se envía un mensaje al servidor del procedimiento de captura de datos con el nuevo periodo 3. El procedimiento ajusta el periodo con el valor obtenido
Formato de Mensaje de Entrada	Mensaje = “Periodo”
Formato de Respuesta	-

Tabla 9.7 Comando Consultar Información de Usuario

Nombre	Consultar Información de Usuario
Precondiciones	-
Postcondiciones	-
Flujo Principal	1. Se decodifica el mensaje para obtener el usuario al que se buscar 2. Se accede a la base de datos y se obtiene el rol y contraseña encriptada del usaurio 3. Se codifica la respuesta y se envía al procedimiento de comunicación socket 4. En caso de que el usaurio no exista la respuesta devuelta no contiene nada
Formato de Mensaje de Entrada	Mensaje = “NombreDeUsuario”
Formato de Respuesta	Respuesta = “Contraseña&Rol”

Tabla 9.8 Comando Crear Usuario

Nombre	Crear Usuario
Precondiciones	-
Postcondiciones	Un nuevo usuario está registrado en el sistema
Flujo Principal	1. Se decodifica el mensaje para obtener los datos del nuevo usuario 2. Se accede a la base de datos y se inserta el nuevo usuario si este no existe 3. Se envía una respuesta de confirmación de la creación del nuevo usuario
Formato de Mensaje de Entrada	Mensaje = "Usuario" Usuario = "Nombre Contraseña Rol"
Formato de Respuesta	Respuesta = "Contraseña&Rol"

Tabla 9.9 Comando Eliminar Usuario

Nombre	Eliminar Usuario
Precondiciones	-
Postcondiciones	El usuario deja de existir en el sistema
Flujo Principal	1. Se decodifica el mensaje para obtener el nombre del usuario que se desea eliminar 2. Se accede a la base de datos y se elimina el usuario
Formato de Mensaje de Entrada	Mensaje = "NombreDeUsuario"
Formato de Respuesta	-

Tabla 9.10 Comando Modificar Contraseña de Usuario

Nombre	Modificar Contraseña de Usuario
Precondiciones	-
Postcondiciones	El registro del usuario se modifica con la nueva contraseña
Flujo Principal	1. Se decodifica el mensaje para obtener el usuario y su nueva contraseña 2. Se accede a la base de datos y modifica el registro del usuario
Formato de Mensaje de Entrada	Mensaje = “NombreDeUsuario Contraseña”
Formato de Respuesta	-

Tabla 9.11 Comando Modificar Rol de Usuario

Nombre	Modificar Rol de Usuario
Precondiciones	-
Postcondiciones	El registro del usuario se modifica con el nuevo rol
Flujo Principal	1. Se decodifica el mensaje para obtener el usuario y su nuevo rol 2. Se accede a la base de datos y modifica el registro del usuario
Formato de Mensaje de Entrada	Mensaje = “NombreDeUsuario Rol”
Formato de Respuesta	-

- **Diseño:**

Para este proceso se busca la mayor modularidad posible para que la adición de nuevos comandos a la aplicación sea sencilla. Por ello el procedimiento debe encargarse solo de la parte de comunicaciones socket, decodificar la información para pasársela al comando que se deba ejecutar y enviar la información que genere dicho comando. Con este propósito los comandos tendrán una estructura igual en cuanto a los parámetros de entrada que reciben y a la salida, así la llamada a estos puede ser totalmente homogénea y abstrae a la parte encargada de la comunicación socket de los detalles de implementación de cada comando.

Tal y como se puede observar en el diagrama de flujo de la ilustración 9.6, el procedimiento inicializa un vector con las funciones que implementan cada comando. Cada posición del vector coincide con el código numérico que identifica a cada comando. Esto se puede hacer gracias a que se ha declarado todas las funciones que implementan la ejecución de un comando con una interfaz de entrada y salida idéntica, lo que permite almacenarlos en un vector como si fuesen elementos del mismo tipo. De esta forma logramos una aplicación potente en cuanto a su capacidad para añadir nuevos comandos sin mayor dificultad.

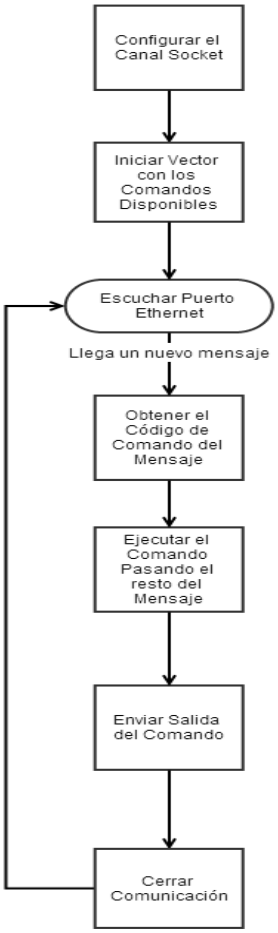


Ilustración 9.6 Diagrama de Flujo del Procedimiento de Comunicación Socket

- **Implementación:**

Para la implementación de los comandos se sigue una estructura de flujo común marcada por el modo de implementar el procedimiento encargado de la comunicación socket. De este modo, todos los comandos siguen este comportamiento (Ver ilustración 9.7):

- En primer lugar el comando debe decodificar el mensaje que le pasa el procedimiento de comunicación socket para determinar los parámetros que debe usar para ejecutarse.
- Una vez hecho esto, se ejecutan las acciones propias de cada comando.
- En caso de haber algún error, se registra un evento en el sistema.
- Si no, el comando debe codificar la respuesta con el formato indicado anteriormente y devolverla al procedimiento de comunicación socket.

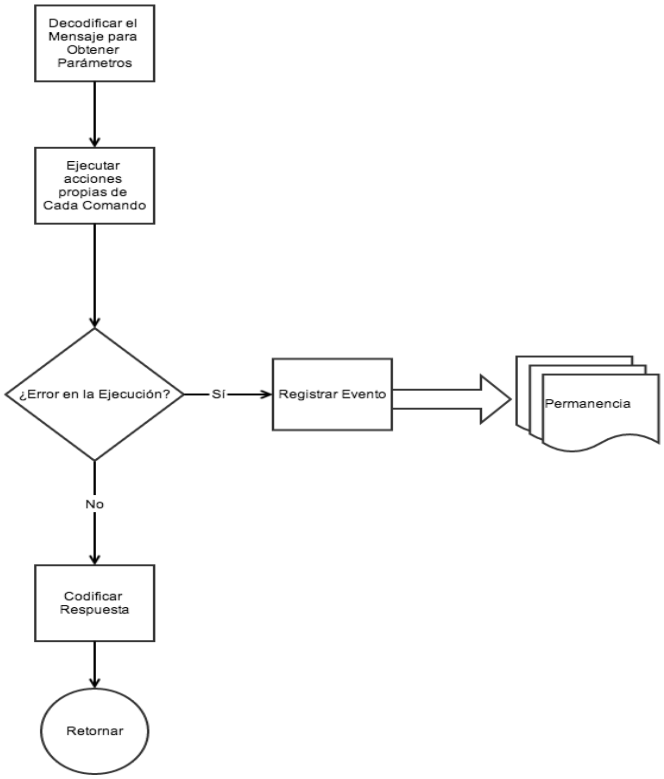


Ilustración 9.7 Diagrama de Flujo General de los Comandos

El código fuente de este procedimiento puede encontrarse en el directorio del CD “Código Fuente/BeagleBone/Comunicación Socket”

- **Criterio de Verificación:**

Para este procedimiento el criterio de verificación es más complejo, ya que depende de las comunicaciones socket con la interfaz de usaurio. Para dar por válido este procedimiento es necesario que cada uno de los comandos declarados sea ejecutado correctamente cuando la interfaz de usuario lo ordene. Es por ello que la verificación de este procedimiento se hará junto a la verificación del sistema una vez esté desarrollada la interfaz de usuario.

10. Comunicación entre Procedimientos

Debido a que la aplicación se ha diseñado separando las distintas funcionalidades para mejorar la arquitectura de la misma, es necesario diseñar un sistema de comunicación entre procedimientos para poder transmitir la información que necesita cada uno de ellos. Esto se debe a que el procedimiento de captura de datos necesita conocer en tiempo de ejecución el periodo de espera entre lecturas de las señales físicas indicado por el usuario. Esta información le llega al procedimiento de comunicación socket a través del comando "Configurar Periodo de Captura de Datos", por lo que es necesaria una vía de comunicación entre ambos procedimientos. Esto sólo es posible con los recursos que proporciona un sistema operativo en tiempo real como lo es QNX. A continuación, se describe las etapas de diseño e implementación del hilo que se añade, que será el encargado de gestionar la configuración del procedimiento cuando el usuario lo requiera

- **Diseño:**

Para poder realizar esta comunicación directa entre procedimientos se realiza una propuesta de diseño basada otra de las funcionalidades de programación que ofrece QNX, el paso de mensajes. Esto permite enviar datos entre procedimientos siempre y cuando el procedimiento emisor conozca el identificador del receptor y que ambos conozcan la estructura de datos del mensaje y la respuesta.

De este modo, el procedimiento de comunicaciones socket enviará un mensaje con el periodo entre lecturas al procedimiento de obtención de datos. Una vez recibido el mensaje, el procedimiento modifica su comportamiento para reflejar el cambio indicado por el usuario.

Con todos estos elementos la arquitectura del sistema central queda definida tal y como se muestra en el siguiente modelo de la ilustración 10.1.

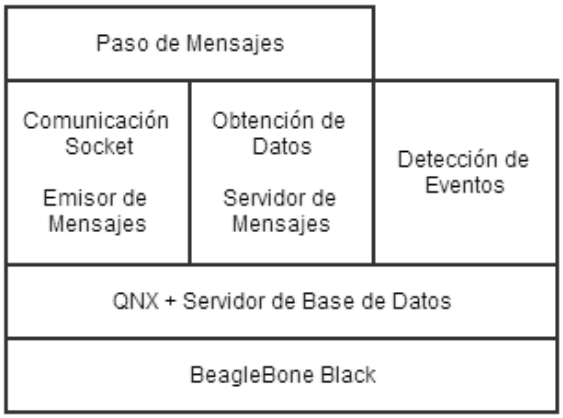


Ilustración 10.1 Estructura de la Aplicación por Capas de Abstracción

Como se puede apreciar, existen cuatro capas dentro de la arquitectura. En los niveles más bajos de abstracción se encuentran el hardware y el sistema operativo con el servidor de la base de datos, cimientos sobre los que se asienta la aplicación, dividida en tres procedimientos que trabajan conjuntamente, como ya se ha explicado. El último nivele de la arquitectura corresponde al método de comunicación entre procedimientos por paso de mensajes propio del sistema operativo, que permite una comunicación directa entre el protocolo de comunicaciones y el procedimiento de recogida de datos.

- **Implementación:**

Para la implementación del método de comunicación es necesario modificar el procedimiento de captura de datos. Para soportar la comunicación entre éste y el procedimiento de comunicación será necesario añadir un hilo al procedimiento de lectura de señales que hará la función de servidor de mensajes. Este hilo estará bloqueado a la espera de que el sistema operativo le notifique un mensaje proveniente del procedimiento de comunicación socket consistente en un único valor numérico que indica el periodo de tiempo entre lecturas que desea el usuario. Una vez el hilo recibe el mensaje, éste lo usa para actualizar el timer, que ahora se encontrará en una zona de memoria compartida por los dos hilos del procedimiento de lectura de señales. De este modo el hilo de lecturas es ajeno a todas las modificaciones en el timer y funciona sin interrumpir sus funciones. (Ver Ilustración 10.2)

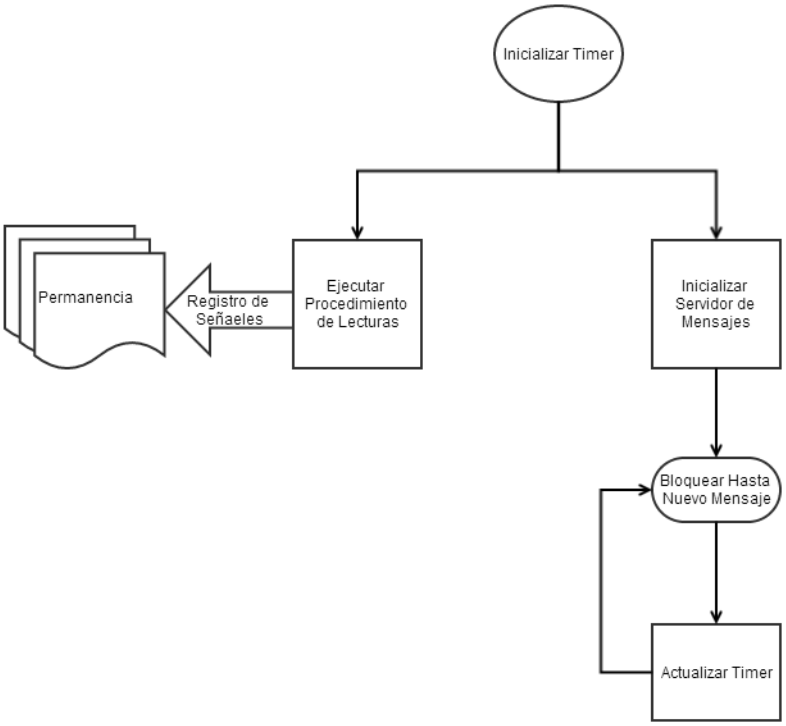


Ilustración 10.2 Diagrama de Flujo del Servidor de Mansajes del Procedimiento de Captura de Datos

11. Interfaz de Usuario

Para finalizar, y como control del buen funcionamiento del sistema, se realiza una aplicación web que hace uso del protocolo de comunicación con el sistema central (placa BeagleBone). De esta manera será posible visualizar la monitorización de los datos que se recojan, controlar las señales físicas de salida y configurar los parámetros de control de la recogida de datos.

Dentro de la aplicación existirán tres tipos de usuarios (Roles):

- Monitorización: El usuario con un rol de monitorización podrá acceder exclusivamente a la visualización de datos de los datos y al histórico de datos en el tiempo de cada señal física.
- Control: El usuario que disponga de los permisos de control podrá, además de visualizar los datos como el rol de monitorización, modificar el estado de las señales físicas de salida y configurar los parámetros de control de la recogida de datos.
- Administración: El administrador es el usuario con todos los privilegios de la aplicación. Puede realizar todas las funciones de los roles de monitorización y control y, además, administrar los usuarios registrados en la aplicación y consultar los eventos que se han producido en el sistema.

- **Análisis:**

El primer paso para la implementación de la interfaz de usuario es plantear los casos de uso. Todos los casos de uso pueden verse a continuación.

Tabla 11.1 Caso de Uso Iniciar Sesión

Nombre	Iniciar sesión
Actor	Usuario registrado en el sistema
Precondiciones	-
Postcondiciones	Usuario logueado en el sistema
Flujo Principal	1. El usuario introduce su nombre de usuario y contraseña en la interfaz 2. La interfaz se comunica con el sistema central para la consulta del usuario 3. El sistema central ejecuta el comando de Obtención de Datos de Usuario y devuelve el resultado 4. La interfaz compara la contraseña introducida por el usuario con la contraseña encriptada obtenida del sistema central 5. A. Si el encriptado de las contraseñas coincide se da acceso a la interfaz al usuario 6. Fin del caso de uso

Tabla 11.2 Caso de Uso Visualizar Valor de las Señales

Nombre	Visualizar Valor de las Señales
Actor	Usuario con rol de monitorización
Precondiciones	Usuario logueado en el sistema
Postcondiciones	-
Flujo Principal	1. El usuario selecciona la opción de monitorización de señales 2. La interfaz se comunica con el sistema central para la consulta 3. El sistema central ejecuta el comando de Obtención de Valores de las Señales y devuelve la respuesta 4. El sistema de interfaz muestra la información 5. Fin del caso de uso

Tabla 11.3 Caso de Uso Consultar Histórico de Datos

Nombre	Consultar Histórico de Datos
Actor	Usuario con rol de Monitorización
Precondiciones	El usuario debe estar logueado en el sistema
Postcondiciones	-
Flujo Principal	1. El usuario selecciona la opción de visionado de histórico de datos 2. La interfaz accede al sistema central para obtener las señales registradas y las muestra al usuario 3. El usuario selecciona la señal que desea monitorizar 4. La interfaz se conecta al dispositivo central para la consulta 5. El dispositivo central ejecuta el comando de obtención de histórico de valores de una señal y devuelve la respuesta 6. La interfaz genera una gráfica con los valores obtenidos y la muestra al usuario 7. Fin del caso de uso

Tabla 11.4 Caso de Uso Modificar Perfil de Usuario

Nombre	Modificar Perfil de Usuario
Actor	Usuario con rol de Monitorización
Precondiciones	El usuario debe estar logueado en el sistema
Postcondiciones	La información del usuario se modifica en la base de datos
Flujo Principal	1. El usuario selecciona la opción de modificar perfil de usuario 2. El usuario introduce sus nuevos datos de usuario 3. La interfaz se comunica con el dispositivo central y notifica la modificación 4. El dispositivo central ejecuta el comando Modificación de Usuario y confirma el cambio 5. Fin del caso de uso

Tabla 11.5 Caso de Uso Seleccionar Tasa de Refresco del Valor de las Señales

Nombre	Seleccionar Tasa de Refresco del Valor de las Señales
Actor	Usuario con rol de Control
Precondiciones	El usuario debe estar logueado en el sistema
Postcondiciones	El periodo temporal de captura de señales se modifica
Flujo Principal	1. El usuario selecciona en la interfaz la opción de configuración de la tasa de refresco 2. El usuario introduce la nueva tasa de refresco 3. El sistema de interfaz se conecta al sistema central y notifica la modificación 4. El sistema central ejecuta el comando Modifiación del Periodo de Lecturas 5. Fin del caso de uso

Tabla 11.6 Caso de Uso Modificar el Valor de una Señal de Salida

Nombre	Modificar el Valor de una Señal de Salida
Actor	Usuario con rol de Control
Precondiciones	El usuario debe estar logueado
Postcondiciones	El valor de la señal de salida seleccionada es modificada
Flujo Principal	1. El usuario selecciona la opción de control del sistema 2. El usuario selecciona la señal que desea modificar e introduce el nuevo valor 3. La interfaz se comunica con el sistema central para notificar el cambio de estado de la señal 4. El sistema central modifica el valor de salida de la señal 5. El sistema central almacena en la base de datos el registro con el nuevo valor de la señal modificada y confirma el cambio al sistema de interfaz de usuario 6. Fin del caso de uso

Tabla 11.7 Caso de Uso Crear Usuario

Nombre	Crear usuario
Actor	Administrador
Precondiciones	El administrador debe estar logueado en el sistema de interfaz
Postcondiciones	El nuevo usuario se almacena en la base de datos y está disponible para iniciar sesión con sus datos
Flujo Principal	1. El administrador selecciona la opción de crear usuario en la sección de administración de usuarios 2. El administrador introduce el nuevo nombre, contraseña y rol del usuario 3. La interfaz se comunica con el dispositivo central y envía los datos del nuevo usuario 4. El sistema central ejecuta el comando de Creación de Nuevo Usuario 5. Fin del caso de uso

Tabla 11.8 Caso de Uso Eliminar Usuario

Nombre	Eliminar usuario
Actor	Administrador
Precondiciones	El administrador debe estar logueado en el sistema de interfaz
Postcondiciones	El usuario seleccionado se elimina de la base de datos y ya no es posible iniciar sesión con sus datos
Flujo Principal	1. El administrador selecciona la opción de eliminar usuario sobre el usuario deseado 2. La interfaz se conecta con el sistema central y envía el nombre de usuario que se desea eliminar 3. El sistema central ejecuta el comando Eliminación de Usuario 4. El sistema central envía una confirmación al sistema de interfaz 5. Fin del caso de uso

Tabla 11.9 Caso de Uso Modificar Usuario

Nombre	Modificar usuario
Actor	Administrador
Precondiciones	El administrador debe estar logueado en el sistema de interfaz
Postcondiciones	
Flujo Principal	1. El administrador selecciona la opción de modificar usuario sobre el usuario deseado 2. El administrador introduce los nuevos datos de usuario 3. La interfaz se conecta con el dispositivo central y envía el nombre de usuario antiguo junto a los nuevos datos de usuario 4. El sistema central ejecuta el comando de Modificación de Usuario 5. El sistema central envía una confirmación al sistema de interfaz 6. Fin del caso de uso

Tabla 11.10 Caso de Uso Consultar Historial de Eventos

Nombre	Consultar Historial de Eventos
Actor	Administrador
Precondiciones	El administrador debe estar logueado en el sistema de interfaz
Postcondiciones	
Flujo Principal	1. El administrador selecciona en la interfaz la pestaña de administración 2. La interfaz se comunica con el dispositivo central para la consulta de los eventos 3. El dispositivo central ejecuta el comando de Obtención de Registro de Eventos y devuelve la información 4. La interfaz muestra la información de los eventos generados por el sistema organizador por fecha de ocurrencia 5. Fin del caso de uso

Para mantener la seguridad de acceso de los usuarios la contraseña será almacenada codificada para evitar que intrusos puedan acceder a los datos de usuario.

- **Diseño:**

El diseño para la interfaz web se base en tres componentes básicos: Las vistas (View), los comandos (Command) y la clase de comunicación socket (SocketCommunication), que permite unificar en un mismo sitio todos los accesos al sistema central.

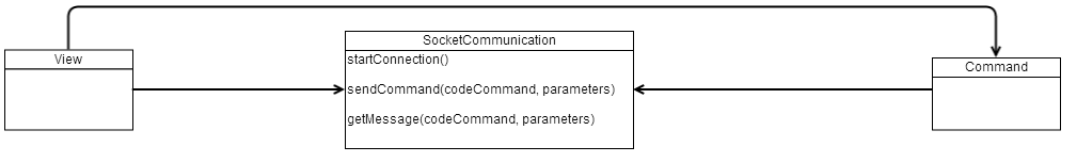


Ilustración 11.1 Diseño Base de las Clases de la Aplicación Web

Como se puede apreciar en la Ilustración 11.1 las vistas usan la clase de comunicación socket para obtener los datos que deben mostrar y a los comandos para ejecutar aquello que el usuario desee. Todo esto se hace a través de la interfaz que ofrece la clase de comunicaciones socket. En ella se aprecia la posibilidad de ejecutar un comando con los parámetros necesarios y obtener el mensaje que devuelve dicho comando a través de la función “getMessage”. Esta función se encarga de iniciar la conexión con el sistema central mediante la función “startConnection” y enviar el mensaje mediante la función “sendCommand”. Una vez hecho esto puede leer el resultado del comando ejecuta del canal socket.

Existen cinco vistas diferentes como se puede ver en la Ilustración 11.2:

- La vista de Monitorización de datos (SignalMonitoringView) permite ver todas las señales de entrada y sus valores.
- La vista de histórico de datos (HistoricalDataView) muestra una lista de las señales registradas y, una vez seleccionada una de ellas, llama al comando de obtención de histórico de datos para esa señal y muestra una gráfica con la evolución de su valor a lo largo del tiempo.
- La vista de control de señales (SignalControlVIew) permite al usuario ver los valores de las señales de salida y configurar su valor a través del comando de modificación de señal. También permite la configuración del periodo de tiempo entre lecturas a través de comando de cambio de la tasa de refresco.
- La vista de administración (AdminView) permite consultar los eventos ocurridos en el sistema y ver el listado de usuarios registrados. En esta vista, y a través de los comandos, se puede crear un nuevo usuario, modificar el rol y contraseña de uno ya existente o eliminarlo.
- La vista de perfil de usuario (UserProfileView) muestra la información del usuario que se encuentra logueado actualmente y permita la modificación de su contraseña.

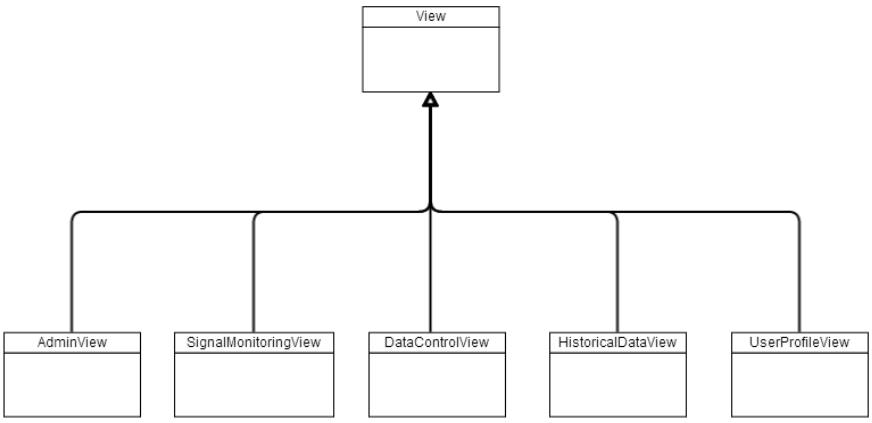


Ilustración 11.2 Diseño de las Clases de Vista

Por último, el usuario puede ejecutar distintos comandos en el sistema, como el inicio de sesión (Login), la administración de usuario (CreateUser, RemoveUser,…) o la configuración de los parámetros de funcionamiento (ChangeRefreshRate). En la ilustración 11.3 pueden verse todos los comandos disponibles en la interfaz de usuario.

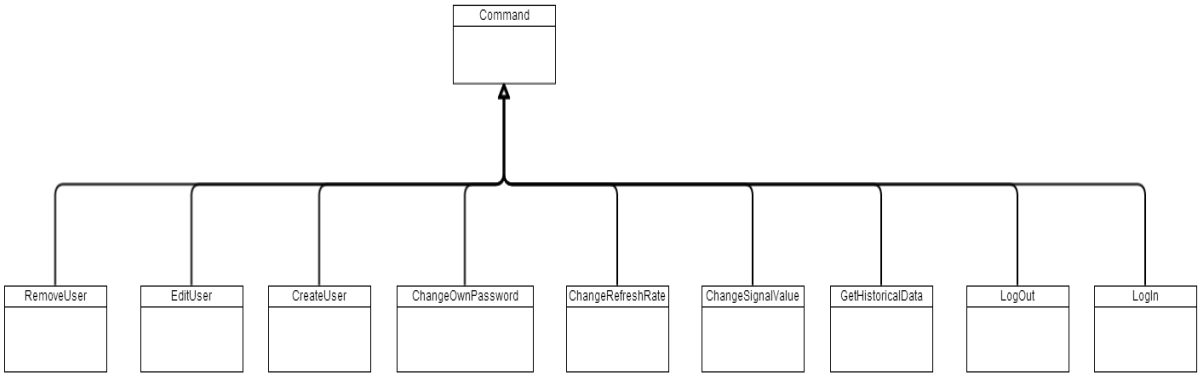


Ilustración 11.3 Comandos de la Interfaz de Usuario

- **Implementación:**

Durante la implementación de la interfaz se ha añadido una clase al modelo que permite encriptar las contraseñas y comparar la contraseña que introduce el usuario con la contraseña encriptada almacenada y así comprobar que es correcta. Esto permite reducir el tamaño de las clases de comando que maneja la contraseña del usuario como el LogIn o la creación de un nuevo usuario.

Para una mejor visualización del registro de valores de las señales se ha utilizado una librería en javascript llamada dygraph. Esta librería permite generar gráficas de una forma muy sencilla e intuitiva.

Una vez cargada la aplicación web en la placa Arduino YUN podemos acceder a ella desde el navegador y trabajar con ella.

En el menú superior hay seis secciones. La primera es la pantalla de inicio, ésta pantalla es a la que accede el usuario en primer lugar cuando accede a la aplicación. Las cinco opciones restantes del menú permiten acceder a las vistas descritas en la fase de diseño.

En la parte derecha de la pantalla se encuentra el menú de inicio de sesión. En este menú el usuario puede introducir su nombre de usuario y contraseña para loguearse en la aplicación o, si ya está logueado, cerrar sesión. (Ver ilustración 11.4)

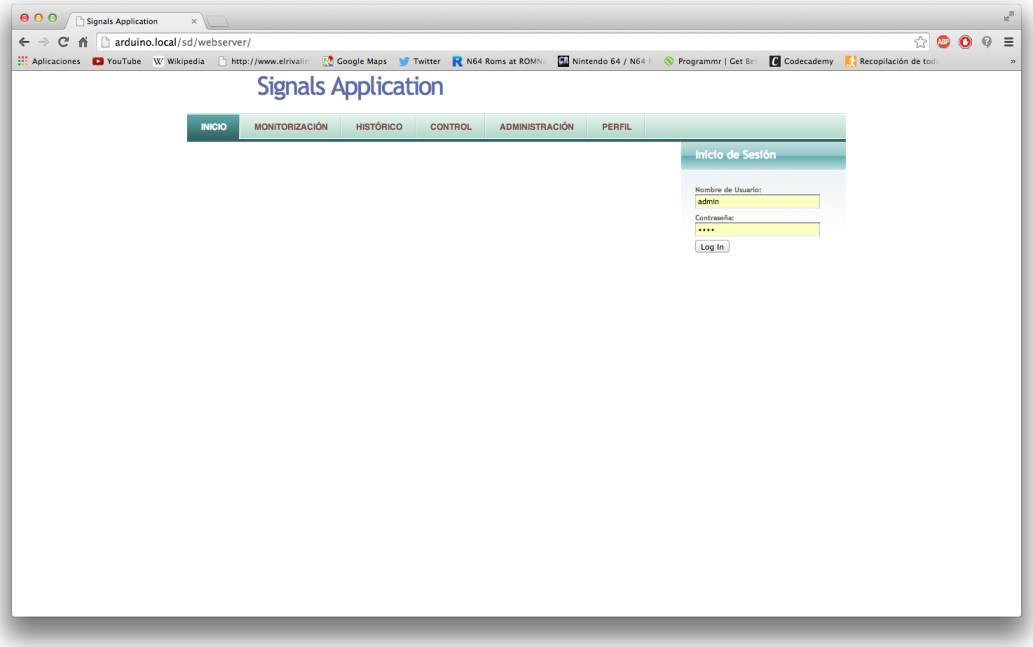


Ilustración 11.4 Pantalla de Inicio de la Aplicación Web

Se ha iniciado sesión con una cuenta de administrador para acceder a todos las partes de la aplicación. En primer lugar accederemos al histórico de datos. En esta vista podemos seleccionar cualquier señal física registrada en el sistema y visualizar los valores que se han registrado de la misma a lo largo del tiempo en una gráfica tal y como se aprecia en la ilustración 11.5.

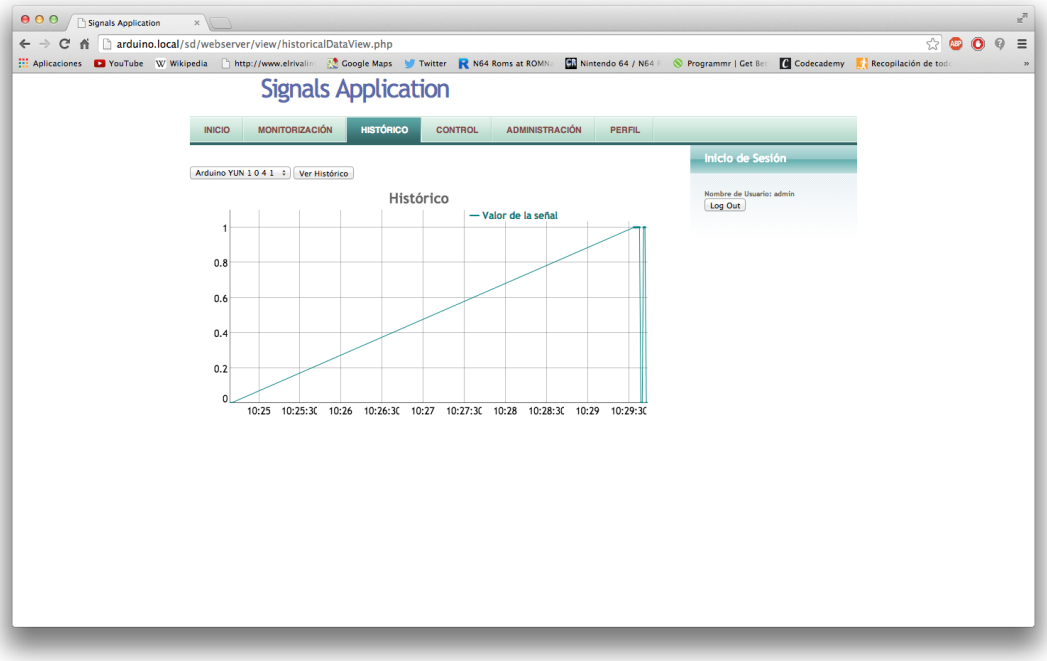


Ilustración 11.5 Vista del Histórico de Datos de una Señal

En la sección de control podemos modificar el periodo entre lecturas y establecer el valor de las señales físicas de salida. (Ilustración 11.6)

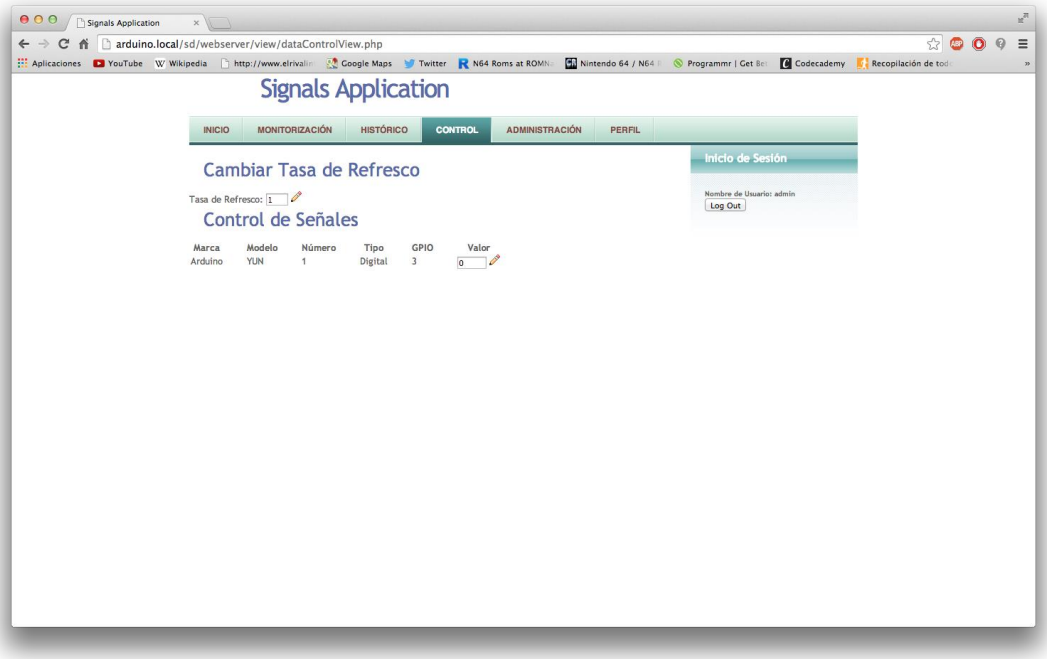


Ilustración 11.6 Vista de la sección de Control del Sistema

En la sección de administración se accede a toda la gestión de usuarios y al registro de eventos. Esto puede verse en la ilustración 11.7.

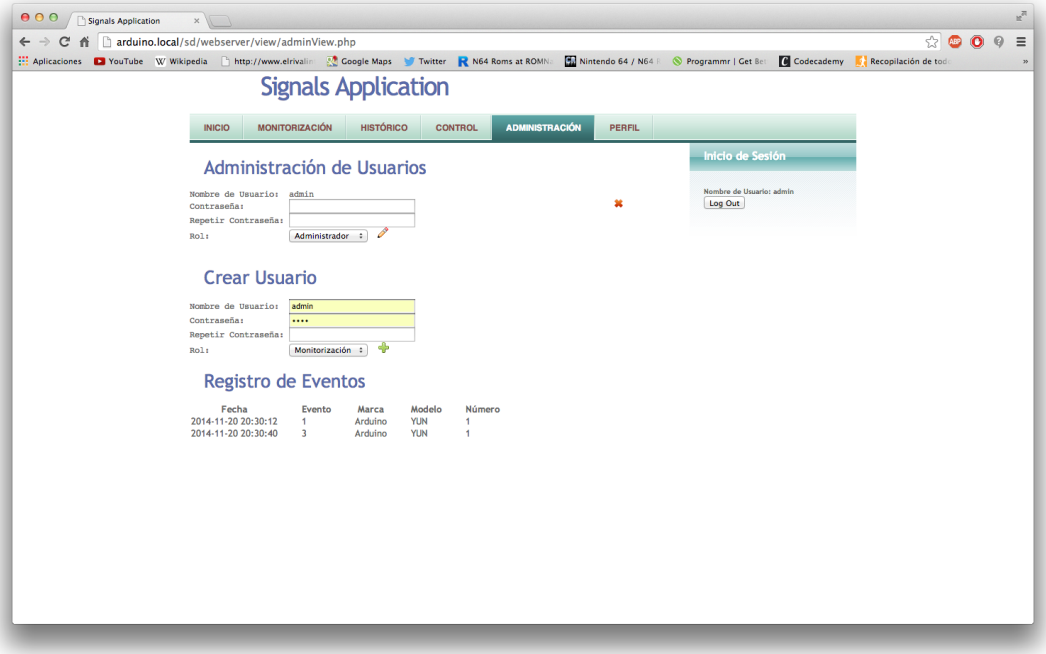


Ilustración 11.7 Vista de Administración

El código de la aplicación web puede consultarse en el directorio del CD “Código Fuente/YUN/Aplicación Web”

12. Conclusiones

Tras finalizar el desarrollo del prototipo de la aplicación podemos concluir lo siguiente:

- Un sistema operativo en tiempo real es una herramienta vital para el desarrollo de aplicaciones con requisitos de tiempo o que dependan de una respuesta muy rápida ante eventos externos. Sin embargo, el sistema operativo solo ofrece herramientas que facilitan el cumplimiento de estos requisitos, pero no dispone de procedimientos automatizados. El programador los tiene que desarrollar e integrar en el sistema operativo.

- Es necesario explorar vías de almacenamiento de la información externa a los dispositivos empotrados del tipo BeagleBone si se desea un registro histórico que ocupe en memoria un tamaño mayor que la capacidad de la BeagleBone Black, que es limitada y se hace pequeña para aplicaciones de gran envergadura. Se propone como alternativa el uso de un script que se ejecute periódicamente que traspase la información recogida a un servidor externo de mayor capacidad. En este TFG se han desarrollado todos los elementos básicos para llevar a cabo este procedimiento.

- A pesar de que la aplicación prototipo está diseñada desde un enfoque generalista, ésta puede ser trasladada a una aplicación real con pocos cambios en su estructura. La capacidad para conectar distintos dispositivos periféricos la convierte en una candidata excepcional como sistema de control en entornos industriales.

Como principales aportaciones de este trabajo se destacan las siguientes:

- Realización de un procedimiento de recogida de datos de señales físicas en un dispositivo empotrados en tiempo real. Pudiendo modificar el periodo de lecturas en tiempo de ejecución.
- Realización de un software para la detección temprana de eventos a través de interrupciones hardware haciendo uso de los pines digitales de un sistema empotrado.
- Realización de un software de comunicaciones vía Ethernet que permite la trasmisión de información entre dos dispositivos.
- Realización de una aplicación web de acceso inalámbrico para visualizar y configurar los datos ubicados en un sistema empotrado remoto a través de un protocolo de comunicaciones vía Ethernet.

13. Bibliografía

- Arduino YUN:

Arduino; Guide to the Arduino Yún; http://arduino.cc/en/Guide/ArduinoYun; acceso: 8/07/2014

Arduino Scuola; Arduino Yun Intro to WebServer; http://scuola.arduino.cc/lesson/b4EoRkV/Arduino_Yn_Intro_to_web_server; acceso: 8/07/2014

Arduino; How to expand the Yun disk space; http://arduino.cc/en/Tutorial/ExpandingYunDiskSpace; acceso: 8/07/2014

Dygraphs; Librería Dygraphs; http://dygraphs.com/; acceso: 8/07/2014

- BeagleBone Black:

BeagleBoard; Documentación Hardware; http://elinux.org/Beagleboard:BeagleBoneBlack#Hardware_Files; acceso: 8/07/2014

- QNX:

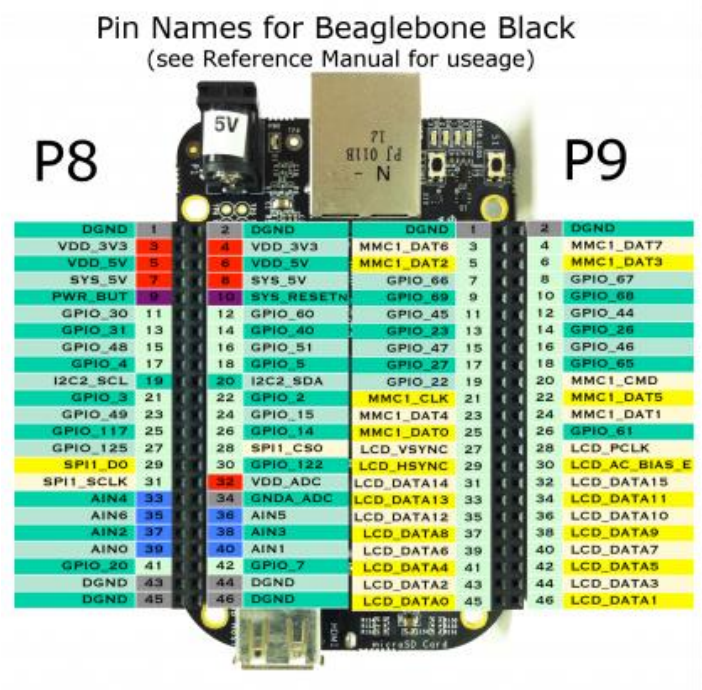
QNX; BSP para la BeagleBone Black y tutorial de instalación; http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/TiAm335Beaglebone; acceso: 8/07/2014

QNX; Developer Reference Manual; http://www.qnx.com/developers/docs/index.html; acceso: 8/07/2014

Apéndice A: Documentación Hardware

Anexo 1: Función Principal de los Pines de Entrada/Salida de la BeagleBone Black

La BeagleBone Black tiene dos cabeceras de expansión en los extremos con los 92 pines (46 en cada una) de entrada/salida que el usuario puede usar para el desarrollo de aplicaciones. En la siguiente gráfica se muestra la ubicación física de cada uno de los pines junto a su modo de funcionamiento por defecto.



P8				P9			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	MMC1_DAT6	3	4	MMC1_DAT7
VDD_5V	5	6	VDD_5V	MMC1_DAT2	5	6	MMC1_DAT3
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	MMC1_CMD
GPIO_3	21	22	GPIO_2	MMC1_CLK	21	22	MMC1_DAT5
GPIO_49	23	24	GPIO_15	MMC1_DAT4	23	24	MMC1_DAT1
GPIO_117	25	26	GPIO_14	MMC1_DAT0	25	26	GPIO_61
GPIO_125	27	28	SPI1_CS0	LCD_VSYNC	27	28	LCD_PCLK
SPI1_D0	29	30	GPIO_122	LCD_HSYNC	29	30	LCD_AC_BIAS_E
SPI1_SCLK	31	32	VDD_ADC	LCD_DATA14	31	32	LCD_DATA15
AIN4	33	34	GNDA_ADC	LCD_DATA13	33	34	LCD_DATA11
AIN6	35	36	AIN5	LCD_DATA12	35	36	LCD_DATA10
AIN2	37	38	AIN3	LCD_DATA8	37	38	LCD_DATA9
AIN0	39	40	AIN1	LCD_DATA6	39	40	LCD_DATA7
GPIO_20	41	42	GPIO_7	LCD_DATA4	41	42	LCD_DATA5
DGND	43	44	DGND	LCD_DATA2	43	44	LCD_DATA3
DGND	45	46	DGND	LCD_DATA0	45	46	LCD_DATA1

Ilustración A.1 Pines de la BeagleBone Black

Como se puede comprobar los pines 17 y 18 de la cabecera de expansión 8 funcionan por defecto como entradas/salidas de propósito general. Sin enbargo, dentro de la aplicación son los pines de reloj y datos que conectan por I2C la BeagleBone Black con el reloj.

Cada pin puede llegar a tener hasta 7 modos de funcionamiento distintos y configurables por el usuario.

Modos de Funcionamiento de la Cabecera 8

Head_pin	$PINS	ADDR/OFFSET	GPIO NO.	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0
P8_01				DGND								
P8_02				DGND								
P8_03	6	0x818/018	38	GPIO1_6	gpio1[6]						mmc1_dat6	gpmc_ad6
P8_04	7	0x81c/01c	39	GPIO1_7	gpio1[7]						mmc1_dat7	gpmc_ad7
P8_05	2	0x808/008	34	GPIO1_2	gpio1[2]						mmc1_dat2	gpmc_ad2
P8_06	3	0x80c/00c	35	GPIO1_3	gpio1[3]						mmc1_dat3	gpmc_ad3
P8_07	36	0x890/090	66	TIMER4	gpio2[2]					timer4		gpmc_advn_ale
P8_08	37	0x894/094	67	TIMER7	gpio2[3]					timer7		gpmc_oen_ren
P8_09	39	0x89c/09c	69	TIMER5	gpio2[5]					timer5		gpmc_be0n_cle
P8_10	38	0x898/098	68	TIMER6	gpio2[4]					timer6		gpmc_wen
P8_11	13	0x834/034	45	GPIO1_13	gpio1[13]	pr1_pru0_pru_r30_15		eQEP2B_in	mmc2_dat1	mmc1_dat5	lcd_data18	gpmc_ad13
P8_12	12	0x830/030	44	GPIO1_12	gpio1[12]	pr1_pru0_pru_r30_14		EQEP2A_IN	MMC2_DAT0	MMC1_DAT4	LCD_DATA19	GPMC_AD12
P8_13	9	0x824/024	23	EHRPWM2B	gpio0[23]			ehrpwm2B	mmc2_dat5	mmc1_dat1	lcd_data22	gpmc_ad9
P8_14	10	0x828/028	26	GPIO0_26	gpio0[26]			ehrpwm2_tripzone_in	mmc2_dat6	mmc1_dat2	lcd_data21	gpmc_ad10
P8_15	15	0x83c/03c	47	GPIO1_15	gpio1[15]	pr1_pru0_pru_r31_15		eQEP2_strobe	mmc2_dat3	mmc1_dat7	lcd_data16	gpmc_ad15
P8_16	14	0x838/038	46	GPIO1_14	gpio1[14]	pr1_pru0_pru_r31_14		eQEP2_index	mmc2_dat2	mmc1_dat6	lcd_data17	gpmc_ad14
P8_17	11	0x82c/02c	27	GPIO0_27	gpio0[27]			ehrpwm0_synco	mmc2_dat7	mmc1_dat3	lcd_data20	gpmc_ad11
P8_18	35	0x88c/08c	65	GPIO2_1	gpio2[1]	mcasp0_fsr			mmc2_clk	gpmc_wait1	lcd_memory_clk	gpmc_clk_mux0
P8_19	8	0x820/020	22	EHRPWM2A	gpio0[22]			ehrpwm2A	mmc2_dat4	mmc1_dat0	lcd_data23	gpmc_ad8
P8_20	33	0x884/084	63	GPIO1_31	gpio1[31]	pr1_pru1_pru_r31_13	pr1_pru1_pru_r30_13			mmc1_cmd	gpmc_be1n	gpmc_csn2
P8_21	32	0x880/080	62	GPIO1_30	gpio1[30]	pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12			mmc1_clk	gpmc_clk	gpmc_csn1
P8_22	5	0x814/014	37	GPIO1_5	gpio1[5]						mmc1_dat5	gpmc_ad5
P8_23	4	0x810/010	36	GPIO1_4	gpio1[4]						mmc1_dat4	gpmc_ad4
P8_24	1	0x804/004	33	GPIO1_1	gpio1[1]						mmc1_dat1	gpmc_ad1
P8_25	0	0x800/000	32	GPIO1_0	gpio1[0]						mmc1_dat0	gpmc_ad0
P8_26	31	0x87c/07c	61	GPIO1_29	gpio1[29]							gpmc_csn0
P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8				gpmc_a8	lcd_vsync
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10				gpmc_a10	lcd_pclk
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9				gpmc_a9	lcd_hsync
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]						gpmc_a11	lcd_ac_bias_en
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn		uart5_rxd	mcasp0_axr1	eQEP1_index	gpmc_a18	lcd_data14
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn		mcasp0_axr3	mcasp0_ahclkx	eQEP1_strobe	gpmc_a19	lcd_data15
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn		mcasp0_axr3	mcasp0_fsr	eQEP1B_in	gpmc_a17	lcd_data13
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn		mcasp0_axr2	mcasp0_ahclkr	ehrpwm1B	gpmc_a15	lcd_data11
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn		mcasp0_axr2	mcasp0_aclkr	eQEP1A_in	gpmc_a16	lcd_data12
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn			mcasp0_axr0	ehrpwm1A	gpmc_a14	lcd_data10
P8_37	48	0x8c0/0c0	78	UART5_TXD	gpio2[14]	uart2_ctsn		uart5_txd	mcasp0_aclkx	ehrpwm1_tripzone_in	gpmc_a12	lcd_data8
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn		uart5_rxd	mcasp0_fsx	ehrpwm0_synco	gpmc_a13	lcd_data9
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6		eQEP2_index		gpmc_a6	lcd_data6
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7	pr1_edio_data_out7	eQEP2_strobe		gpmc_a7	lcd_data7
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4		eQEP2A_in		gpmc_a4	lcd_data4
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5		eQEP2B_in		gpmc_a5	lcd_data5
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2		ehrpwm2_tripzone_in		gpmc_a2	lcd_data2
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3		ehrpwm0_synco		gpmc_a3	lcd_data3
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0		ehrpwm2A		gpmc_a0	lcd_data0
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1		ehrpwm2B		gpmc_a1	lcd_data1

Modos de Funcionamiento de la Cabecera 9

Head_pin	$PINS	ADDR/OFFSET	Name	GPIO NO.	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0
P9_01			GND									
P9_02			GND									
P9_03			DC_3.3V									
P9_04			DC_3.3V									
P9_05			VDD_5V									
P9_06			VDD_5V									
P9_07			SYS_5V									
P9_08			SYS_5V									
P9_09			PWR_BUT									
P9_10			SYS_RESETn									RESET_OUT
P9_11	28	0x870/070	UART4_RXD	30	gpio0[30]	uart4_rxd_mux2		mmc1_sdcd	rmii2_crs_dv	gpmc_csn4	mii2_crs	gpmc_wait0
P9_12	30	0x878/078	GPIO1_28	60	gpio1[28]	mcasp0_aclkr_mux3		gpmc_dir	mmc2_dat3	gpmc_csn6	mii2_col	gpmc_be1n
P9_13	29	0x874/074	UART4_TXD	31	gpio0[31]	uart4_txd_mux2		mmc2_sdcd	rmii2_rxerr	gpmc_csn5	mii2_rxerr	gpmc_wpn
P9_14	18	0x848/048	EHRPWM1A	50	gpio1[18]	ehrpwm1A_mux1		gpmc_a18	mmc2_dat1	rgmii2_td3	mii2_txd3	gpmc_a2
P9_15	16	0x840/040	GPIO1_16	48	gpio1[16]	ehrpwm1_tripzone_input		gpmc_a16	mii2_txen	rmii2_tctl	gmii2_txen	gpmc_a0
P9_16	19	0x84c/04c	EHRPWM1B	51	gpio1[19]	ehrpwm1B_mux1		gpmc_a19	mmc2_dat2	rgmii2_td2	mii2_txd2	gpmc_a3
P9_17	87	0x95c/15c	I2C1_SCL	5	gpio0[5]			pr1_uart0_txd	ehrpwm0_synci	I2C1_SCL	mmc2_sdwp	spi0_cs0
P9_18	86	0x958/158	I2C1_SDA	4	gpio0[4]			pr1_uart0_rxd	ehrpwm0_tripzone	I2C1_SDA	mmc1_sdwp	spi0_d1
P9_19	95	0x97c/17c	I2C2_SCL	13	gpio0[13]		pr1_uart0_rts_n	spi1_cs1	I2C2_SCL	dcan0_rx	timer5	uart1_rtsn
P9_20	94	0x978/178	I2C2_SDA	12	gpio0[12]		pr1_uart0_cts_n	spi1_cs0	I2C2_SDA	dcan0_tx	timer6	uart1_ctsn
P9_21	85	0x954/154	UART2_TXD	3	gpio0[3]	EMU3_mux1		pr1_uart0_rts_n	ehrpwm0B	I2C2_SCL	uart2_txd	spi0_d0
P9_22	84	0x950/150	UART2_RXD	2	gpio0[2]	EMU2_mux1		pr1_uart0_cts_n	ehrpwm0A	I2C2_SDA	uart2_rxd	spi0_sclk
P9_23	17	0x844/044	GPIO1_17	49	gpio1[17]	ehrpwm0_synco		gpmc_a17	mmc2_dat0	rgmii2_rxdv	gmii2_rxdv	gpmc_a1
P9_24	97	0x984/184	UART1_TXD	15	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd		I2C1_SCL	dcan1_rx	mmc2_sdwp	uart1_txd
P9_25	107	0x9ac/1ac	GPIO3_21	117	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7	EMU4_mux2	mcasp1_axr1	mcasp0_axr3	eQEP0_strobe	mcasp0_ahclkx
P9_26	96	0x980/180	UART1_RXD	14	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd		I2C1_SDA	dcan1_tx	mmc1_sdwp	uart1_rxd
P9_27	105	0x9a4/1a4	GPIO3_19	115	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5	EMU2_mux2	mcasp1_fsx	mcasp0_axr3	eQEP0B_in	mcasp0_fsr
P9_28	103	0x99c/19c	SPI1_CS0	113	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3	eCAP2_in_PWM2_out	spi1_cs0	mcasp0_axr2	ehrpwm0_synci	mcasp0_ahclkr
P9_29	101	0x994/194	SPI1_D0	111	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1	mmc1_sdcd_mux1	spi1_d0		ehrpwm0B	mcasp0_fsx
P9_30	102	0x998/198	SPI1_D1	112	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2	mmc2_sdcd_mux1	spi1_d1		ehrpwm0_tripzone	mcasp0_axr0
P9_31	100	0x990/190	SPI1_SCLK	110	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0	mmc0_sdcd_mux1	spi1_sclk		ehrpwm0A	mcasp0_aclkx
P9_32			VADC									
P9_33			AIN4									
P9_34			AGND									
P9_35			AIN6									
P9_36			AIN5									
P9_37			AIN2									
P9_38			AIN3									
P9_39			AIN0									
P9_40			AIN1									
P9_41A	109	0x9b4/1b4	CLKOUT2	20	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16	timer7_mux1	clkout2	tclkin		xdma_event_intr1
P9_41B		0x9a8/1a8	GPIO3_20	116	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6	emu3	Mcasp1_axr0		eQEP0_index	mcasp0_axr1
P9_42A	89	0x964/164	GPIO0_7	7	gpio0[7]	xdma_event_intr2	mmc0_sdwp	spi1_sclk	pr1_ecap0_ecap_capin_apwm_o	spi1_cs1	uart3_txd	eCAP0_in_PWM0_out
P9_42B		0x9a0/1a0	GPIO3_18	114	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4		Mcasp1_aclkx	Mcaspo_axr2	eQEP0A_in	Mcasp0_aclkr
P9_43			GND									
P9_44			GND									
P9_45			GND									
P9_46			GND									

Anexo 2: Especificación de los Procesadores del Arduino YUN

Microcontrolador Arduino

Procesador	**ATmega32u4**
Voltaje de Funcionamiento	5V
Voltaje de las Entradas	5V
Pines de Entrada/Salida Digital	20
Canales PWM	7
Canales Entradas Analógicas	12
Amperaje de Corriente Continua de los Pines Digitales	40 mA
Amperaje de Corriente Continua de los Pines de 3.3V	50 mA
Memoria Flash	32 KB (4KB son para el Bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Velocidad de Reloj	16 MHz

Microprocesador Linux

Procesador	**Atheros AR9331**
Arquitectura	MIPS @400MHz
Voltaje de Funcionamiento	3.3V
Ethernet	IEEE 802.3 10/100Mbit/s
WiFi	IEEE 802.11b/g/n
USB Typo A	2.0 Host
Lector de Tarjetas	Solo Tarjetas Micro-SD
RAM	64 MB DDR2
Memoria Flash	16 MB

Anexo 3: DataSheet Chronodot DS3231, Registros Accesibles a Través de I2C

En la siguiente table se aprecia cómo se codifica la fecha en las zonas de memoria y el mapeo de los comandos para activar alarmas y establecer parámetros de configuración. En la aplicación se acceden a las siete primeras posiciones de memoria y se decodifica la información para transformala en en un formato de fecha comprensible para la aplicación.

Figure 1. Timekeeping Registers

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

Note: *Unless otherwise specified, the registers' state is not defined when power is first applied.*

* El DS3231 es accesible a través de la dirección 0x68 del bus I2C.