Entrega de código (2/2) - TFC de Sistemas empotrados

Diseño e implementación de un:

Sistema de Alarma de Incendios

José Ignacio Quintana Ruiz
Ingeniería Técnica de Sistemas
UOC

Tutor: Jordi Bécares Ferrés

Junio de 2.011

Índice

1. Descripción de la aplicación.....2
2. Descripción del trabajo realizado.....3
3. Desarrollo pendiente.....8
4. Instrucciones de ejecución 8
5. Autoevaluación 8
6. Bibliografía, fuentes de información y referencias 9

1. Descripción de la aplicación

La aplicación "***Sistema de Alarma de incendios***" utiliza una red de sensores inalámbricos ***WSN* cou24** para la monitorización de las condiciones ambientales y alarmas de incendios de un edificio.

En esta fase, continuamos con el desarrollo del software necesario para alcanzar los requerimientos del sistema de alarma de incendios.Se adjunta el diagrama general de funcionamiento del sistema de incendios.

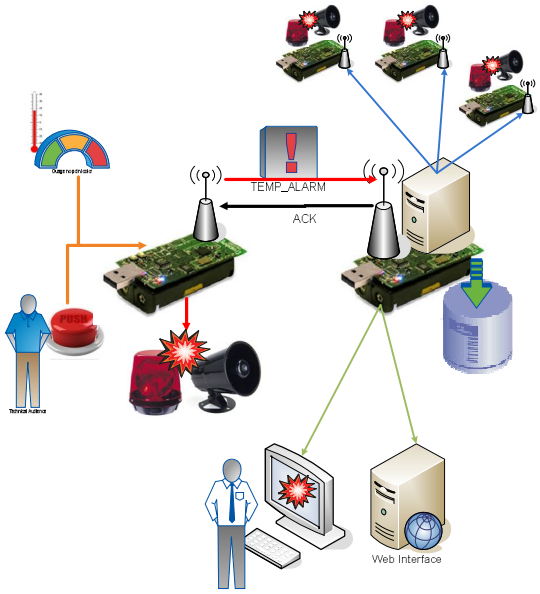


Ilustración 1 – Sistema de alarma de incendios

2. Descripción del trabajo realizado

El trabajo realizado hasta el momento persigue ese objetivo, y se ha centrado en la creación de bloques de código que permitan el desarrollo modular de la aplicación. De esta manera conseguimos la independencia entre los diferentes subsistemas permitiendo que cada uno de ellos pueda ser ampliado o rectificado sin necesidad de reescritura del resto.

Los diferentes bloques funcionales de software que se entregan se clasifican por funcionalidad:

Mote COU24/Z1:

De manera general se proporciona log de actividad mediante el puerto serie. Los parámetros son **19200 8N1** para la plataforma COU24 y **115200 8N1** para la Z1

Ejemplo de compilación/instalación en cada uno de los directorios:

```
COU24$ make cou24 install.34
COU24$ meshprog -t /dev/ttyUSB0 -f ./build/cou24/main.srec.out-34

Z1$    make z1 install.153 bsl,/dev/ttyUSB1
```

- El dispositivo estará conectado en /dev/ttyUSBx
- 34/153 son los NODE_ID con que se identificarán en la red 802.15.4 mediante TinyOS

Ilustración 2 – Ejemplo de compilación e instalación de código en mote COU24/Z1

./mote/NodeStationAppC: Funcionalidad de mote sensor remoto.

./mote/BaseStationAppC: Funcionalidad de mote local conectable a USB, coordinador de la red de sensores y arbitraje de la red mediante el enlace al software desarrollado al efecto.

Estos programas finales, hacen uso de los componentes diseñados al efecto, ubicados en el directorio **./mote/components** y cuyas cabeceras de control residen en **./mote/include**. Algunas opciones especiales de funcionamiento, se especificarán para cada caso concreto. Los componentes se detallan a continuación:

./mote/components/AlarmRadio[CP].nc

Envío de alarmas y datos de medida con y sin ACK dependiendo del valor del campo **is_alarm** del mensaje enviado de tipo **SCX_ALARM**. Proporciona notificaciones de envio, recepción y recepción de ACK y también medida del parámetro RSSI (*Receive Signal Strength Indicator*) en decibelios del paquete recibido.

./mote/components/AskConfigRadio[CP].nc

Solicitud de parámetros de configuración mediante broadcast desde el sensor al nodo coordinador.

./mote/components/ConfigRadio[CP].nc

Envío de configuración de la red del nodo controlador a los sensores. Se generan en dos situaciones: cuando el nodo controlador recibe un cambio de parámentros, lo envía por broadcast a toda la red y en segundo lugar, cuando un nodo se inicia y solicita configuración por *broadcast* mediante **AskConfigRadio** al nodo coordinador, la respuesta del controlador es *unicast* hacia el nodo peticionario.

./mote/components/BeaconCtrl[CP].nc

Control de señalización de baliza visual y sonora. La baliza sonora sólo se activa si definimos el parámetro **#define HAD_SIREN_HARDWARE** en el fichero de cabecera **./mote/include/beacon_def.h**
Requiere hardware conectado al mote según el diagrama:

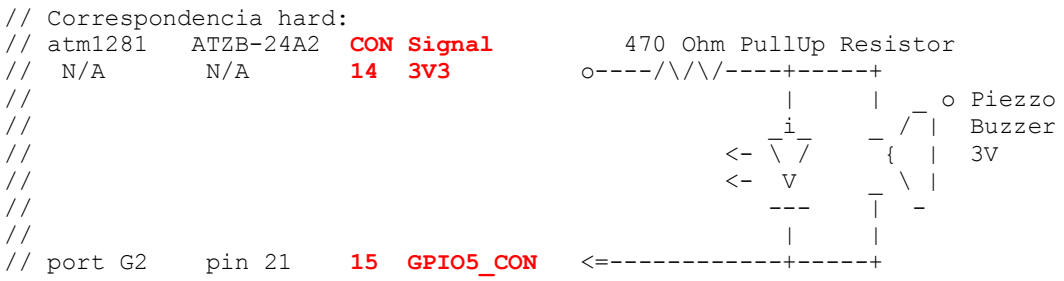


Ilustración 3 – Diagrama hardware Sirena

Significado	Led	Ciclo
Cobertura Red OK	verde	Flash x1
Cobertura Red NO OK	verde	Flash x2
Modo debug activado	naranja	Flash x1
Nivel batería bajo	naranja	Flash x2
Fuera de cobertura	naranja	Flash x3
Alarma detectada	naranja	Fijo
	rojo	Flash x2
Alarma desactivada	rojo	Flash x1
Alarma recibida	rojo	Fijo 50%
Alarma general	rojo	Fijo

Ilustración 4 – Estado baliza

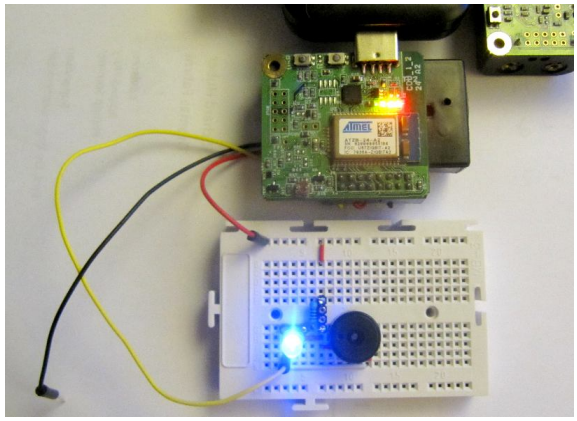


Ilustración 5 – Prototipo de baliza.

./mote/components/Siren[CP].nc

Control del hardware adicional externo al mote descrito en el módulo anterior.

./mote/components/NetProbe[CP].nc

Escaneo de los canales 11 hasta el 26 hasta encontrar el nodo coordinador y descarga de la configuración de red posterior. A veces falla la detección por interferencias en la banda de 2.4GHz. El canal que mejor resultado me da es el 26.

./mote/components/ParseBSerial[CP].nc

Trasformación de mensajes serie a estructuras de mensajes radio.

./mote/components/SensorAdc[CP].nc

Control del ADC para medición de temperatura, tensión de batería y luminosidad. Es un control que en caso de estar ocupado internamente replanifica una tarea con reintentos.

./mote/components/Button[CP].nc

Control de botón de usuario para disparo de alarma manual.

./mote/components/SensorHall[CP].nc

Activación de alarmas mediante campo magnético (2 detecciones activar alarma / 4 desactivar)

./mote/components/SerialConsole[CP].nc

Gestión de detección de comandos por el puerto serie y notificación a la aplicación.

./mote/components/Wdt[CP].nc

Gestión del watchdog interno en plataforma COU24. El rearme del watchdog se realiza ejecutando una tarea (task/post) disparada por timer para tener la seguridad de que el mote no se ha colgado.

Adaptación y uso de componentes bajo licencia LGPL localizados en "tiny-2.x-contrib/diku/freescale":

```
./mote/components/ConsoleC.nc
./mote/components/ConsoleDebugM.nc
./mote/components/ConsoleInput.nc
./mote/components/ConsoleOutput.nc
./mote/components/ConsoleP.nc
```

El diagrama de mensajes a implementado es el siguiente:

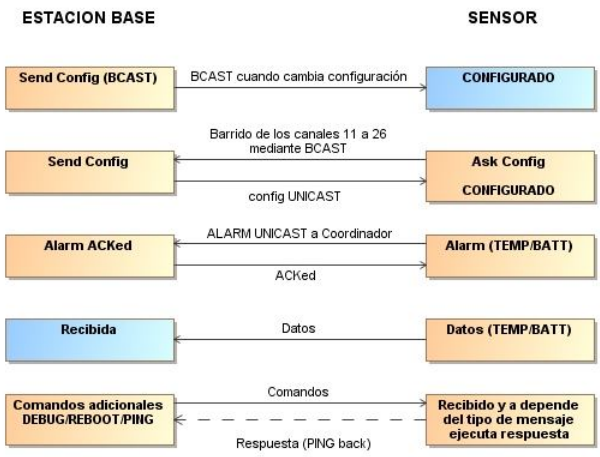


Ilustración 6 – Flujo final de mensajes

- Directorio ***PC:*** Hay dos directorios, ***Linux*** y ***Windows***.

En el directorio **linux** encontramos el software redirector de puerto serie a TCP: **./PC/linux/Serial2TCP/remserial-1.4**

```
COU24$ ./remserial -d -p 2000 -s "19200 raw icanon cs8 sane" /dev/ttyUSB0 &
Z1$    ./remserial -d -p 2000 -s "115200 raw icanon cs8 sane" /dev/ttyUSB0 &
```

Ilustración 7 – Ejemplo de ejecución redirector Serie-TCP

También encontraremos scripts de ayuda en la programación, debug y configuración del entorno TinyOS

En el directorio **Windows** encontramos la aplicación de control. Podemos seleccionar el host/IP al que conectar, pero el puerto TCP usado es el **2000**. En caso de necesitar cambio del puerto, hay que recompilar o configurarlo para usar un puerto guardado en el fichero de configuración **mote.ini**. He proporcionado binarios para **i386** y **x64.** Se adjuntan capturas del programa:

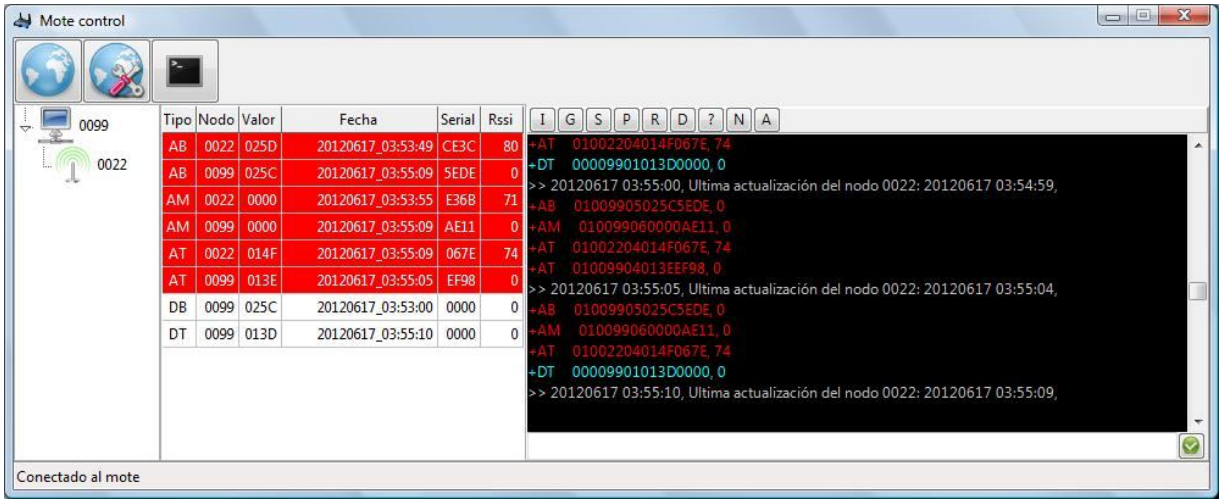
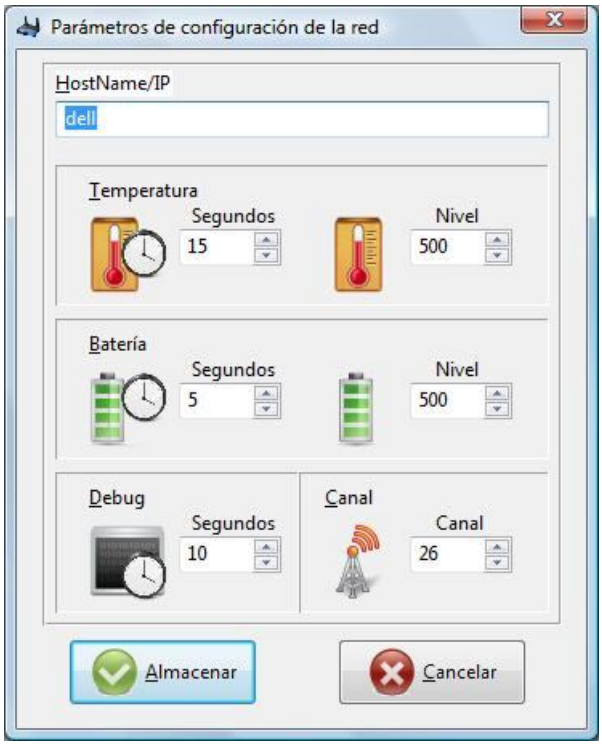
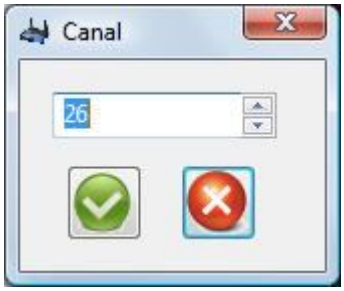
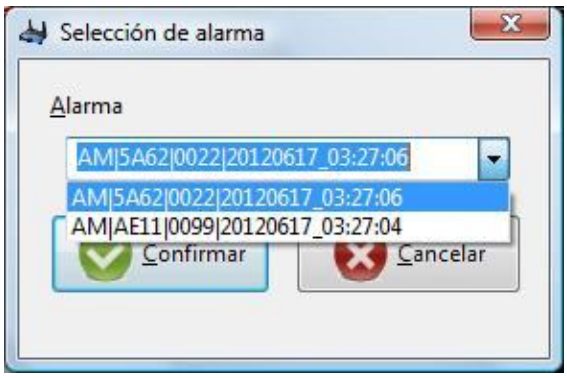


Ilustración 8 – Pantalla principal de la aplicación de control y configuración







3. Desarrollo pendiente

Ninguno

4. Instrucciones de ejecución

Se han descrito anteriormente en los diferentes módulos. El único cuidado especial que se debe tener es cuando se ejecuta la redirección serie. Funciona como el software ***Listen*** de java de tinyOS pero no requiere java y puede ser una ventaja si el mote controlador y software redirector se ejecuta en un router inalámbrico.

5. Autoevaluación

Bueno, creo que las palabras huelgan.

No he llegado a los plazos fijados para los diferentes hitos.

Han contribuido varios factores:

- Asignación en el trabajo a un proyecto de lanzamiento que me exigía más tiempo y me dejaba menos para dedicar al proyecto.

- Problemas inesperados con componentes que *"parecían"* funcionar correctamente si lo hacían independientemente pero luego juntos tenían interacciones inesperadas (botón/detector hall).

- Cuelgues y reinicios inesperados e intento de localizar el problema. Esto ha motivado reescribir desde cero varias veces los módulos de comunicación, los de gestión del puerto serie y notificación de comandos. Erróneamente creí que el problema se debía a la función '*memcpy()*' y reescribí todo para que no lo usase. Posteriormente creí que se debía al componente de escritura/lectura del puerto serie ya que usa llamadas síncronas y yo los usaba en eventos async (ver warnings en compilación de NodeStationAppC). En este caso, reescribí todo el código para que los eventos llamasen mediante post a funciones sin problemas de concurrencia (las ejecuciones mediante '*post*/task' son sincronas.

  Todo hasta que localicé el bug; lo encontré en el módulo ***ParseBSerialP.nc*** en algunas de las funciones que hacían uso de *memcpy()* con un buffer intermedio insuficiente y que motivaban reinicios aleatorios de los mote. El problema principal para localizar el bug ha sido que **sólo** fallaban algunas de las funciones de conversión y otras funcionaban perfectamente, haciendo que buscase en otros lugares es dichoso bug.

6. Bibliografía, fuentes de información y referencias

[1] **UOC** (2010). "*Arp@:Embedded Systems Lab@Home"* [fecha de consulta: 2011-09-19]. <http://cv.uoc.edu/app/mediawiki14/>

[2] **Varios** (2010). "*TinyOS Documentation Wiki"* [fecha de consulta: 2011-09-21]. <http://docs.tinyos.net/tinywiki/index.php/Main_Page>

[3] **LEVIS, Philip y GAY, David** (2009). "*TinyOS Programming"*. Cambridge University Press. 282p. ISBN: 978-0521896061

[4] **FALUDI, Robert** (2010). "*Building Wireless Sensor Networks"*. O'Reilly Media. 322p. ISBN: 978-0596807733

[5] **Zolertia Z1** (2011). "*Z1 Platform*". [fecha de consulta: 2011-10-25]. http://www.zolertia.com/ti

7. Software y licencias usadas

Plataforma desarrollo Mote

Linux (Osian): Ya no está disponible para descarga.
Licencia: >> BSD License vía http://en.wikipedia.org/wiki/OSIAN

TinyOS: http://www.tinyos.net/
Licencia: >> New BSD license:
http://www.opensource.org/licenses/bsd-license.php

Plataforma desarrollo Aplicación de control

Lazarus: http://www.lazarus.freepascal.org/ | http://sourceforge.net/projects/lazarus/files
Entorno de programación multiplataforma clónico a delphi basado en FreePascal

```
Licencia:    http://wiki.lazarus.freepascal.org/Lazarus_Faq#Licensing
             >> LCL is licensed under the LGPL with an exception, which
             >> allows you to link to it statically without releasing
             >> the source of your application.
```

Componentes para Lazarus

http://wiki.freepascal.org/lNet → **Conexión TCP**

```
Licencia:    >> lNet units (units in lib and lazaruspackage
             >> directories) are licensed under a modified LGPL
             >> license. See file lnet/LICENSE and lnet/LICENSE.ADDON.
```

http://wiki.freepascal.org/UniqueInstance → **Control de una única ejecución**

```
Licencia:    >> Modified LGPL
```

http://wiki.freepascal.org/CmdLine → **Emulación de consola (Debug)**

```
Licencia:    >> LGPL
```

Otro Software

http://lpccomp.bc.ca/remserial → **Redirección de puerto serie a TCP**

```
Licencia:    >> GNU GPL
```