PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA
MAESTRÍA EN REDES DE COMUNICACIONES

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PARA EL MONITOREO Y CUANTIFICACIÓN DE FLUJOS DE LODO EN LOS VOLCANES COTOPAXI Y TUNGURAHUA BASADO EN UNA RED INALÁMBRICA DE SENSORES

TRABAJO PREVIO A LA OBTENCIÓN DEL TÍTULO DE MÁSTER EN REDES DE COMUNICACIONES

MARCILLO LARA PABLO DANIEL

DIRECTOR: Ph.D. IVÁN BERNAL

Quito, Septiembre 2014

DECLARACIÓN

Yo PABLO DANIEL MARCILLO LARA, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional, y que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo los derechos de propiedad intelectual correspondientes a este trabajo, a la Pontificia Universidad Católica del Ecuador, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normativa institucional vigente.

_____

Pablo Marcillo Lara

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Pablo Marcillo Lara bajo mi supervisión.

_____

Ph.D. Iván Bernal

Director del Proyecto

DEDICATORIA

A mi madre y a mi padre que han sido ejemplo de probidad, tolerancia, respeto y cariño desmedido.

A mis hermanos:

A Verónica ejemplo de cordura, generosidad y amor.
A Omar ejemplo de sabiduría, esperanza y apoyo incondicional.
A Andrea ejemplo de dedicación, coraje y prolijidad.
Y a Alejandro ejemplo de constancia, alegría y pasión por la vida.

“Sometimes life hits you in the head with a brick. Don't lose faith. I'm convinced that the only thing that kept me going was that I loved what I did. You've got to find what you love. And that is as true for your work as it is for your lovers. Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do. If you haven't found it yet, keep looking. Don't settle. As with all matters of the heart, you'll know when you find it. And, like any great relationship, it just gets better and better as the years roll on. So keep looking until you find it. Don't settle.”

Steve Jobs

Parte del discurso dado en la Universidad de Stanford el 12 de Junio de 2005.

AGRADECIMIENTO

A mis padres y hermanos por la confianza depositada en mí, por sus palabras de aliento y sobretodo por su cariño.

A los profesores que dirigieron mi proyecto de investigación. A Iván, Juan Francisco y Carlos por el apoyo brindado durante el desarrollo del mismo.

Al personal del Instituto Geofísico por su apertura e interés en este proyecto de investigación.

A Carlos por su tiempo, por sus acertados consejos, por su apoyo desinteresado e incondicional y principalmente por su amistad.

A María por sus reprimendas y palabras de aliento, por su cariño y amistad sincera.

A todos mis amigos, especialmente a mi hermano y amigo de toda la vida, a Daniel por su amistad inigualable.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El gran desarrollo e importancia que tienen actualmente las redes inalámbricas ha significado un aumento considerable tanto de aplicaciones orientadas a este tipo de tecnología así como de personas interesadas en la misma. La acogida de esta tecnología, sin duda, se debe a las ventajas que ésta ofrece versus otras alternativas; las principales son: flexibilidad, escalabilidad, movilidad, simplicidad y rapidez de instalación y también los bajos costos de mantenimiento. En vista de las ventajas de las redes inalámbricas, su uso se ha diseminado por diferentes áreas del conocimiento.

El Instituto Geofísico (IG) de la Escuela Politécnica Nacional, como la institución a cargo de la vigilancia y monitoreo de volcanes, en su afán de incluir nuevas tecnologías en el monitoreo volcánico, ha trabajado en conjunto con otras instituciones en interesantes iniciativas, como por ejemplo, la propuesta de la Universidad de Harvard y la Universidad de Carolina del Norte que planteó el desarrollo de una Red Inalámbrica de Sensores (WSN[1]) que permita colectar información de señales sísmicas y acústicas.

La instrumentación volcánica, como herramienta clave para el monitoreo de desastres naturales como: erupciones volcánicas, avalanchas e inundaciones, ha ayudado en gran medida a la difícil tarea, de científicos e investigadores, de monitorear este tipo de eventos. Por tal motivo, el presente trabajo plantea el diseño e implementación de un sistema basado en una red inalámbrica de sensores que permita el monitoreo y cuantificación de flujos de lodo producidos por erupciones volcánicas. Tal red se ha concebido como un grupo de sensores (sísmicos, de altura, de temperatura y de humedad), módulos inalámbricos (de procesamiento, control, y de visualización) y dispositivos (cámaras de alta resolución provistas de potentes iluminadores infrarrojos); misma que está basada en la plataforma inalámbrica imote2[2], la cual integra el estándar de comunicaciones de corto alcance IEEE[3] 802.15.4.

[1] Wireless Sensor Network
[2] Nombre dado a la plataforma para WSN desarrollada por Crossbow Inc.
[3] Institute of Electrical and Electronics Engineers. Instituto de Ingenieros Eléctricos y Electrónicos. Es una asociación internacional sin fines de lucro, integrada por profesionales de las ramas de la tecnología, dedicados a establecer y promulgar estándares relacionados a la tecnología e innovación.

PRESENTACIÓN

El presente proyecto titulado, "Diseño e implementación de un sistema para el monitoreo y cuantificación de flujos de lodo en los volcanes Cotopaxi y Tungurahua, basado en una red inalámbrica de sensores", establece un modelo de infraestructura de comunicaciones para transportar y procesar información generada por una red de monitoreo y cuantificación de flujos de lodo.

El objetivo general del proyecto es:

- Diseñar e implementar un sistema para el monitoreo y cuantificación de flujos de lodo y escombros para los volcanes Cotopaxi y Tungurahua, basado en una red inalámbrica de sensores.

Mientras que los objetivos específicos son los siguientes:

- Desarrollar un módulo inalámbrico para procesamiento de información, que permita la comunicación con el resto de módulos que integran la red inalámbrica, la interacción con un receptor GPS, y el envío de información hacia una interfaz de comunicación basada en el estándar RS-232.

- Desarrollar un módulo inalámbrico para monitoreo visual, que interactúe con una cámara de video de alta resolución y un iluminador infrarrojo.

- Desarrollar un módulo inalámbrico de sensores, que permita la lectura de distintos tipos de sensores (sísmico, altura, temperatura y humedad).

- Diseñar una red de acceso, que permita la transmisión de información generada por la red de sensores hacia el Centro de Datos del IG.

- Desarrollar un módulo de software para el monitoreo y cuantificación de flujos de lodo y escombros e integrarlo al ya existente Sistema de Adquisición y Monitoreo Integrado (SAMI).

El desarrollo del presente trabajo se halla dividido en 5 capítulos. A continuación se presenta una breve descripción de cada uno de estos capítulos.

El Capítulo 1 presenta una breve introducción de los sistemas de alerta temprana, un estudio detallado de cada uno de los estándares que integran el grupo de trabajo IEEE 802.15 y finalmente un análisis de todo lo concerniente a redes inalámbricas de sensores como plataformas, lenguajes de programación, sistemas operativos, tecnologías, y aplicaciones.

El Capítulo 2 presenta los diferentes lineamientos en diseño usados para la construcción de los módulos que conforman la red inalámbrica de sensores, un estudio de radio propagación de la red de acceso y un análisis de los medidas a tomar para integrar la red de acceso a la red microondas.

El Capítulo 3 presenta una breve descripción del Sistema de Adquisición y Monitoreo Integrado (SAMI), los lineamientos usados para el diseño e implementación de un módulo de software para la red de monitoreo y cuantificación de flujos de lodo, y las estrategias para la integración de este con el SAMI.

El Capítulo 4 además de presentar varias imágenes que dan una idea de la forma en que se desarrollaron las pruebas de laboratorio, presenta los resultados obtenidos de tales pruebas, mismas que fueron efectuadas a cada uno de los módulos y al sistema en conjunto.

Y finalmente, el Capítulo 5 presenta las conclusiones que resultaron del desarrollo del presente trabajo y las recomendaciones propuestas para el mismo.

CAPÍTULO 1

MARCO TEÓRICO

1.1 Sistemas de alerta temprana

Un Sistema de Alerta Temprana (SAT) es un conjunto de herramientas y procesos, implementados en organizaciones o instituciones afines, para aplacar los efectos causados por desastres naturales. Un SAT se basa en el conocimiento y comprensión del riesgo, el monitoreo del fenómeno, la difusión de alertas a los involucrados y en la preparación que tengan éstos para actuar [1]. Los desastres naturales, en su mayoría, son fenómenos inevitables para el ser humano, lo cual obliga e impulsa la creación de sistemas que permitan mitigar sus efectos. La Figura 1-1 muestra el procedimiento de un SAT, el cual es usado para la ocurrencia de flujos de lodo y escombros en el volcán Tungurahua.

Aplicando la definición de un SAT a la realidad del país, las personas, instituciones y organizaciones que son parte de los SATs son: los dirigentes de municipios y jefes de juntas parroquiales afectados; las poblaciones que por su ubicación se ven afectadas directamente por el fenómeno; la Secretaria Nacional de Gestión de Riesgo (SNGR) como encargada de garantizar la protección de personas ante desastres naturales; y, el Instituto Geofísico como el encargado de proveer información técnica y especializada del estado de un fenómeno.

Fig. 1-1 Proceso de un SAT.

Fuente: [2]

Partiendo del hecho de que un SAT provee información veraz e instantánea, de manera que la toma de una u otra decisión sea ágil y a tiempo para así evitar pérdidas humanas y en lo posible pérdidas materiales, el IG experimentó casos exitosos con su sistema de detección de lahares. Dos de estos casos son los dados en los volcanes Tungurahua y Reventador [3]. En el primer caso, se instalaron AFMs[4] en las quebradas Palma Urcu y Vazcún, lo cual permitió generar alertas tempranas con un margen de tiempo de 21 minutos, y un acierto cercano al 100 %, antes de que el flujo alcance las zonas de influencia.

En el segundo caso, tras la erupción del Volcán Reventador en el año 2002, el volcán experimentó gran actividad, lo cual produjo continuos flujos de lodo y escombros que descendían por los ríos Marker y Reventador. Estos pusieron en peligro la vida de seres humanos que trabajaban en ese mismo año en la construcción de un tramo del Oleoducto de Crudos Pesados (OCP). La construcción de este tramo conllevó dificultades y desafíos, ya que se encontraba en un sector de alto riesgo de ocurrencia de flujos. Es así que con el objetivo de evitar

[4] Acostic Flow Monitor. Sistema diseñado por la USGS para el monitoreo y detección de flujos de lodo y escombros (lahares).

perdidas humanas y materiales, se instalaron AFMs en tales ríos. En este caso la generación de alertas alcanzó al igual que en el primer caso un acierto cercano al 100 %.

1.2 Redes Inalámbricas de Área Personal (WPAN)

1.2.1 Introducción

Las WPAN[5] son redes inalámbricas usadas para interconectar dispositivos personales como teléfonos celulares, PDAs[6] o equipos portátiles. Este tipo de redes se hallan definidas por un espacio personal de operación (POS[7]). Un POS está definido como el espacio de una persona u objeto que abarca una distancia referencial de 10 m en todas las direcciones [4]. El propósito de una WPAN, en un principio, fue la conectividad entre dispositivos encontrados en una oficina como laptops, escáneres, impresoras y otros, para luego convertirse en un medio de conectividad entre dispositivos personales como teléfonos celulares y equipos portátiles.

La IEEE, a través de su grupo de trabajo 802.15, estableció un estándar para WPAN denominado IEEE 802.15, mismo que es un compendio de cuatro estándares generados por los grupos de trabajo 802.15.1, 802.15.2, 802.15.3 y 802.15.4. Los estándares generados abarcan especificaciones de las capas de control de acceso al medio (MAC) y física (PHY); modelos de coexistencia entre WPAN y WLAN, y estándares para redes de alta y baja velocidad.

1.2.2 IEEE 802.15

El estándar IEEE 802.15 lo integran los estándares IEEE 802.15.1 el cual define las especificaciones de las capas de control de acceso al medio (MAC) y física (PHY) para WPAN

[5] Wireless Personal Area Network
[6] Personal Digital Assistance. Asistente Digital Personal. Es un dispositivo portátil que integra capacidades de cómputo y almacenamiento y esta dirigido para uso personal o de negocios.
[7] Personal Operating Space

de tasas de transmisión medianas, siendo este estándar la base de lo que es Bluetooth[8]; IEEE 802.15.2 el cual define la coexistencia de WPAN y WLAN en la banda de frecuencia no licenciada; IEEE 802.15.3 el cual define las especificaciones de las capas MAC y PHY para WPAN de alta velocidad, y finalmente el estándar IEEE 802.15.4 el cual define las especificaciones para WPAN de baja velocidad, siendo este estándar la base de la tecnología ZigBee[9]. A continuación se detalla cada uno de los estándares mencionados anteriormente.

1.2.2.1 IEEE 802.15.1

El estándar IEEE 802.15.1-2005 define los mecanismos necesarios para la mejor coexistencia de dispositivos WPAN con dispositivos basados en el estándar IEEE 802.11b. Tanto la primera versión (IEEE 802.15.1-2002) como su revisión (IEEE 802.15.1-2005) están basadas en el desarrollo de la tecnología inalámbrica Bluetooth. A partir de este hecho, al estándar IEEE 802.15.1 se lo conoce comúnmente como Bluetooth.

El diseño de la pila de protocolos de la tecnología Bluetooth difiere un poco del diseño del resto de estándares IEEE, ya que define componentes opcionales sobre la capa física (PHY[10]) y la de control de acceso al medio (MAC[11]). Este diseño permite la integración de nuevos componentes para aplicaciones específicas y definidas por la capa de aplicación. La Figura 1-2 presenta la pila de protocolos de la tecnología Bluetooth.

[8] Nombre de la tecnología inalámbrica de corto alcance desarrollada por la empresa Ericsson.
[9] Tecnologia para WPAN que solventa requerimientos de bajo costo de implementacion y bajo consumo de energia.
[10] Acrónimo de Physical. La Capa Física dentro del modelo OSI se encarga de la definición de especificaciones de hardware, ejecución de funciones de codificación y señalización y de la transmisión de datos.
[11] Media Access Control. Control de Acceso al Medio. La capa MAC dentro del modelo OSI se encarga de establecer mecanismos para gestionar el medio de trasmisión y así evitar conflictos.

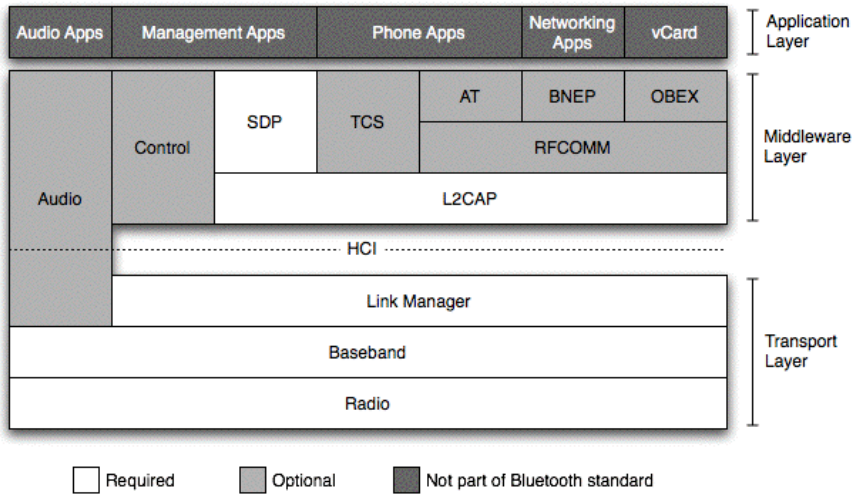


Fig. 1-2 Pila de protocolos de la tecnología Bluetooth.

Fuente: [5]

Haciendo una analogía, la capa de transporte de la tecnología Bluetooth corresponde a las capas PHY y subcapa MAC del modelo OSI. La *Capa Radio* establece o determina la frecuencia, potencia y modulación usadas por la tecnología. Bluetooth opera en la banda ISM[12] de 2.4 GHz, banda que se encuentra en el rango de 2400 a 2483.5 MHz. La banda se halla dividida en 79 canales de 1 MHz, más 2 bandas de guarda que se encuentran entre 2400-2401.5 y 2480.5-2483.5 MHz. Bluetooth usa el método FHSS[13] para transmitir señales de radio. En cuanto a la modulación, el estándar usa GFSK[14] con una tasa física de 1Mbps, sin embargo, versiones recientes de Bluetooth usan modulaciones π/4-DQPSK[15] y 8DPSK[16] alcanzando tasas de 2 y 3 Mbps, respectivamente.

La tecnología Bluetooth incluye en su especificación tres grupos o clases de dispositivos, tal diferenciación se la hace en base a la potencia de transmisión, lo cual afecta también la distancia

[12] Industrial Scientific Medical. Es un rango de frecuencias no licenciadas usado para fines industriales, científicos y médicos.
[13] Frequency Hoping Spread Spectrum. Método de transmisión de señales de radio que consiste en cambiar a una portadora a un canal de frecuencia, de varios disponibles, usando una secuencia pseudo aleatoria.
[14] Gaussian Frequency Shift Keying. Tipo de modulación que usa un filtro Gausiano para suavizar desviaciones de frecuencia. Esta modulación establece una desviación positiva para un 1 lógico y una negativa para un 0 lógico.
[15] Differential Quadrature Phase Shift Keying. Tipo de modulación del tipo de fase continua (PCM) en la cual los pulsos son aplicados a un modulador de fase.
[16] Differential Phase Shift Keying. Tipo de modulación en la que la información binaria de entrada está compuesta por la fase entre dos elementos sucesivos de señalización y no por la fase absoluta.

de cobertura. La *Clase 1* usa de 1 a 100 mW, la *Clase 2* de 0.25 a 2.5 mW y la *Clase 3* usa 1 mW; los dispositivos de *Clase 2* y *3* son capaces de variar su potencia de transmisión. En cuanto a la cobertura, los dispositivos de *Clase 1*, *2* y *3* alcanzan distancias máximas de 100, 10 y 1 m, respectivamente.

Los dispositivos Bluetooth forman redes denominadas *piconets* y *scaternets*. Una *piconet* es una red formada por un nodo denominado *maestro* y hasta 7 nodos *esclavos*. La integración de dos o más *piconets* forma una *scaternet*. La comunicación dentro de una *piconet* está basada en un esquema TDD[17], además, el tiempo es dividido en ranuras de 625 μs para alcanzar un total de 1600 ranuras/s. En un esquema TDD las ranuras impares son usadas para la comunicación desde el *maestro* hacia el *esclavo*, mientras que las pares son usadas del *esclavo* hacia el *maestro*.

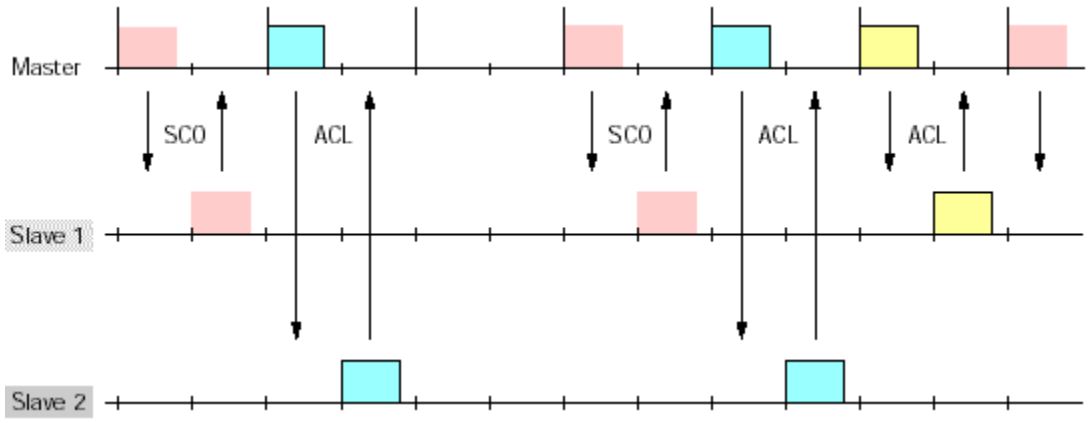


Fig. 1-3 Esquema TDD de Bluetooth.

Fuente: [6]

En lo que respecta a la transmisión de datos, ésta puede ser realizada de forma síncrona o asíncrona a través de los métodos mencionados a continuación. El método SCO[18] permite reservar ranuras para comunicaciones síncronas y el eSCO[19] permite reservar ranuras extras para

[17] Time Division Duplexing. Duplexación por División de Tiempo. Es un mecanismo que simula la comunicación full dúplex a partir de una comunicación half dúplex asignando ranuras de tiempo a las transmisiones de subida y bajada.
[18] Synchronous Connection Oriented: Síncrono Orientado a Conexión.
[19] Enhanced Synchronous Connection Oriented: Síncrono Orientado a Conexión Extendido.

retrasmisión de datos. El ACL permite transmitir durante las ranuras no reservadas, y los métodos ASB[20] y PSB[21] permiten al nodo *maestro* enviar datos de control a los nodos *esclavos*.

La *Capa Middleware* la integran varios componentes de software opcionales. Entre los más importantes se puede destacar los protocolos: L2CAP[22], el cual provee servicios orientados y no orientados a la conexión a protocolos de capas superiores; SDP[23], el cual suministra el medio para que aplicaciones clientes descubran los servicios provistos por aplicaciones de servidor; TCS[24], el cual proporciona una manera para realizar llamadas de audio entre dispositivos; BNEP[25], que permite la encapsulación de paquetes de varios protocolos de red, transportados por el protocolo L2CAP; y, el componente RFCOMM[26] el cual permite emular enlaces seriales sobre el protocolo L2CAP.

1.2.2.2 IEEE 802.15.2

Este estándar incluye varios mecanismos de coexistencia para redes WPAN y WLAN. Estos mecanismos están divididos en dos grupos, los colaborativos y los no colaborativos. La diferencia entre estos grupos de mecanismos radica en la existencia o no de un enlace de comunicación entre la WLAN y la WPAN. Un mecanismo colaborativo es mejor implementado en sistemas en los que dispositivos WLAN y WPAN se encuentran embebidos en un mismo equipo. La Figura 1-4 presenta una lista con los mecanismos de coexistencia definidos por el estándar.

[20] Active Slave Broadcast: Difusión para Esclavos Activos.
[21] Parked Slave Broadcast: Difusión para Esclavos Aparcados.
[22] Logical Link Control and Adaptation Protocol: Protocolo de Adaptación y Control del Enlace Lógico.
[23] Service Discovery Protocol: Protocolo de Descubrimiento de Servicio.
[24] Telephony Control Protocol: Protocolo de Control de Telefonía.
[25] Bluetooth Network Encapsulation Protocol: Protocolo Bluetooth de Encapsulación de Red.
[26] Radio Frequency Communication: Comunicación por Radio Frecuencia.

Name	Type	Clause/Annex
Alternating wireless medium access	collaborative	Clause 5
Packet traffic arbitration	collaborative	Clause 6
Deterministic interference suppression	collaborative	Clause 7
Adaptive interference suppression	non-collaborative	Clause 8
Adaptive packet selection	non-collaborative	Clause 9
Packet scheduling for ACL links	non-collaborative	Clause 10
Packet scheduling for SCO links	non-collaborative	Annex A
Adaptive frequency-hopping	non-collaborative	Annex B

Fig. 1-4 Mecanismos de coexistencia.

Fuente: [7]

1.2.2.3 IEEE 802.15.3

El estándar IEEE 802.15.3 se crea a partir del hecho de definir un conjunto de especificaciones para redes inalámbricas que integren características como bajo costo y bajo consumo de energía, altas tasas de transferencia y calidad de servicio. Este protocolo se amolda perfectamente al concepto de multimedia inalámbrico. A partir de la aprobación de la UWB[27] por parte de la FCC[28], se crea el sub grupo de trabajo 802.15.3a con el mismo objetivo del grupo 802.15.3 más la inclusión y estandarización de la UWB; sin embargo los miembros de este sub grupo no dieron paso a tal iniciativa, misma que fue vetada tiempo después.

A diferencia de los estándares IEEE 802.15.1 y IEEE 802.15.4, en los cuales las señales son enviadas durante periodos en el orden de los μs o ms usando una pequeña porción del espectro, UWB usa pulsos en el rango de los ρs y ηs usando una gran porción del espectro. La estrategia

[27] Ultra Wide Band: Banda ultra ancha.
[28] Federal Communication Commission. Agencia de los Estados Unidos que regula las comunicaciones por radio, televisión, satélite y cable.

mencionada anteriormente y que fue adoptada por UWB está basada en el Teorema de Shannon, que establece que la tasa de transferencia puede ser incrementada de forma más efectiva expandiendo el ancho de banda, que incrementando la potencia de transmisión. Debido a discordancias entre el grupo de trabajo 802.15.3a no se logró definir una sola estrategia UWB; es así que se definieron 2 estrategias DS-UWB[29] apoyado por el UWB Forum y MB-OFDM[30] apoyado por WiMedia.

1.2.2.3.1 DS-UWB

La estrategia DS-UWB usa un solo pulso que se da en dos de las bandas de frecuencia establecidas para tal propósito. DS-UWB define las bandas de 3.1-4.85 y 6.2-9.7 GHz. Con respecto a la modulación, DS-UWB puede ser implementada usando 4BOK[31] en el caso de que la calidad de la señal sea buena y BPSK[32] en el caso de que la calidad de la señal no sea del todo buena y no sea necesaria una tasa de transferencia alta. Es así que DS-UWB ofrece tasas de transferencia de 55 Mbps a 1.32 Gbps en la banda de 3.1-4.85 y 55Mbps a 2 Gbps en la banda de 6.2-9.7 GHz. La Figura 1-5 presenta el impulso DS-UWB.

[29] Direct Sequence Ultra Wide Band. Banda ultra ancha de secuencia directa.
[30] Multi Band Orthogonal Frequency Division Multiplexing. Multiplexacion por division de frecuencias ortogonales multi banda.
[31] 4-ary Bi-Orthogonal Keying. Uno de los dos tipos de modulación propuesto para la estrategia DS-UWB.
[32] Binary Phase Shift Keying. Tipo de modulación basada en PSK. BPSK implementa un esquema de 2 fases el cual permite codificar 1 bit por cada símbolo. Los cambios de fase se encuentran en 0 y 180°.

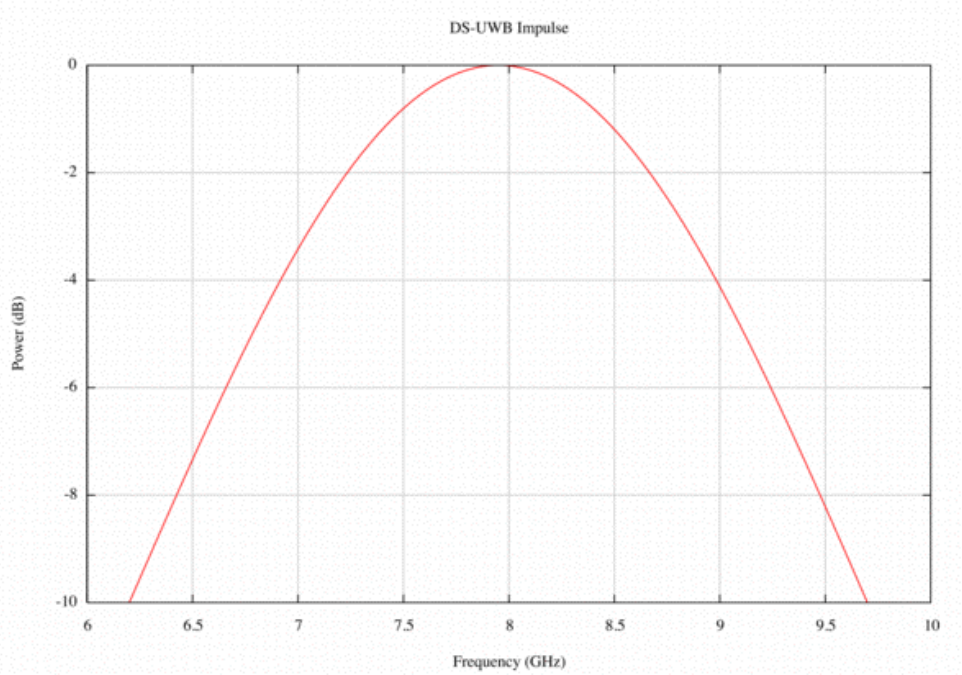


Fig. 1-5 Impulso DS-UWB

Fuente: [5]

1.2.2.3.2 MB-OFDM

A diferencia de US-UWB que usa un solo pulso en todo el espectro, MB-OFDM divide a la banda de 3.1-10.6 GHz en 14 sub bandas de 528 MHz; sub bandas por las cuales las señales MB-OFDM son transmitidas usando la técnica de saltar entre sub bandas durante periodos de tiempo fijos para así evitar interferencias. MB-OFDM divide al tiempo en ranuras de 312 ηs con periodos de guarda entre saltos de 9.5 ηs. En cuanto a la modulación, MB-OFDM implementa QPSK[33] (2 bits/señal). MB-OFDM ofrece tasas de transferencias en el rango de 55 a 480 Mbps. La Figura 1-6 presenta el impulso de la estrategia MB-OFDM.

[33] Quadrature Phase Shift Keying. Tipo de modulación basada en PSK. BPSK implementa un esquema de 4 fases el cual permite codificar 2 bits por cada símbolo. Los cambios de fase se encuentran en 45, 135, 225 y 315°.

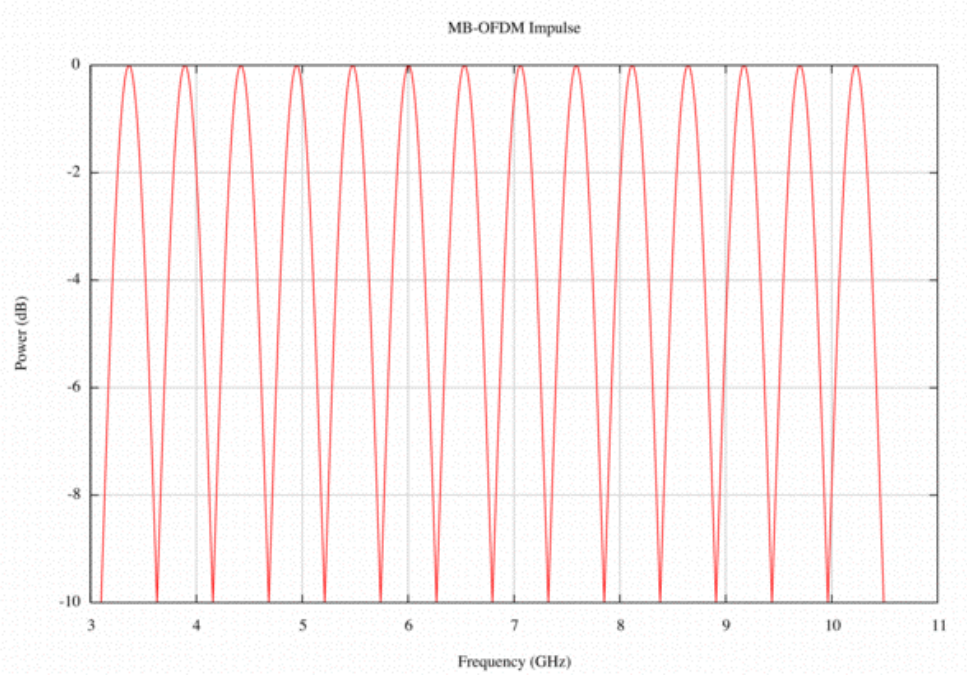


Fig. 1-6 Impulso MB-OFDM

Fuente: [5]

1.2.2.4 IEEE 802.15.4

El estándar IEEE 802.15.4 define las especificaciones para la capa PHY y MAC para comunicaciones inalámbricas de baja velocidad y orientada a dispositivos con limitado o muy bajo consumo de energía. El objetivo del estándar es proveer un mecanismo para desarrollar redes inalámbricas de muy bajo consumo de energía, donde los dispositivos sean capaces de funcionar durante largos periodos de tiempo, ya sean meses o años, sin que sus baterías sean recargadas o reemplazadas.

IEEE 802.15.4 define 27 canales distribuidos en diferentes bandas de frecuencia. Un canal se halla en la banda de 868 MHz, 10 en la banda de 902 - 928 MHz, con una separación entre canales de 2 MHz, y, finalmente, 16 canales de 3 MHz con una separación de 5 MHz en la banda de 2.4-2.4835 GHz. En lo que respecta a la cobertura, el estándar define un rango de 1 a 100 m considerando la potencia de transmisión.

El estándar usa el método DSSS[34] para transmitir señales de radio. A diferencia de Bluetooth, en el cual los dispositivos transmiten dando saltos entre canales de frecuencia, los dispositivos asociados a una PAN y basados en IEEE 802.15.4 usan un único canal de transmisión. Con respecto a la modulación, la banda de 2.4-2.4835 GHz usa QPSK alcanzado una tasa de transferencia de 250 Kbps, mientras que las bandas de 868 MHz y 902-928 MHz usan BPSK alcanzado tasas de 20 y 40 Kbps respectivamente.

El estándar IEEE 802.15.4 divide a los dispositivos en 2 grandes categorías, dispositivos que pueden comunicarse con cualquier otro dispositivo dentro de la PAN denominados FDD[35] y dispositivos que únicamente pueden comunicarse con FDDs denominados RFDs[36]. Esta categorización determina la topología de la red y el medio de acceso a la misma; es así que se tienen tres topologías, la tipo estrella, la topología punto a punto o malla y la topología árbol (cluster tree). La Figura 1-7 presenta las topologías definidas para el protocolo IEEE 802.15.4.

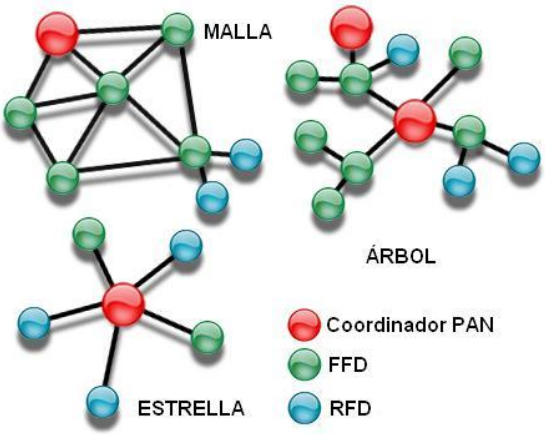


Fig. 1-7 Topologías definidas para el protocolo IEEE 802.15.4.

Fuente: [8]

[34] Direct Sequence Spread Spectrum: Espectro Expandido por Secuencia Directa. Es un método de codificación que usa una secuencia denominada código de dispersión, la cual es combinada con la señal de datos. La señal resultante debido a su espectro expandido y a su bajo nivel de potencia es muy parecido al del ruido, lo cual permite que sea descartada por todos los receptores a excepción del destinatario.
[35] Full Function Device: Dispositivo con funcionalidad completa.
[36] Reduced Function Device: Dispositivo con funcionalidad limitada.

De manera opcional, un FFD puede actuar como nodo coordinador regulando el acceso al medio, para lo cual se hace uso de una súper estructura conocida como *superframe*. El nodo coordinador se encarga de enviar señales, comúnmente conocidas como *beacons*, que identifican la PAN. La unión de dos *beacons* forma una *superframe*, el cual está dividido en 16 ranuras de tiempo agrupadas en ranuras pertenecientes al periodo libre de contención y las pertenecientes al periodo de contienda o contención. Las ranuras del periodo libre de contención pueden ser asignadas a los miembros de una PAN usando lo que se conoce como ranuras de tiempo garantizadas (GTSs[37]), mientras que las otras son accesibles a través del esquema CSMA-CA[38]. La Figura 1-8 presenta la estructura de un *superframe*.

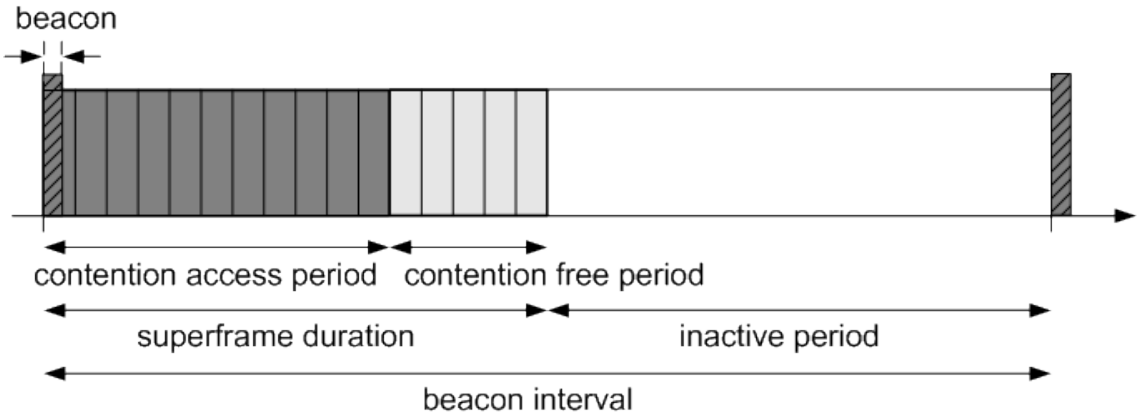


Fig. 1-8 Estructura de una superframe.

Fuente: [9]

[37] Guaranteed Time Slots. Ranuras de tiempo garantizadas. Son ranuras que aparecen en una superframe, localizadas luego de las ranuras pertenecientes al periodo de contienda o contención.

[38] Carrier Sense Multiple Access with Collision Avoidance: Acceso Múltiple por Detección de Portadora con Evasión de Colisiones. Es un proceso mediante el cual se verifica si el canal de transmisión está disponible, y así evitar colisiones producidas por transmisiones simultáneas.

1.3 Redes Inalámbricas de Sensores (WSN)

1.3.1 Introducción

El continuo y creciente avance de la tecnología, especialmente en lo que tiene que ver con miniaturización de hardware y comunicaciones inalámbricas han permitido el desarrollo de sistemas embebidos inalámbricos de pequeño tamaño pero con características apreciadas por la mayoría de sistemas. El bajo consumo de energía, lo cual se traduce en mayor autonomía, es una de ellas. Otra característica es su bajo precio tomando en cuenta que en la mayoría de estos se hallan embebidos una cantidad considerable de sensores o interfaces de comunicación. La integración de estos sistemas embebidos en una misma red es lo que se conoce como Red Inalámbrica de Sensores (WSN).

Muchas empresas han comprendido el potencial y los beneficios que ofrece toda la tecnología envuelta en las Redes Inalámbricas de Sensores, y han apostado por su desarrollo. En la actualidad, el mercado orientado a esta tecnología es tan diverso y amplio que es posible encontrar un gran número de plataformas inalámbricas, sistemas operativos, lenguajes de programación, simuladores, ambientes de desarrollo e inclusive soluciones completas listas para ser usadas.

Con respecto al uso de esta tecnología, o a sus aplicaciones, cabe mencionar que muchas áreas de la ciencia se han valido de esta tecnología para cumplir con ciertas tareas. El uso que se le ha dado a esta tecnología es tan variado, que es posible encontrar aplicaciones orientadas a la agricultura, a la domótica, al monitoreo ambiental, al monitoreo de estructuras civiles, al monitoreo de tráfico vehicular y al control industrial; es tal la apertura que se le ha dado a esta tecnología que existen incluso aplicaciones orientadas a la exploración espacial como la propuesta en [10].

1.3.2 Plataformas

El mercado orientado a redes inalámbricas de sensores ofrece una gama extensa de plataformas de desarrollo. Los diferencia una de otras su precio, su portabilidad, su soporte para diferentes tipos de protocolos, la cantidad de sensores embebidos, sus características de hardware como frecuencia de operación, velocidad del microprocesador y capacidad de memoria. Por mencionar algunas plataformas, están *telosb*, *mica2*, *micaz*, *iris*, *imote2* [11], *waspmote* [12]. La red inalámbrica propuesta en el presente trabajo está basada en la plataforma *imote2*, la cual será cubierta a detalle en la sección 2.2.1.

1.3.2.1 TelosB

TelosB es una plataforma de código abierto diseñada principalmente para fines experimentales y de investigación. Integra un microprocesador TI MSP430 a 8 MHz y un radio IEEE 802.15.4 que opera en la banda ISM de 2.4 GHz. Posee una gama de sensores de alta precisión de temperatura, humedad y luminosidad. TelosB es alimentado por dos pilas AA y proporciona un puerto USB para realizar tareas de programación y comunicación. Las Figuras 1-9 y 1-10 presentan la plataforma TelosB y su diagrama de bloques.

Fig. 1-9 Plataforma TelosB

Fuente: [13]

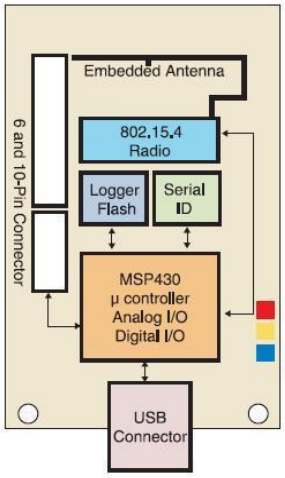


Fig. 1-10 Diagrama de bloques de la plataforma TelosB

Fuente: [13]

1.3.2.2 IRIS

IRIS es una plataforma basada en el microprocesador ATmega1281 que opera desde 8 a 16 MHz. Ofrece mejoras sustanciales con respecto a su plataforma predecesora *mica*; tales mejoras van desde integrar un radio IEEE 802.15.4 potenciado, el cual mejora su rango de cobertura en hasta 3 veces el rango de cobertura estándar y un incremento de memoria de programa al doble. IRIS posee un conector de expansión de 51 pines que provee interfaces como A/D, SPI, I2C y UART. Las Figuras 1-11 y 1-12 presentan la plataforma IRIS y su diagrama de bloques.

Fig. 1-11 Plataforma IRIS

Fuente: [14]

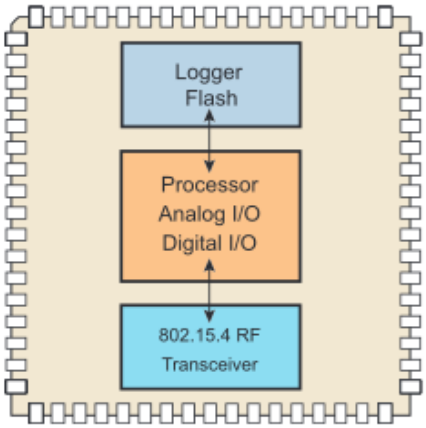


Fig. 1-12 Diagrama de bloques de la plataforma IRIS

Fuente: [14]

1.3.2.3 Waspmote

Waspmote está basado en un microprocesador ATmega1281 que opera desde 8 a 16 MHz. Soporta diferentes estándares de comunicación y opera a frecuencias de 868, 900 MHz y 2.4 GHz. Es una plataforma modular que permite la conexión de tarjetas de expansión o módulos. Waspmote dispone de una considerable variedad de tarjetas de expansión o módulos. Por mencionar algunos están: tarjetas de sensores (aproximadamente 50 tipos), módulos de comunicación (basados en 802.15.4, WiFi, Bluetooth, GSM/GPRS, 3G/GPRS, RFID) y módulos GPS. La Figura 1-13 presenta la plataforma operando en conjunto con algunos módulos. Las Figuras 1-14 y 1-15 presentan una vista frontal y posterior de la plataforma.

Fig. 1-13 Plataforma Waspmote

Fuente: [15]

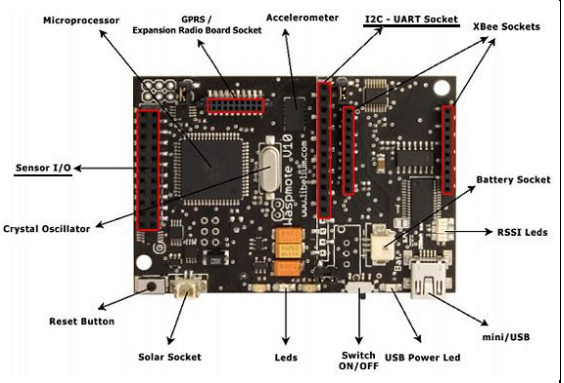


Fig. 1-14 Vista frontal de la plataforma

Fuente: [15]

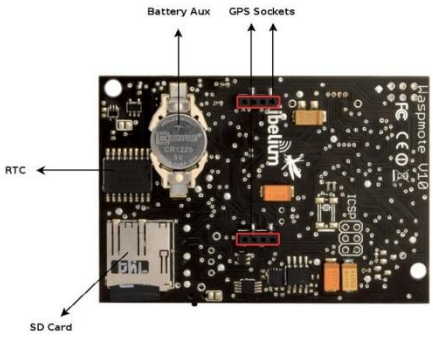


Fig. 1-15 Vista posterior de la plataforma

Fuente: [15]

1.3.3 Sistemas Operativos

De la misma manera que el mercado de redes sensores ofrece una extensa gama de plataformas, tal mercado ofrece también una cantidad considerable de sistemas operativos a ejecutarse en tales plataformas. Es posible encontrar desde simples y básicos hasta complejos y robustos. Entre

los más importantes y relevantes se podría destacar FreeRTOS [16], eCOS [17], ArdOS [18], Contiki [19] y TinyOS [20]. El sistema operativo que ha tenido mayor penetración y es considerado el más popular dentro de las redes de sensores es sin duda TinyOS, lo cual se debe al soporte para gran cantidad de plataformas y a su continuo desarrollo. A continuación se hace una revisión de dos de estos sistemas operativos.

1.3.3.1 TinyOS

TinyOS es un sistema operativo de código abierto, diseñado para sistemas inalámbricos de bajo consumo de energía y recursos limitados de hardware; está construido a partir de un conjunto de componentes de software reusables. Su reducido tamaño, el cual es poco menos de 400 bytes, implica que pueda ser cargado en dispositivos con recursos muy limitados de memoria. Su diseño está motivado por los siguientes aspectos: recursos limitados de hardware, concurrencia reactiva[39], flexibilidad y bajo consumo de energía [21].

TinyOS ha sido escrito en una variación del lenguaje de programación C, denominado NesC. TinyOS soporta un modelo de concurrencia basado en interfaces de fase dividida[40], eventos asíncronos y procedimientos aplazados[41] denominados tareas; además, implementa un modelo de programación basado en componentes. Este modelo se centra en los componentes como elementos que agrupan servicios los cuales se hallan definidos en interfaces.
Un aspecto muy importante en el diseño de TinyOS es su arquitectura de red, la cual está basada en pequeños paquetes denominados *Active Messages* (AM). AM provee un protocolo de datagramas no confiable y de un único salto, y una interfaz de comunicación que contempla

[39] Es una funcionalidad que permite administrar la confluencia de diferentes eventos respetando limitaciones de tiempo y recursos. La necesidad de administrar la concurrencia, se debe a que muchos de estos eventos requieren de procesamiento en tiempo real.
[40] Es una característica en la que la funcionalidad que provee una interfaz es procesada por separado o dividida en fases. En TinyOS, por ejemplo, los comandos solicitan la ejecución de un servicio y los eventos lanzan un evento luego de que la tarea asociada a ese servicio haya sido completada. Los comandos y eventos no pueden bloquearse entre sí.
[41] Son funciones de tipo síncrona que no requieren procesamiento inmediato y que generalmente son ejecutadas luego de que el procesador haya liberado recursos.

comunicaciones por radio y puerto serial. Existen también protocolos que proveen saltos múltiples e incluso variaciones de AM que proporcionan seguridad a nivel de enlace.

TinyOS ofrece soporte para una variedad de plataformas de hardware, lo cual incluye también soporte para circuitos integrados para procesamiento y para transmisión de radio frecuencia. Tal soporte permite que la misma aplicación compilada para una cierta plataforma pueda ser usada sin ninguna modificación en el resto de plataformas. El soporte de TinyOS incluye, además, un ambiente de simulación llamado TOSSIM, soporte para *JTAG*[42] para depuración de código, y de herramientas de visualización, e integración con otros lenguajes de programación.

1.3.3.2 Contiki

Contiki es un sistema operativo de código abierto orientado a sistemas con recursos limitados, enfocándose principalmente en dispositivos de bajo consumo de energía agrupados bajo el concepto de Internet de las cosas (IoT[43]). Provee la pila de protocolos TCP/IP y la pila RIME para comunicaciones inalámbricas de corto alcance, además de ofrecer soporte para IPv6. Posee un núcleo multitarea y ofrece la característica de multitarea preventiva por aplicación.

El modelo de programación de Contiki está basado en un mecanismo de programación concurrente denominado *protothread* o protohilo. Contiki integra en su modelo características de la ejecución por hilos de procesamiento y la programación orientada a eventos para evitar sobrecargar los protohilos. Los protohilos de un proceso son invocados por el núcleo o *kernel* del sistema operativo en respuesta a eventos internos o externos. Tales protohilos están programados de tal manera que el proceso asociado a estos retorna el control al núcleo cada cierto tiempo.

Contiki incluye un simulador de red denominado *Cooja* el cual permite simular diferentes tipos de plataformas de sensores, esto se lo hace a través de librerías previamente cargadas y compiladas. Cooja incluso puede emular plataformas a nivel de hardware, lo cual permite

[42] Join Test Action Group. Es el nombre comúnmente usado para la norma IEEE 1149.1.
[43] Internet of Things. Grupo de objetos interconectados de uso habitual o cotidiano.

observar y analizar el comportamiento de estas de manera real y precisa. El punto débil de este simulador está en la dependencia hacia herramientas externas como compiladores y enlazadores o *linkers* [22].

1.3.4 Tecnologías

1.3.4.1 ZigBee

ZigBee es una tecnología para WPAN basada en el estándar IEEE 802.15.4 y creada por un conglomerado de empresas de tecnología que conforman la ZigBee Alliance. Esta tecnología provee servicios de encripción, asociación, autenticación, encaminamiento o *routing* y servicios de capa de aplicación. ZigBee implementa las capas de nivel superior que sumadas a las capas ya definidas por el estándar IEEE 802.15.4 (MAC y PHY) forman la pila de protocolos de la tecnología ZigBee.

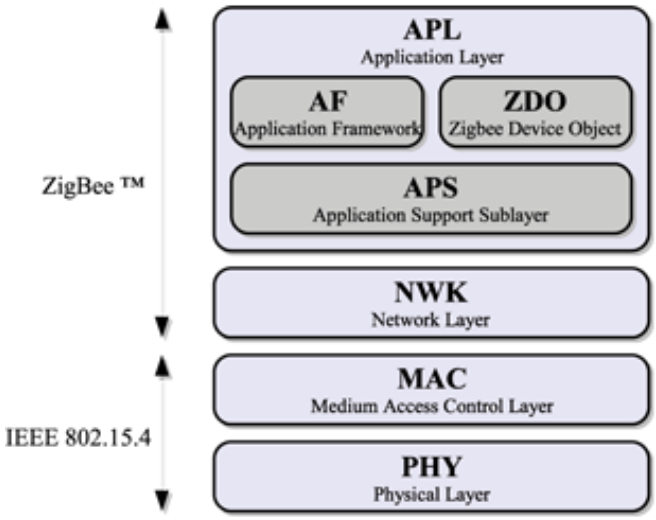


Fig. 1-16 Pila de protocolos de la tecnología ZigBee

Fuente: [23]

La tecnología ZigBee nativamente soporta topologías tipo estrella y árbol y una topología genérica tipo malla. Los nodos pueden cumplir el rol de coordinadores, encaminadores y dispositivos finales. Las funciones del nodo coordinador son controlar la red y proveer mecanismos de seguridad; los nodos encaminadores o routers extienden las redes sirviendo de puente entre los nodos finales y el coordinador; y, por, último los nodos finales son los encargados de tomar la información del medio. La Figura 1-17 presenta el ejemplo de una red ZigBee y los diferentes roles que desempeñan los nodos en la misma.

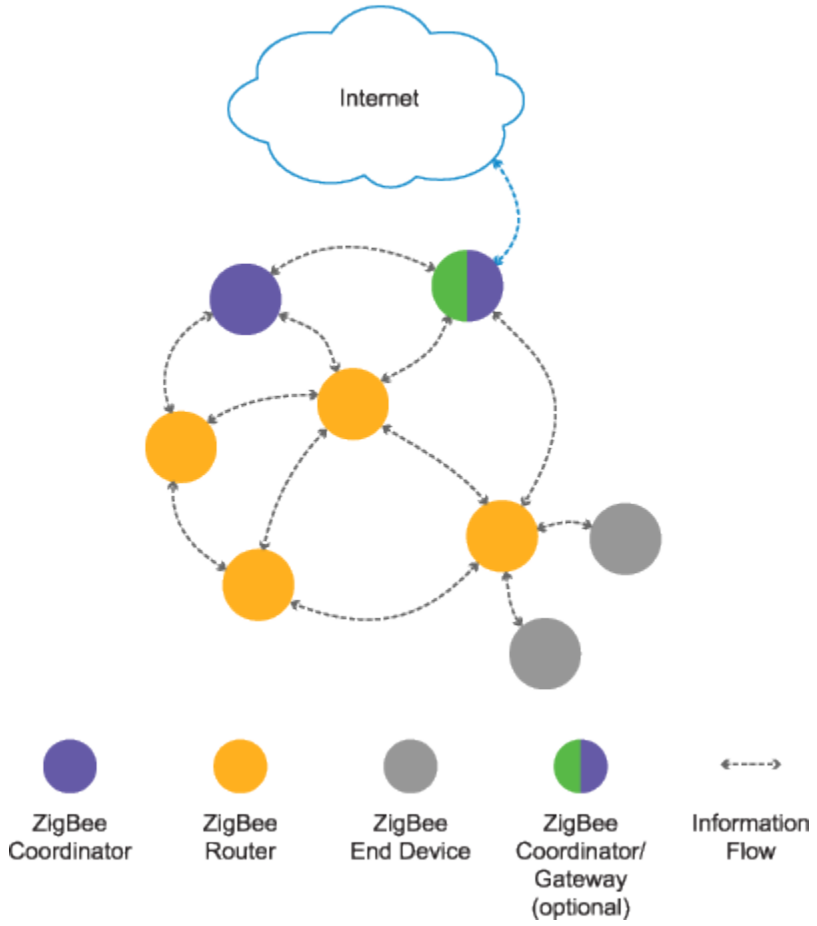


Fig. 1-17 Ejemplo de una red ZigBee

Fuente: [24]

1.3.4.1.1 Capa de Red (NWK)

Uno de los objetivos de la capa de Red es el ruteo de paquetes, es así que ZigBee define un protocolo de ruteo por vector distancia bajo demanda conocido como AODV[44]. En AODV, los nodos no almacenan información de rutas ni se hallan inmersos en un constante intercambio de tablas de ruteo. El momento en que los nodos requieren comunicarse entre sí, se inicia el proceso de descubrimiento de la red conocido como *Path Discovery*; el algoritmo, por lo tanto envía paquetes denominados paquetes de solicitud de ruta o *RREQ*[45], únicamente cuando es necesario, es decir, cuando se requiere informar de cambios en la red, y para diferenciar entre procesos de gestión y mantenimiento.

Tales paquetes incluyen información como direcciones de origen y destino, números de secuencia origen y destino, identificador de difusión, y un campo para el número de saltos. Los RREQs son retransmitidos o desechados dependiendo del identificador de difusión y de la dirección de origen. Los RREQs transmitidos añaden el peso del enlace después de cada salto, el cual difiere del número de saltos. Finalmente, la ruta con el valor de peso más bajo es la ruta escogida. En el recorrido desde el nodo origen hacia los destinos, la trayectoria de ida o *forward path* y la trayectoria de regreso o *reverse path* son definidas [25].

1.3.4.1.2 Capa de Aplicación (APL)

La especificación define un acuerdo, basado en políticas de red y seguridad, denominado perfil, el cual provee interoperabilidad para diferentes fabricantes de dispositivos. Existen tres tipos de perfiles: públicos, privados y liberados. Los perfiles públicos son administrados por la ZigBee Alliance, los privados son de uso restringido de los fabricantes de dispositivos y los liberados son perfiles de propiedad de fabricantes y que pueden ser publicados. Por mencionar algunos de los perfiles publicados o liberados, están ZigBee Home Automation, ZigBee Smart Energy, ZigBee Telecommunication Services y ZigBee Health Care [26].

[44] Ad-hoc On Demand Distance Vector
[45] Route Request

La capa de Aplicación incluye la mayoría de componentes definidos por la especificación ZigBee. Uno de estos componentes es el objeto de datos ZigBee o ZDO, el cual se encarga de establecer el rol o modo de operación de los dispositivos dentro de la red, la búsqueda de nuevos dispositivos y la determinación de los servicios de aplicación provistos por los mismos nodos. La subcapa de soporte de aplicación o APS funciona como puente entre la capa de red y los componentes de la capa de aplicación. APS se encarga de mantener actualizadas las tablas de enlace, las cuales permiten emparejar dos dispositivos basados en sus servicios y necesidades; de reenviar mensajes entre dispositivos enlazados, y de filtrar paquetes de dispositivos finales no registrados o perfiles no definidos. APS tiene a cargo también, la tarea de descubrir dispositivos que están operando dentro del espacio personal de operación de un dispositivo.

1.3.5 Aplicaciones

El vertiginoso desarrollo que ha sufrido la tecnología para redes inalámbricas de sensores, ha influido en la cantidad de aplicaciones desarrolladas para muchas aéreas de la ciencia. En la actualidad, existen aplicaciones orientadas a la agricultura, a la prevención de desastres naturales, al monitoreo de construcciones civiles, al monitoreo ambiental, y a la domótica por mencionar algunas; es más, posiblemente son muy pocas las áreas de la ciencia que no puedan hacer uso de esta tecnología para su beneficio. Las razones de tal incremento en la cantidad de aplicaciones, se deben sin duda al desarrollo del que han sido objetos las plataformas inalámbricas en cuanto a asequibilidad, desempeño, optimización, disponibilidad e interoperabilidad. A continuación se detallan tres casos prácticos del uso de esta tecnología, uno de los cuales es ejecutó en convenio con el Instituto Geofísico de la Escuela Politécnica Nacional y las universidades de Harvard y Carolina del Norte.

1.3.5.1 Monitoreo Ambiental en Serbia

EkoBus es un sistema desarrollado en conjunto con la empresa de telecomunicaciones Ericsson. El objetivo principal de este sistema es monitorear parámetros ambientales por medio del sistema

de transporte público. El sistema ofrece además, el servicio de localización y estimación de tiempo de arribo de los buses a sus distintas paradas. El proyecto fue concebido como una red inalámbrica de sensores de 20000 nodos repartidos en diferentes ciudades europeas [27]. La Figura 1-18 presenta el esquema de red del proyecto EkoBus.

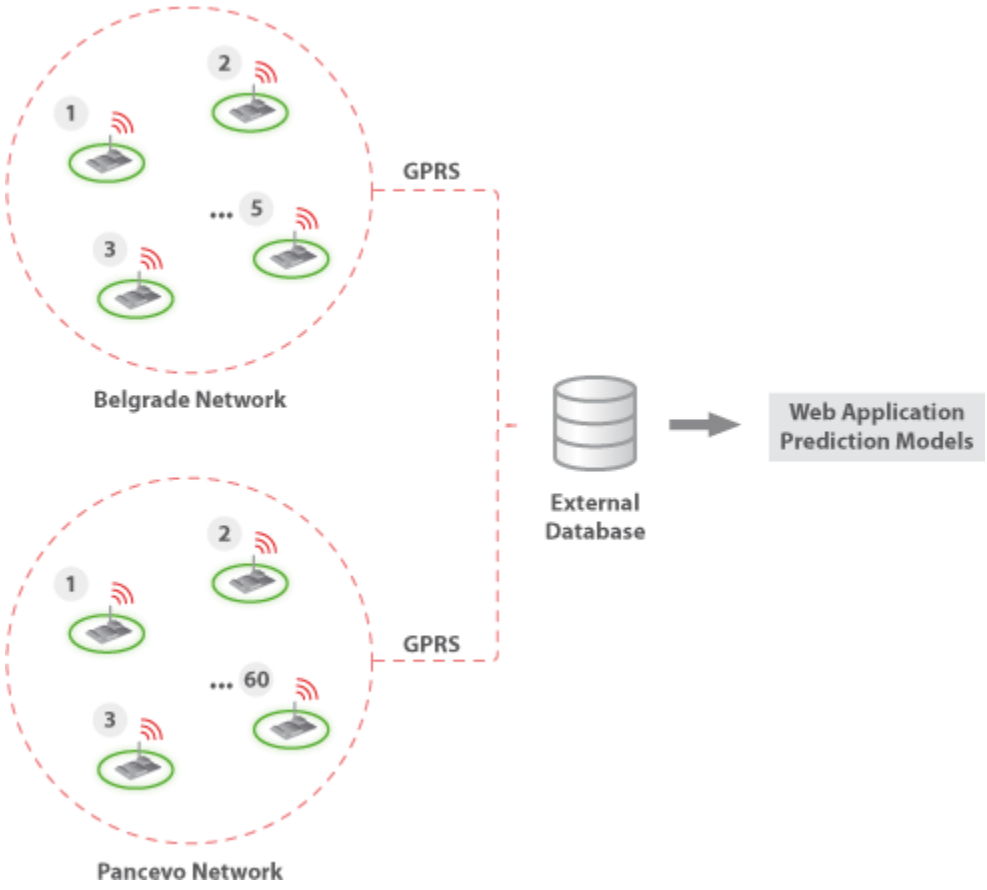


Fig. 1-18 Proyecto EkoBus

Fuente: [27]

La red inalámbrica de sensores está formada por 65 nodos repartidos en las ciudades serbias de Belgrado y Pancevo. En la ciudad de Belgrado fueron instalados 60 y 5 nodos en Pancevo. Los parámetros a medir son temperatura, humedad relativa, monóxido de carbono (CO), dióxido de carbono (CO_2) y dióxido de nitrógeno (NO_2). Todos los componentes como la tarjeta de sensores, el módulo inalámbrico, el módulo GPS y el módulo GPRS fueron encapsulados dentro

de una caja, la misma que fue instalada en el techo de los buses del sistema de transporte público [27].

Fig. 1-19 Dispositivos EkoBus instalados en el techo de los buses
Fuente: [27]

Los nodos hacen lecturas periódicas de los sensores anteriormente mencionados y envían esa información hacia un servidor por medio del módulo GPRS[46]. La información en el servidor es analizada y almacenada, para luego ser usada por aplicaciones para teléfonos inteligentes basados en el sistema operativo *Android*. Incluso es posible solicitar información del tiempo de arribo de los buses a través de mensajes de texto (SMS[47]) [27]. La Figura 1-20 presenta el diseño de la aplicación para teléfonos inteligentes.

[46] General Packet Radio Service. Servicio de transmisión de datos para teléfonos celulares, basado en conmutación de paquetes.
[47] Short Message Service. Servicio de mensajes cortos, disponible para teléfonos fijos y celulares.

Fig. 1-20 Interfaz gráfica de la aplicación para teléfonos inteligentes

Fuente: [27]

1.3.5.2 Detección de incendios en España

SISVIA Vigilancia y Seguimiento Ambiental es el nombre dado a un sistema de detección de incendios. El sistema cubre una extensión de 210 hectáreas, área comprendida entre las comunidades españolas de Galicia y Asturias. El objetivo principal de este sistema es proveer a las instituciones de control, una infraestructura que permita la generación de alertas tempranas. El sistema esta integrado por 3 componentes: una red inalámbrica de sensores, la red de transporte de datos y el Centro de Datos [28]. La Figura 1-21 presenta el diseño de red del sistema *SISVIA.*

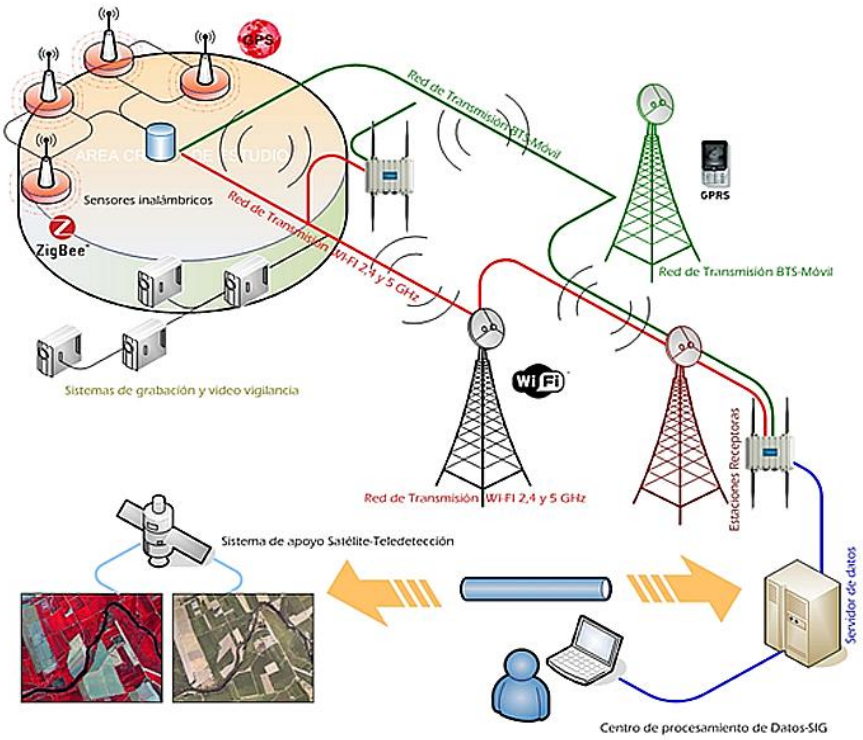


Fig. 1-21 SISVIA (Vigilancia y Seguimiento Ambiental)

Fuente: [28]

La red inalámbrica de sensores está formada por 90 nodos ubicados en sectores estratégicos. Los nodos integran sensores de dióxido de carbono (CO_2), monóxido de carbono (CO), temperatura y humedad relativa. Cada sensor muestrea estos parámetros en intervalos de 5 minutos. El sistema dependiendo de si los valores analizados superan umbrales establecidos, toma la decisión de enviar alertas tempranas a instituciones de control y rescate, como por ejemplo a bomberos. Por su parte, los bomberos sabrán con exactitud el lugar donde se está dando lugar un conato de incendio o un incendio, además, que podrán valerse de la información en tiempo real para conocer el comportamiento de los incendios [28]. La Figura 1-22 presenta la interfaz gráfica de *SISVIA*.

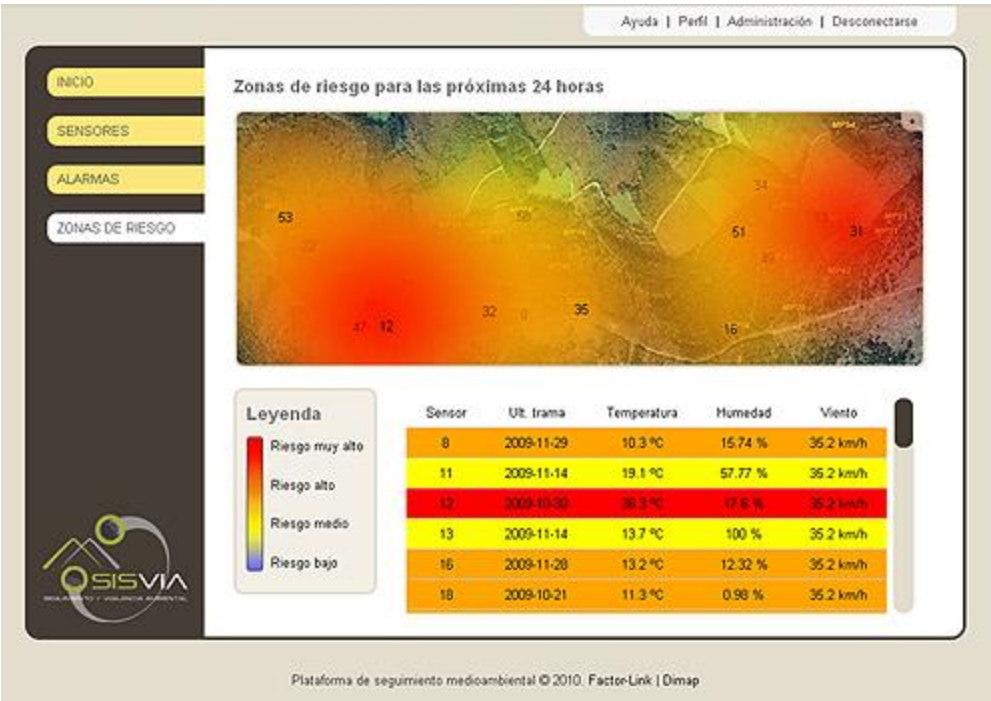


Fig. 1-22 Interfaz gráfica de SISVIA

Fuente: [28]

1.3.5.3 Desarrollo de una red inalámbrica de sensores en un volcán activo en Ecuador

En el año 2004, un grupo de expertos en computación de la universidad de Harvard establecen un convenio de colaboración con científicos y vulcanólogos de la universidad de Carolina del Norte y del Instituto Geofísico de la Escuela Politécnica Nacional para el desarrollo de una red inalámbrica de sensores para el estudio de volcanes activos. Es así que en el 2004 y como prueba de concepto, se instaló una pequeña red de sensores en el volcán Tungurahua. La red formada por 3 nodos, equipados con micrófonos, recogió por 3 días información del volcán. Año seguido fue instalado en el volcán Reventador una red mucho más grande, la cual constaba de 16 nodos equipados con sensores sismo acústico. La Figura 1-23 presenta la arquitectura propuesta para la red inalámbrica de sensores para monitoreo volcánico.

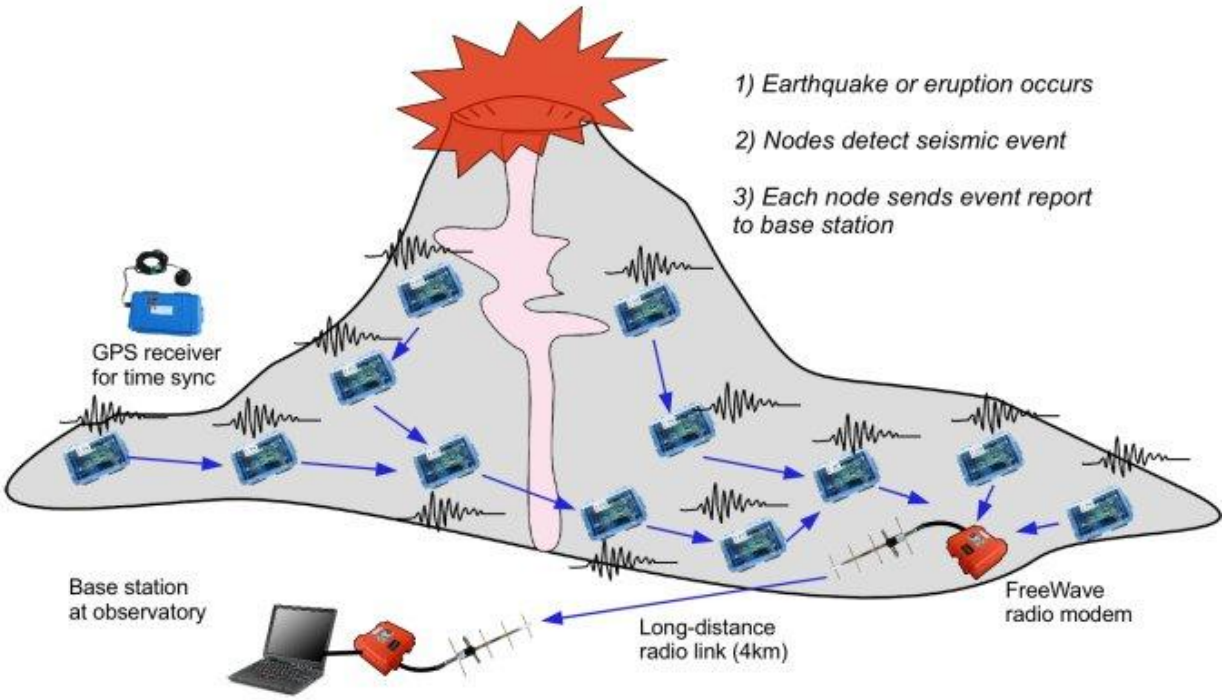


Fig. 1-23 Arquitectura de la red inalámbrica de sensores usada para el monitoreo volcánico

Fuente: [*29*]

La información generada fue ruteada a través de la red inalámbrica de sensores, que fue concebida como una red multi salto, y por un enlace de radio hacia el Centro de Datos ubicado a 4 km del lugar de pruebas. La limitada tasa de transmisión de los nodos influyo en el diseño de la red, la cual se enfoco únicamente en capturar eventos en lugar de señales continuas. La red propuesta estuvo operativa durante 3 semanas, tiempo en el cual se registraron alrededor de 230 eventos volcánicos, que significaron al mismo tiempo 107 megabytes de información almacenada.

CAPÍTULO 2

DISEÑO DEL SISTEMA

2.1 Topología de las redes

El sistema propuesto ha sido definido como el conjunto de dispositivos, infraestructura de comunicaciones y herramientas de software que permiten cumplir con el objetivo de monitorear y cuantificar flujos de lodo y escombros. Al sistema se lo ha bautizado con el acrónimo SMYCF, que viene de Sistema de Monitoreo Y Cuantificación de Flujos. Además, el SMYCF haría uso de una red de acceso (RA) y de la red microondas (RM) actual del IG para transportar la información generada por la RIS hacia el Centro de Datos del IG. Lo concerniente al diseño de la RA será cubierto al final de este capítulo en la sección 2.3. La Figura 2-1 presenta un diagrama de las distintas redes involucradas en el SMYCF.

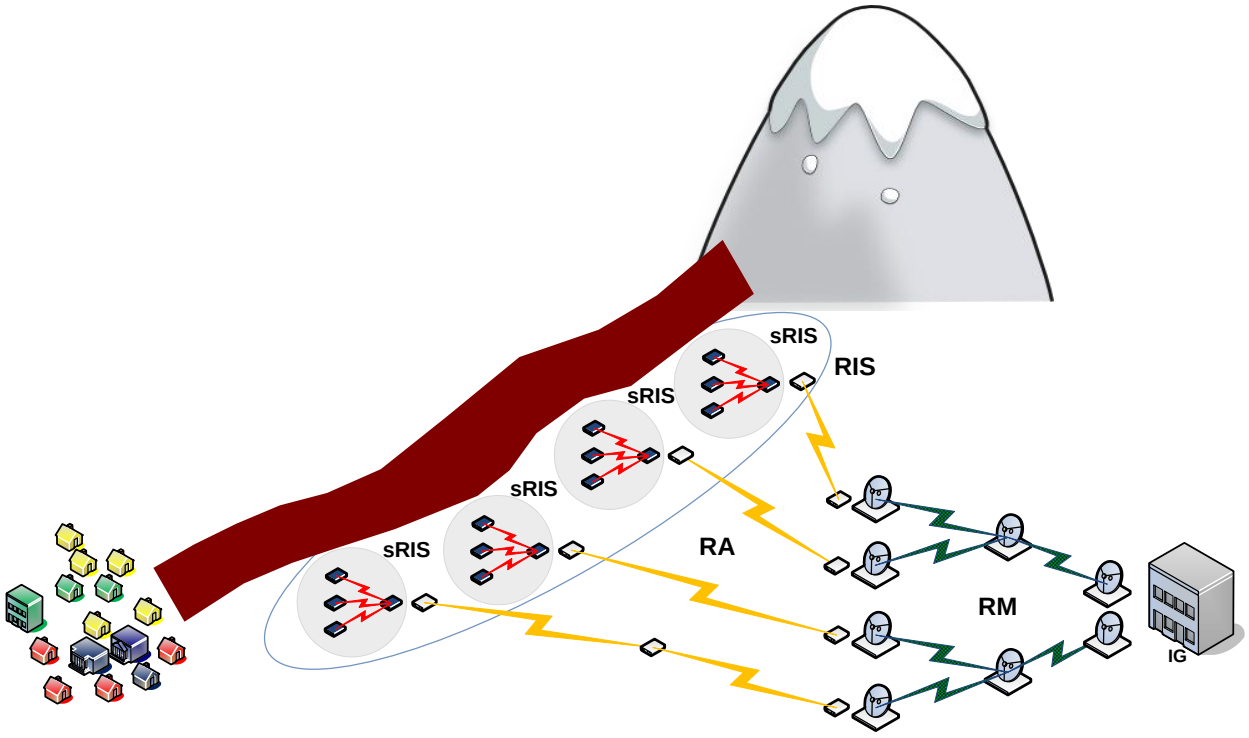


Fig. 2-1 Redes involucradas en el SMYCF

2.1.1 Topología de la sub red inalámbrica de sensores (sRIS)

La sRIS fue ideada como una pequeña red en la que un módulo base se encargue de tareas como: recibir, procesar y enviar información generada por la misma hacia la RA; sincronización de tiempo, y otras, mientras módulos finales se encarguen de tareas mínimas como por ejemplo hacer lecturas periódicas de sensores. Es así que la topología que mejor se adaptó al hecho de contar con una red inalámbrica de sensores, donde un módulo base actúe como coordinador para el resto de módulos, y a su vez como puerta de enlace entre la sRIS y la RA, fue una topología tipo estrella.

Cabe mencionar que existe la posibilidad de adaptar la sRIS a una topología tipo malla donde módulos finales actúen además como estaciones de repetición, dando lugar a la característica denominada multi salto; sin embargo dado que la sRIS propuesta ha sido concebida para ser instalada dentro de una misma área de cobertura, donde módulos finales y módulo base no requieran más de un salto para comunicarse entre si, la topología tipo estrella es la que mejor se adaptó. La Figura 2-2 presenta la topología adoptada por la sRIS.

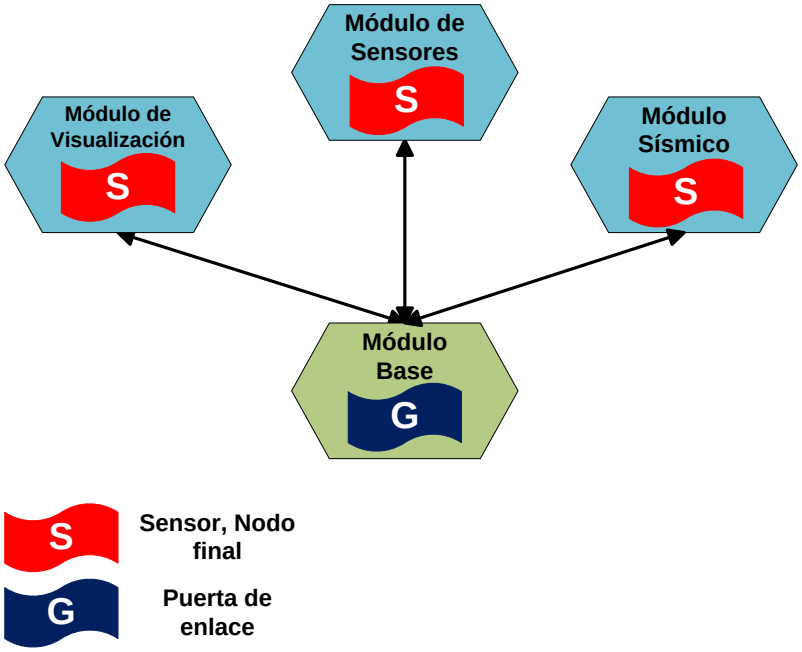


Fig. 2-2 Topología tipo estrella adoptada por la sRIS

2.1.2 Topología de la red microondas (RM)

El diseño de la RM está basado en una topología tipo árbol, sin embargo, cada una de las estaciones de repetición funciona como nodo central de una configuración tipo estrella. A estos nodos de repetición se conectan estaciones o dispositivos de distinto tipo como digitalizadores, sensores sísmicos, acelerómetros, estaciones GPS[48] y AFMs, cámaras de video, y otros, La RM usa equipamiento con tecnología PDH[49] provista por la empresa SIAE Microelettronica a través de su representante en el Ecuador (SIAEMICRO Andina S.A.), con su serie ALC orientada a redes de bajo a mediano volumen de tráfico de datos.

En lo que tiene que ver con los dispositivos de hardware, cada punto de la RM está equipado con una unidad para exteriores comúnmente denominada ODU[50] y una unidad para interiores denominada IDU[51]. La IDU provee las interfaces de comunicación y las unidades controladoras, mientras que la ODU realiza la conversión de señales RF[52] a IF[53] y viceversa. La Figura 2-3 presenta una imagen del equipamiento en mención y algunos datos técnicos de los mismos, la Figura 2-4 presenta el esquema de la RM del IG y la Figura 2-5 presenta la ubicación de las estaciones de repetición de la RM.

[48] Global Positioning System o Sistema de Posicionamiento Global. Sistema de navegación basado en satélites, el cual provee información de localización.
[49] Plesiochronous Digital Hierarchy. Tecnología de telecomunicaciones diseñada para transportar grandes volúmenes de datos usando técnicas de multiplexación.
[50] Outdoor Unit o Unidad Externa.
[51] Indoor Unit o Unidad Interna.
[52] Radio Frequency o Radio Frecuencia. Conjunto de frecuencias comprendida en el rango de 3 Hz a 300 GHz.
[53] Intermediate Frequency o Frecuencia Intermedia. Frecuencia que resulta de mezclar la frecuencia de portadora y una frecuencia variable generada por el dispositivo.

	ODU Versión 1+1 hot stand-by con antena integrada, que consiste de 2 ODUs 1+0 aseguradas mecánicamente a una misma estructura. Frecuencia de operación de 7/8 GHz. Temperatura de operación entre -33 y 55 °C. IDU Provee 16 canales de tributario E1 (2 Mbps). Provee 3 interfaces Ethernet 10/100 BaseT. Capacidad total desde 4 a 64 Mbps. Tipo de modulación 4QAM/8QAM/16QAM. Temperatura de operación entre -5 y 45 °C.

Fig. 2-3 Imagen del equipamiento y especificaciones técnicas de los dispositivos de la serie ALC

Fuente: [29]

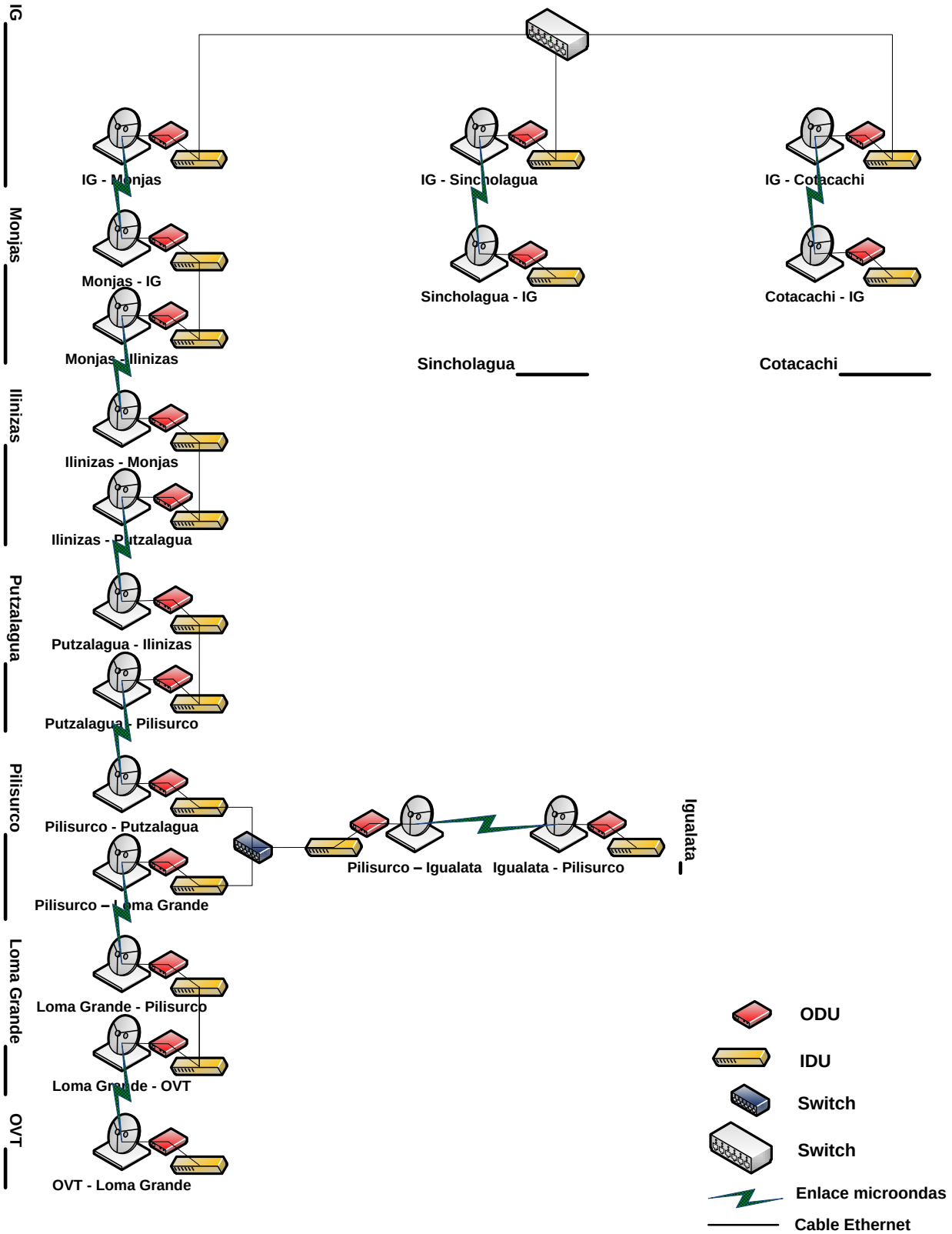


Fig. 2-4 Esquema de la RM del IG

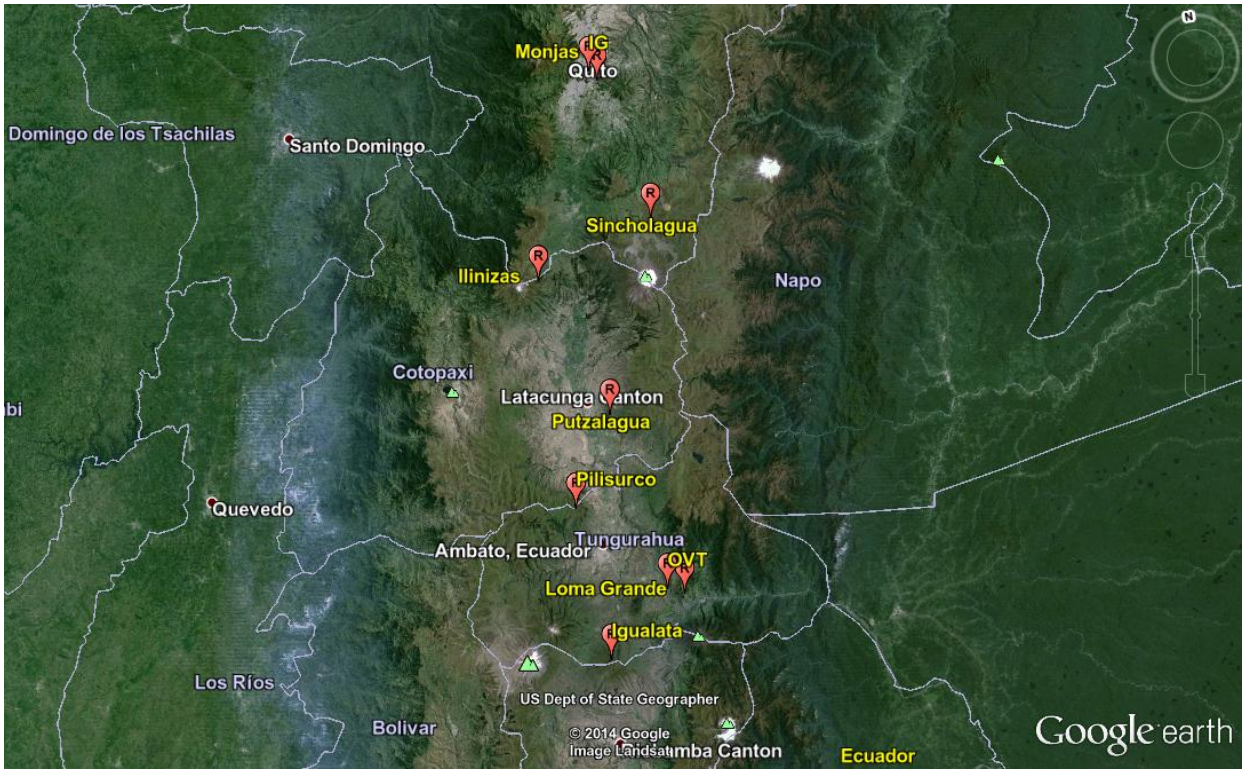


Fig. 2-5 Ubicación de las estaciones de repetición de la RM

2.2 Diseño y desarrollo de la sRIS

2.2.1 Plataforma Inalámbrica

En el año 2009, por medio de un proyecto auspiciado por el Banco Interamericano de Desarrollo (BID), el IG hizo la adquisición de un gran número de kits de desarrollo y algunas plataformas para redes inalámbricas de sensores. Tal disponibilidad más algunos aspectos técnicos que serán mencionados más adelante, llevaron a la decisión de basar el desarrollo de la sRIS en imote2. La plataforma fue desarrollada inicialmente por Crossbow Technology Inc. para luego ser parte de la línea de productos adquirida por Memsic Inc. en el 2010. La plataforma fue escogida debido a sus características de hardware que superan en gran medida las de otras plataformas como tmote sky, mica2, micaz, waspmote y otras, así como por su disponibilidad para trabajar con distintos sistemas operativos como por ejemplo: TinyOS, .NET Micro Framework, Linux y SOS; y claro,

está también el hecho de que la plataforma ha sido potenciada debido al continuo desarrollo que se le ha dado a la misma, en especial al sistema operativo TinyOS a través de contribuciones de código.

Cabe mencionar, que Memsic provee tarjetas de expansión para la plataforma imote2, las cuales añaden valor y funcionalidad a ésta; es posible, sin embargo, la integración de tarjetas desarrolladas por terceros con funcionalidades muy específicas. Para este caso en específico, se diseñó e implementó una tarjeta de interfaces, la cual será revisada posteriormente en este capítulo en la sección 2.2.2.1. A continuación se describe brevemente cada una de las tarjetas de expansión de las que dispone la plataforma y que son usadas en el presente trabajo.

2.2.1.1 IPR2400

La tarjeta de procesamiento o formalmente conocida como IPR2400 integra un procesador escalable Intel PXA271. La escalabilidad en este caso está dada por el hecho de que el procesador puede operar a diferentes frecuencias, sacrificando el consumo de energía para mejorar el rendimiento del procesador y viceversa. En modo de bajo consumo alcanza una frecuencia de operación de 13 MHz, sin embargo esta puede alcanzar los 416 MHz con un consumo alto de energía por medio de una característica denominada “Escalamiento Dinámico de Voltaje”. Su diseño modular y apilable, permite a tarjetas de expansión como tarjetas de sensores o de alimentación, sean conectadas a la plataforma. La Figura 2-6 presenta la tarjeta de procesamiento IPR2400 y la Figura 2-7 el diagrama de bloques de la misma.

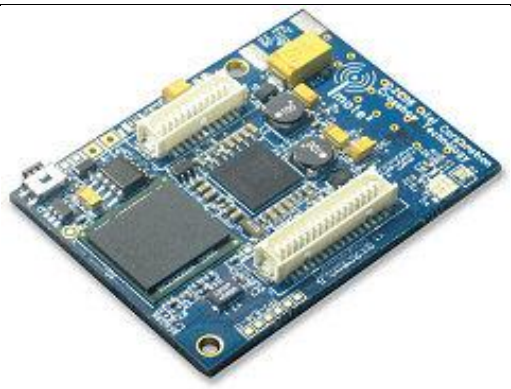


Fig. 2-6 Tarjeta de Procesamiento IPR2400

Fuente: [30]

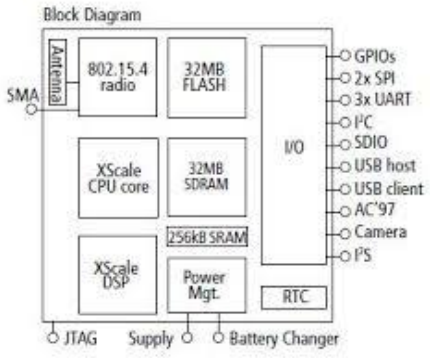


Fig. 2-7 Diagrama de bloques de la IPR2400

Fuente: [30]

2.2.1.2 ITS400

La tarjeta de sensores, formalmente conocida como ITS400, integra una cantidad considerable de sensores tales como un acelerómetro triaxial de 12 bits, un sensor de alta precisión de temperatura de 14 bits y uno de humedad de 12 bits, un sensor de luz de dos canales de 16 bits para medir luz visible e infrarroja y un conversor analógico digital de 4 canales de 12 bits. Al igual que la IPR2400, la ITS400 posee un diseño modular y apilable. La Figura 2-8 muestra la tarjeta de sensores ITS400 y la Figura 2-9 el diagrama de bloques de la misma.

Fig. 2-8 Tarjeta de sensores ITS400

Fuente: [30]

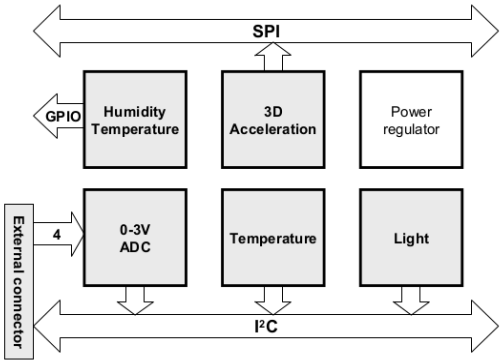


Fig. 2-9 Diagrama de bloques de la ITS400

Fuente: [30]

2.2.1.3 IBB2400

La IBB2400 es una tarjeta de alimentación que permite energizar todas las tarjetas apiladas a través de sus dos pares de conectores. Su diseño permite conectar 3 baterías AAA NiMH[54] recargables o no recargables. Posee un interruptor manual en un extremo para interrumpir el paso de corriente y un circuito de protección contra sobrecargas. La Figura 2-10 presenta la tarjeta de alimentación IBB2400 conectada a la tarjeta de sensores ISM400[55] usada para monitoreo estructural.

Fig. 2-10 Tarjeta de alimentación IBB2400 conectada a la tarjeta de sensores ISM400
Fuente: [31]

2.2.1.4 IIB2400

Esta tarjeta de interfaces, formalmente conocida como IBB2400, provee 2 puertos USB a Serial y una interface JTAG para depuración de código. A través de la interface JTAG y por medio de un cable JTAG es posible la reprogramación de la memoria flash de la IPR2400. Cabe mencionar

[54] Níquel Metal Hidruro.
[55] Tarjeta de sensores diseñada por Open Systems Laboratory & Smart Structures Technology Laboratory de la Universidad de Illinois [65].

que la IPR2400 trae cargado por defecto la plataforma .NET Micro Framework[56]. Adicionalmente, Crossbow Technology Inc. en su momento libero un kit de desarrollo compuesto por herramientas de software y librerías denominado Imote2 SDK, el cual fue discontinuado tras la adquisición de la plataforma por parte de Memsic Inc. Los requerimientos de procesamiento de los módulos que integran la sRIS, así como la discontinuidad del kit de desarrollo, determinó que las IPR2400 usadas en el presente trabajo sean reprogramadas con el sistema operativo TinyOS. El cable JTAG usado es el Olimex ARM-USB-TINY-H. La Figura 2-11 presenta la conexión requerida para reprogramar la memoria flash de la IPR2400. La Figura 2-12 presenta la tarjeta de interfaces IBB2400 y la Figura 2-13 el diagrama de bloques de la misma.

Fig. 2-11 Olimex ARM-USB-TINY-H JTAG, IPR2400 e IIB2400

Fuente: [32]

[56] Plataforma .NET de código abierto para dispositivos con recursos limitados.

Fig. 2-12 Tarjeta de interfaces IIB2400

Fuente: [30]

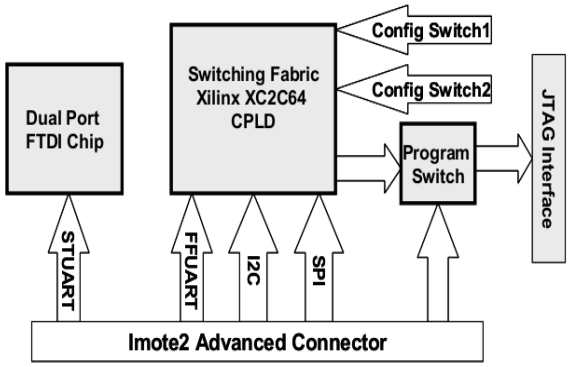


Fig. 2-13 Diagrama de bloques de la IIB2400

Fuente: [30]

2.2.2 Módulos Inalámbricos

2.2.2.1 Módulo Base

El Módulo Base está integrado por la tarjeta de procesamiento IPR2400 y la tarjeta de interfaces IEF100, diseñada por el Departamento de Instrumentación del IG, la cual lleva embebido un circuito integrado MAX3232[57] que permite transformar señales TTL[58] en señales RS232 y viceversa. La IEF100 fue diseñada y construida para suplir los requerimientos que demandaba el Módulo Base, como por ejemplo la disponibilidad de dos de los tres puertos UART[59] que trae la IPR2400. La Figura 2-14 presenta la tarjeta de interfaces IEF100.

[57] Circuito integrado que permite convertir señales TTL a RS232 y viceversa [79].
[58] Tecnología para diseño y construcción de circuitos integrados digitales [80].
[59] Universal Asynchronous Receiver Transmiter. Interfaz de comunicación serial [81].

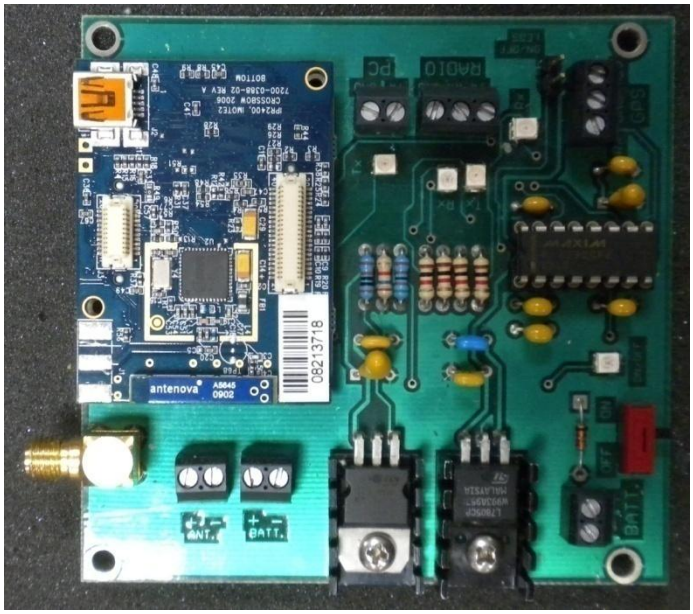


Fig. 2-14 Tarjeta de interfaces IEF100

IEF100 incluye un par de conectores avanzados tipo hembra los cuales permiten adherirse a los conectores tipo macho de la tarjeta IPR2400. Para este prototipo se usó 2 de los 3 canales disponibles en el MAX3232. Uno de estos es usado para convertir señales RS232 provenientes de un receptor GPS en señales TTL, mientras que el otro es usado para convertir señales TTL en señales RS232. La tarjeta IPR2400 hace uso de dos de sus tres UARTs para recibir y transmitir información desde y hacia los distintos dispositivos externos. La plataforma en cuestión establece, de acuerdo a su funcionalidad, prefijos para los diferentes UARTs, es así que se tiene: ST (Standard), FF (Full Function) y BT (Bluetooth). Para este caso los puertos de alta velocidad a ser usados son: STUART y FFUART. La línea de transmisión del puerto STUART es usada como puerto de comunicaciones; mientras que las líneas de recepción y transmisión del puerto FFUART como puertos de sincronización y depuración respectivamente. La Figura 2-14 presenta los puertos de alta velocidad que provee la tarjeta IPR2400 y sus funciones dentro del Módulo Base.

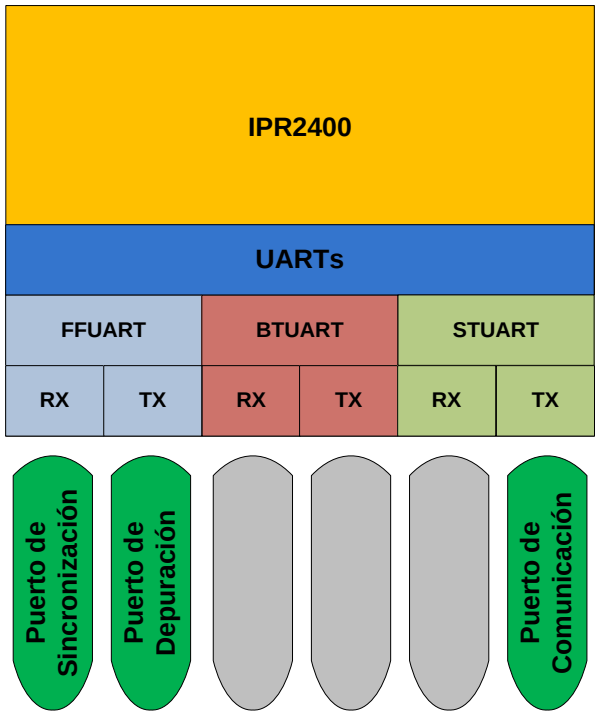


Fig. 2-15 Puertos UARTs de la tarjeta IPR2400 y sus usos dentro del Módulo Base

En lo que se refiere a la sincronización de tiempo del Módulo Base, se usó un receptor GPS 18x LVC del fabricante GARMIN. Este modelo maneja niveles de voltaje TTL e incluye la especificación NMEA 0183[60] para comunicación entre dispositivos marítimos. El Módulo Base hace uso de las sentencias GPGGA y GPRMC definidas en la especificación, para su sincronización. Los parámetros tiempo, longitud, latitud, y el número de satélites son obtenidos de la sentencia GPGGA; y el parámetro fecha de la sentencia GPRMC. Las Figuras 2-16 y 2-17 presentan las estructuras de las sentencias NMEA usadas por el Módulo Base. El Módulo Base ha sido programado de manera que el proceso de sincronización de tiempo sea automático. En otras palabras, cada cierto tiempo, el Módulo Base se engancha al GPS para sincronizar su tiempo, tras luego de lo cual, se desengancha del mismo.

Con respecto al mantenimiento y verificación del estado de la red, el Módulo Base implementa un puerto de depuración. La interfaz que integra el Módulo Base para este propósito es una bajo

[60] Acrónimo de National Marine Electronics Association. Estándar que define una interfaz eléctrica y un protocolo de datos para comunicación entre dispositivos marítimos.

la especificación RS232. Por otro lado, el puerto de comunicación es usado para transmitir información de la red inalámbrica hacia un radio modem. De la misma manera que el puerto de depuración, el de comunicaciones implementa un puerto serial con especificación RS232. La Tabla 2-1 presenta la configuración por defecto de los puertos que integran el Módulo Base.

```
 $GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Where:
     GGA          Global Positioning System Fix Data
     123519       Fix taken at 12:35:19 UTC
     4807.038,N   Latitude 48 deg 07.038' N
     01131.000,E  Longitude 11 deg 31.000' E
     1            Fix quality: 0 = invalid
                               1 = GPS fix (SPS)
                               2 = DGPS fix
                               3 = PPS fix
                               4 = Real Time Kinematic
                               5 = Float RTK
                               6 = estimated (dead reckoning) (2.3 feature)
                               7 = Manual input mode
                               8 = Simulation mode
     08           Number of satellites being tracked
     0.9          Horizontal dilution of position
     545.4,M      Altitude, Meters, above mean sea level
     46.9,M       Height of geoid (mean sea level) above WGS84
                      ellipsoid
     (empty field) time in seconds since last DGPS update
     (empty field) DGPS station ID number
     *47          the checksum data, always begins with *
```

Fig. 2-16 Estructura de la sentencia GPGGA

Fuente: [33]

```
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

Where:
     RMC          Recommended Minimum sentence C
     123519       Fix taken at 12:35:19 UTC
     A            Status A=active or V=Void.
     4807.038,N   Latitude 48 deg 07.038' N
     01131.000,E  Longitude 11 deg 31.000' E
     022.4        Speed over the ground in knots
     084.4        Track angle in degrees True
     230394       Date - 23rd of March 1994
     003.1,W      Magnetic Variation
     *6A          The checksum data, always begins with *
```

Fig. 2-17 Estructura de la sentencia GPRMC

Fuente: [33]

Puerto	Especificación	Configuración
Sincronización	RS232	4800 baudios, 8N1
Depuración	RS232	4800 baudios, 8N1
Comunicación	RS232	19200 baudios, 8N1

Tabla 2-1 Configuración de los puertos UARTs del Módulo Base

El Módulo Base se encarga de realizar cuatro tareas específicas. La primera de ellas consiste en mantenerse a la escucha de mensajes provenientes de los diferentes módulos que forman la red; cada uno de los módulos genera programáticamente un tipo de mensaje que incluye información de identificación, así como datos correspondientes a cada módulo. La segunda tarea consiste en sincronizar su reloj interno con la información proveniente del receptor GPS; la sincronización se la realiza cada cierto tiempo luego de lo cual el Módulo Base se desengancha del receptor con el fin de ahorrar energía y recursos de procesamiento. La tercera tarea consiste en generar programáticamente paquetes de datos en base a los mensajes enviados por los diferentes módulos, y enviarlos a través del puerto de comunicaciones; los paquetes de datos incluyen información de identificación, datos correspondientes a cada módulo, una estampa de tiempo e información adicional que es usada por el software de adquisición. Y finalmente, la última tarea consiste en gestionar las alertas generadas por los diferentes módulos; un tipo de mensaje que incluye un identificador de alerta es enviado desde el Módulo Base hacia el Módulo de Visualización.

A parte de estas tres tareas, el Módulo Base ejecuta la tarea adicional de enviar información relacionada al estado de la red y todas las tareas que se vayan ejecutando a través del puerto de depuración, lo cual ayuda de sobremanera a la tarea de mantenimiento y depuración de la red ya que permite identificar fácilmente errores de funcionamiento y corregirlos rápidamente. La Figura 2-18 presenta un diagrama que ejemplifica el comportamiento del Módulo Base y sus funcionalidades.

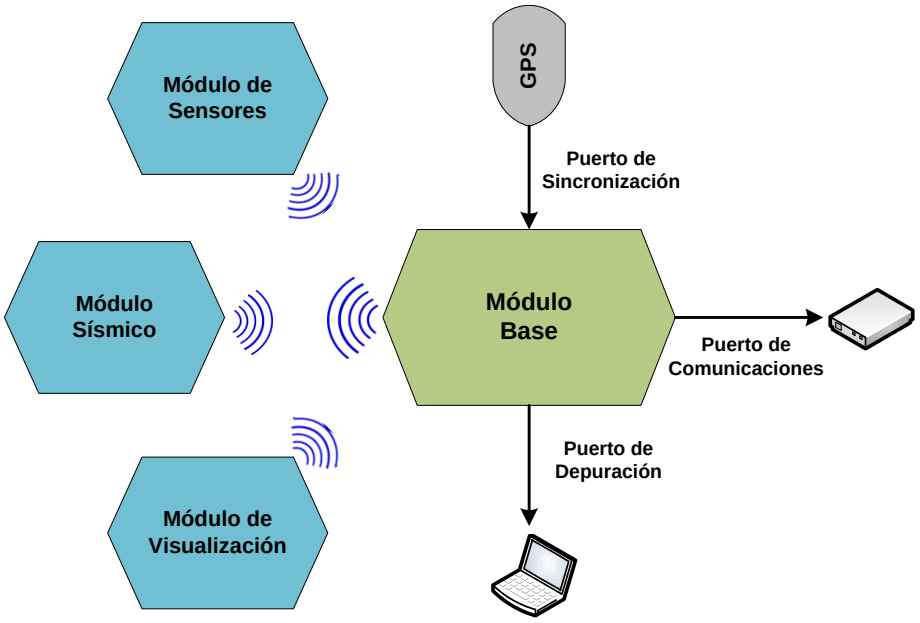


Fig. 2-18 Funcionalidades del Módulo Base

Para el envío de mensajes desde los diferentes módulos hacia el Módulo Base, así como para la transmisión de paquetes de datos desde el Módulo Base hacia la RA, se crearon estructuras de datos, estableciendo diferencias entre cada caso. Las estructuras manejadas dentro de la sRIS se denominan *mensajes*, mientras que las estructuras manejadas fuera de la sRIS se denominan *paquetes*. Para identificar las estructuras denominadas mensajes de los paquetes se usaron los sufijos *Msg* y *Pck*. La Figura 2-19 presenta un diagrama que incluye los mensajes y paquetes manejados por el Módulo Base y la Tabla 2-2 presenta el detalle de cada una de las estructuras creadas para manejar la información.

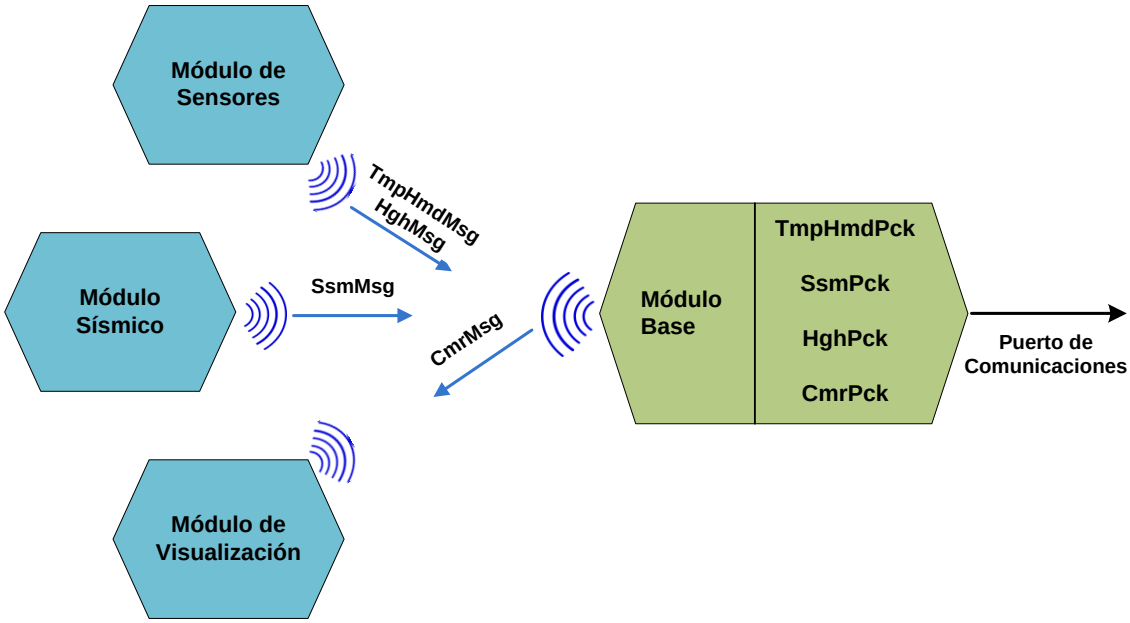


Fig. 2-19 Estructuras de datos usadas por los diferentes módulos que integran la sRIS

Módulo	Mensajes	Paquetes

Módulo de Sensores	//Estructura que incluye la lectura de los sensores de temperatura y humedad
	
	typedef nx_struct TmpHmdMsg {
	nx_uint8_t alrt; //identificador alerta
	nx_uint8_t snid; //identificador sRIS
	nx_uint16_t tmpr; //temperatura
	nx_uint16_t hmdt; //humedad
	} TmpHmdMsg;
	
	//Estructura que incluye la lectura del sensor de altura o radar
	
	typedef nx_struct HghMsg {
	nx_uint8_t alrt; //identificador alerta
	nx_uint8_t snid; //identificador sRIS
	nx_uint16_t height; //altura
	} HghMsg;
		//Estructura que incluye la lectura de los sensores de temperatura y humedad y la estampa de tiempo
		
		typedef nx_struct TmpHmdPck {
		nx_uint8_t alrt; //identificador alerta
		nx_uint8_t snid; //identificador sRIS
		nx_uint16_t tmpr; //temperatura
		nx_uint16_t hmdt; //humedad
		nx_uint16_t year; //año
		nx_uint8_t month; //mes
		nx_uint8_t day; //día
		nx_uint8_t hour; //hora
		nx_uint8_t mnte; //minuto
		nx_uint8_t scnd; //segundo
		nx_uint8_t stll; //número satélites
		} TmpHmdPck;

		//Estructura que incluye la lectura del sensor de altura y la estampa de tiempo
		
		typedef nx_struct HghPck {
		nx_uint8_t alrt; //identificador alerta
		nx_uint8_t snid; //identificador sRIS
		nx_uint16_t height; //altura
		nx_uint16_t year; //año
		nx_uint8_t month; //mes
		nx_uint8_t day; //día
		nx_uint8_t hour; //hora
		nx_uint8_t mnte; //minuto
		nx_uint8_t scnd; //segundo
		nx_uint8_t stll; //número satélites
		} HghPck;

Módulo Sísmico	//Estructura que incluye el valor de las tres componentes del sensor sísmico
	
	typedef nx_struct SsmMsg {
	nx_uint8_t alrt; //identificador alerta
	nx_uint8_t snid; //identificador sRIS
	nx_uint16_t full; //componente full
	nx_uint16_t fthg; //componente high
	nx_uint16_t ftlw; //componente low
	} SsmMsg;
		//Estructura que incluye el valor de las tres componentes del sensor sísmico y la estampa de tiempo
		
		typedef nx_struct SsmPck {
		nx_uint8_t alrt; //identificador alerta
		nx_uint8_t snid; //identificador sRIS
		nx_uint16_t full; //componente full
		nx_uint16_t fthg; //componente high
		nx_uint16_t ftlw; //componente low
		nx_uint16_t year; // año
		nx_uint8_t month; //mes
		nx_uint8_t day; //día
		nx_uint8_t hour; //hora
		nx_uint8_t mnte; //minuto
		nx_uint8_t scnd; //segundo
		nx_uint8_t stll; //número satélites
		} SsmPck;

Módulo de Visualización	//Estructura usada por el Módulo Base para notificar al Módulo de Visualización de cambios en el estado de alerta de la sRIS.	

	typedef nx_struct CmrMsg {
	nx_uint8_t alrt; //identificador alerta
	nx_uint8_t snid; //identificador sRIS
	} CmrMsg;	

Tabla 2-2 Detalle de las estructuras de datos usadas por los diferentes módulos que integran la sRIS

TinyOS provee una estructura de información para mensajes de radio denominada *message_t*, la cual es presentada en la Figura 2-20. Las estructuras de datos definidas como *mensajes* son encapsuladas dentro del campo *data* de la estructura message_t.

```
typedef nx_struct message_t {
  nx_uint8_t header[sizeof(message_header_t)];
  nx_uint8_t data[TOSH_DATA_LENGTH];
  nx_uint8_t footer[sizeof(message_footer_t)];
  nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```

Fig. 2-20 Estructura del message_t para comunicación por radio y provista por TinyOS 2.x, usada para el envío de mensajes

Fuente: [35]

TinyOS establece también un formato para comunicación serial, el cual es presentado en la Figura 2-21. De manera similar a los *mensajes*, las estructuras de datos definidos como *paquetes* son encapsulados en el campo payload.

```
 ____________________________________
| | | | |                          | | |
| | | | |                          | | |
|_|_|_|_|__________________________|_|_|
 F P S D         Payload            CR F

F       = Framing byte, denoting start of packet: HdlcTranslateC
P       = Protocol byte: SerialP
S       = Sequence number byte: SerialP
D       = Packet format dispatch byte: SerialDispatcherC
Payload = Data payload (stored in SerialDispatcherC): SerialDispatcherC
CR      = Two-byte CRC over S to end of Payload: SerialP
F       = Framing byte denoting end of packet: HdlcTranslateC
```

Fig. 2-21 Formato para comunicación serial provista por TinyOS 2.x, usada para el envío de paquetes

Fuente: [35]

Con respecto a la programación del Módulo Base, se codificó un programa que incluye todas las tareas y/o requerimientos planteados para el módulo durante la etapa de diseño. El código fuente del programa se halla incluido en los sección Anexos, sub sección "Código fuente de programas para módulos" bajo el nombre "Base Station".

Para la construcción del Módulo Base, se tomó ventaja de una caja impermeable de la marca Pelican para resguardar las tarjetas electrónicas de factores climáticos. La caja fue adecuada a los requerimientos, por lo cual se instalaron conectores militares tipo hembra para los puertos de comunicación, depuración, sincronización y alimentación. Previendo la necesidad de estandarizar todo lo concerniente a la construcción de los módulos se establecieron normas con respecto al tipo de conector, número de pines de cada unos de ellos y el color de los cables que serán usados para las diferentes conexiones. Tales normas se encuentran en la sección de Anexos bajo el nombre "Normas de construcción de módulos".

Y además y previendo factores geográficos y climáticos que puedan afectar la transmisión y recepción de información a través de la antena interna de la tarjeta IPR2400, así como el hecho de aumentar el rango de cobertura de los módulos que integran la SRIS, se incluyó en la caja una entrada para una antena externa. El diseño de este prototipo esta basado en una antena omnidireccional de 8 dBi del fabricante Pctel. Este diseño fue replicado en la construcción del resto de módulos que conforman la sRIS. La Figura 2-22 presenta algunas imágenes del diseño final del Módulo Base.

Fig. 2-22 Diseño final del Módulo Base

2.2.2.2 Módulo de Sensores

El Módulo de Sensores hace uso de la tarjeta de procesamiento IPR2400 y de la tarjeta de sensores ITS400, además de un radar Reflex VG7 del fabricante HYCONTROL. El radar en cuestión es un sensor que mide la distancia que existe entre su antena y una superficie cualquiera. En este caso, el radar será usado para medir la distancia (altura) que existe entre su antena y la superficie del flujo de lodo y escombros. Los parámetros a medir por este módulo son temperatura, humedad y altura, sin embargo, es posible incluir también, si fuera el caso, parámetros como aceleración e intensidad lumínica. En vista de que el radar opera a 24 V, fue necesario el diseño y construcción de un pequeño circuito electrónico que incluye un convertidor de corriente continua que transforma 12 a 24V. La Figura 2-23 presenta el sensor de altura usado por el Módulo de Sensores.

Fig. 2-23 Sensor Reflex VG7

Fuente: [36]

El Reflex VG7 emplea el principio de onda continua con modulación de frecuencia (FMCW[61]). El radar envía una señal de alta frecuencia sobre una superficie, la cual es reflejada debido al Efecto Doppler[62] y recibida por el radar como una señal de frecuencia más alta que la original.

[61] Frequency Modulated Continuous Wave [78]
[62] Fenómeno físico que consiste en la variación de la frecuencia y la longitud de onda recibidas de las transmitidas.

Esta modulación de frecuencia se debe a que en la fase de medición, la frecuencia de la señal aumenta de forma lineal. La Figura 2-24 presenta el principio de onda continua con modulación de frecuencia.

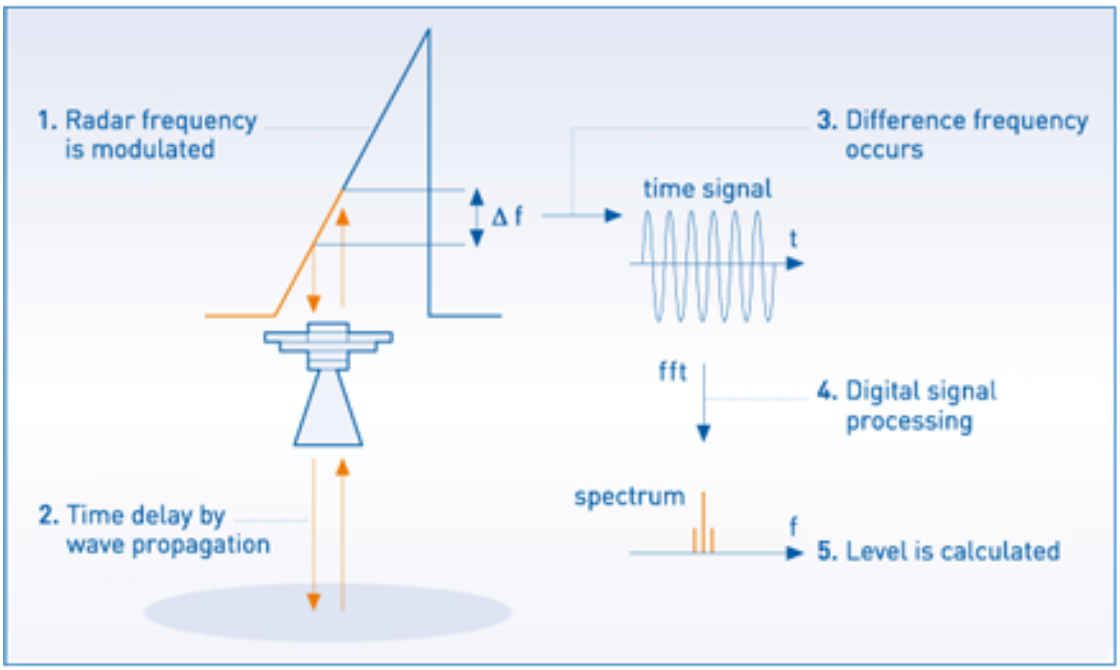


Fig. 2-24 Principio de onda continua con modulación de frecuencia (FMCW)

Fuente: [37]

La distancia es calculada mediante la siguiente relación:

$$tiempo\ retardo = \frac{2d}{c} \qquad (1)$$

De lo cual se desprende la siguiente relación:

$$d = \frac{c * tiempo\ retardo}{2} = \frac{c * \Delta t}{2} = \frac{c * \Delta f}{2 * \left(\frac{df}{dt}\right)} \qquad (2)$$

Donde:

d = distancia medida desde la antena del radar a la superficie del objeto

c = velocidad de la luz

$\frac{df}{dt}$ = cambio de frecuencia por unidad de tiempo

Δf = diferencia de frecuencia

Δt = diferencia de tiempo

El sensor que incluye la tarjeta ITS400 para muestrear temperatura y humedad es un SHT15 del fabricante Sensirion. Con respecto al sensor se puede mencionar que es un sensor de alta precisión con una variación de +/- 0.3 °C y temperatura de operación que va desde -40 hasta 123.8 °C.

Con respecto a la programación del Módulo de Sensores, se codificó un programa que incluye todas las tareas y/o requerimientos planteados para el módulo durante la etapa de diseño. El código fuente del programa se halla incluido en los sección Anexos, sub sección "Código fuente de programas para módulos" bajo el nombre "Sensor Station".

Para la construcción del Módulo de Sensores se replicó el diseño del Módulo Base salvo el número de puertos disponibles. El diseño del Módulo de Sensores establece la instalación de un puerto de entrada para el radar y un puerto de alimentación de corriente continua. Con el objetivo de normalizar el diseño y la construcción del Módulo de Sensores se adoptaron las normas de construcción especificadas en el documento "Normas de construcción de módulos", el cual se halla en la sección Anexos. La Figura 2-25 presenta algunas imágenes del diseño final del Módulo de Sensores.

Fig. 2-25 Diseño final del Módulo de Sensores

2.2.2.3 Módulo Sísmico

El Módulo Sísmico, al igual que el Módulo de Sensores, hace uso de la tarjeta IPR2400 y de uno de los puertos del conversor analógico digital que incluye la tarjeta ITS400. El contar con un conversor analógico digital de 12 bits significa manejar 4096 niveles o comúnmente denominadas cuentas. El geófono[63] usado en el Módulo Sísmico es un modelo L-10AR del fabricante Sercel. Posee una frecuencia de respuesta de 10 a 300 Hz. Cabe mencionar que al trabajar en ese rango de frecuencia evita que el sensor sea saturado por tremores originados por la tierra o un volcán, los cuales se dan bajo los 6 Hz. La Figura 2-26 presenta una gráfica con la respuesta a vibraciones de la tierra de un sismómetro[64] y un sensor AFM.

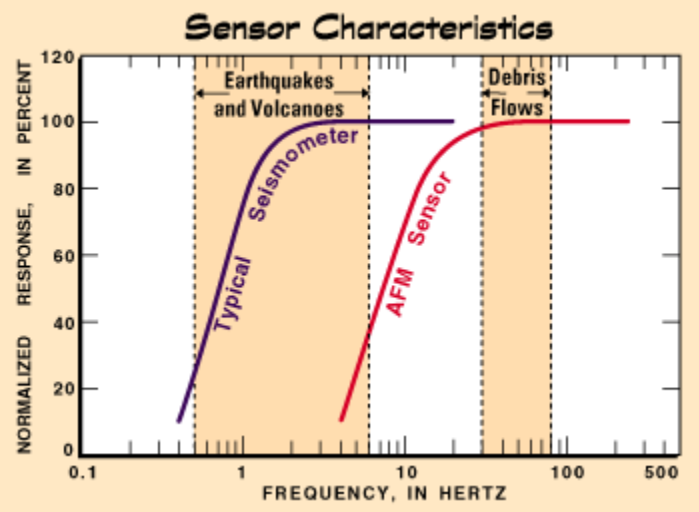


Fig. 2-26 Respuesta a vibraciones de la tierra de un sismómetro y un sensor AFM

Fuente: [38]

El modo de funcionamiento se adapta a algunos parámetros definidos en el estudio de la USGS[65] donde se usó un sensor sísmico para la detección y monitoreo de flujos de lodo y escombros. Tal estudio se llevó a cabo en el volcán Pinatubo en Filipinas y del cual se definieron tres elementos claves para la detección de un flujo de lodo y escombros que son la duración, el umbral y la

[63] Es un dispositivo capaz de transformar la velocidad del movimiento del suelo en una señal eléctrica. Consiste de una bobina sujeta a resortes que se ven afectados por un campo magnético generado por un imán.
[64] Sensor que detecta el movimiento de la tierra.
[65] U.S. Geological Survey. Es una agencia científica del gobierno de los Estados Unidos de América. Sus principales campos de estudio son: Hidrología y Geología.

banda de frecuencia. Para el caso del volcán Pinatubo, se obtuvo que caudales con concentraciones muy grandes de material se concentran en la banda de frecuencias bajas, es decir de 10 – 100 Hz, mientras que caudales con concentraciones normales en la banda de 100 - 300 Hz. Además que el sensor detectó efectivamente el paso de un lahar luego de un tiempo de 45 s y con un valor umbral de 500 mV.

Tomando en cuenta este estudio y el hecho de que se lograron definir tres elementos claves para la detección de un flujo de lodo y escombros, al Módulo Sísmico se lo diseñó integrando estos parámetros, sin descartar el hecho de que este estudio se realizó para un volcán específico y los valores de estos parámetros podrían variar de un volcán a otro. Uno de los tres elementos que se establecieron en el estudio fue la banda de frecuencia. El estudio define dos bandas de frecuencia, la primera comprendida entre 10 y 100 Hz que se halla definida como la componente *low*, donde caudales con concentraciones muy grandes de material están presentes; y la segunda comprendida entre 100 y 300 Hz que se halla definido como la componente *high*, donde caudales con concentraciones normales de material están presentes. Adicionalmente se halla definida una componente *full* que comprende la banda de frecuencia completa, es decir entre 10 y 300 Hz. El Módulo Sísmico implementó un modo "alerta" de funcionamiento, el cual es activado cuando el valor que resulta de obtener el promedio de los valores de la componente *low* supera el valor del umbral experimental. El modo alerta implica que la frecuencia de envío de paquetes aumente. Es así que los paquetes de datos serán enviados cada segundo durante un periodo de 20 s. En modo "normal", sin embargo, la frecuencia de envío se estableció en 20 s. Los parámetros enviados por el Módulo Sísmico son las componentes full, low y high de la señal y un identificador del módulo.

El diseño del Módulo Sísmico estableció la implementación de 3 filtros digitales, que serán aplicados a la señal digital, que definen las 3 bandas de frecuencia de la señal analógica proveniente del sensor. Es así que se diseñaron un filtro para la banda de 10-300Hz, uno para la banda de 10-100Hz y uno para la banda de 100-300 Hz. En lo que tiene que ver con la frecuencia o tasa de muestreo del sensor, ésta se estableció considerando el Teorema de Nyquist[66], que

[66] O Teorema del Muestreo determina que para reconstruir exactamente una forma de onda, la tasa de muestreo debe ser mayor o igual que el doble de la frecuencia máxima a muestrear.

determina que la tasa de muestreo debe ser mayor o igual que el doble de la frecuencia máxima a muestrear. La frecuencia máxima a muestrear es 300 Hz, lo cual implica que la tasa de muestreo mínima se establezca en 600 Hz. El Módulo Sísmico requirió establecer el valor en milisegundos del *timer* que permite establecer la tasa de muestreo; el mínimo valor sería 1 milisegundo que significaría muestrear a 1 KHz, mientras que el valor de 2 milisegundos significaría muestrear a 500 Hz, debido a esta limitación y a motivos experimentales, la tasa de muestreo se estableció en 1 KHz.

El diseño de los filtros digitales está basado en el tipo FIR[67], el cual fue escogido ya que es considerado un tipo de filtro muy estable y también debido a que los efectos causados al usar aritmética de punto fijo son menos severos en comparación a otros tipos de filtros. Por ejemplo, filtros tipo FIR son menos propensos al efecto causado por redondeo en las operaciones aritméticas. Los coeficientes de un filtro pueden ser definidos como una secuencia de valores que definen un filtro. El número de coeficientes definido para cada uno de los filtros diseñados es de 51. La expresión matemática de un filtro FIR viene dada por:

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k) = b_0 x(n) + b_1 x(n-1) + \cdots + b_{M-1} x(n-M+1) + b_0 x(n) \qquad (3)$$

Donde $y(n)$ = señal de salida

$x(n)$ = señal de entrada

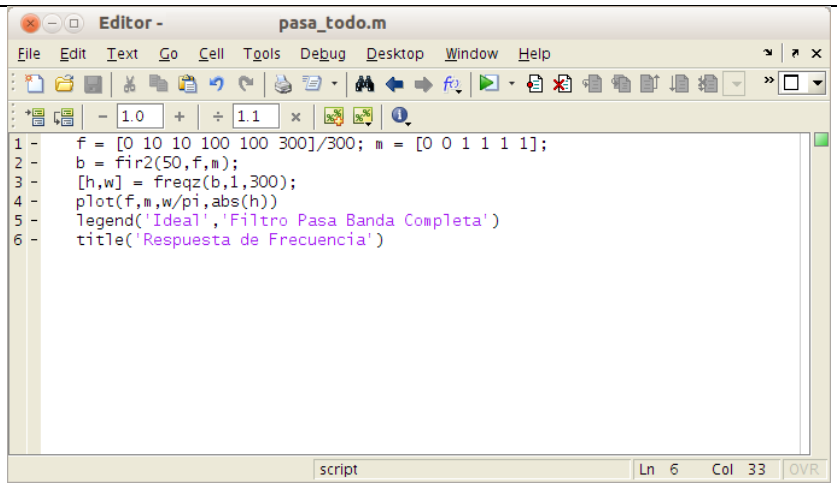
b_k = coeficientes del filtro

y M = orden del filtro o número de coeficientes

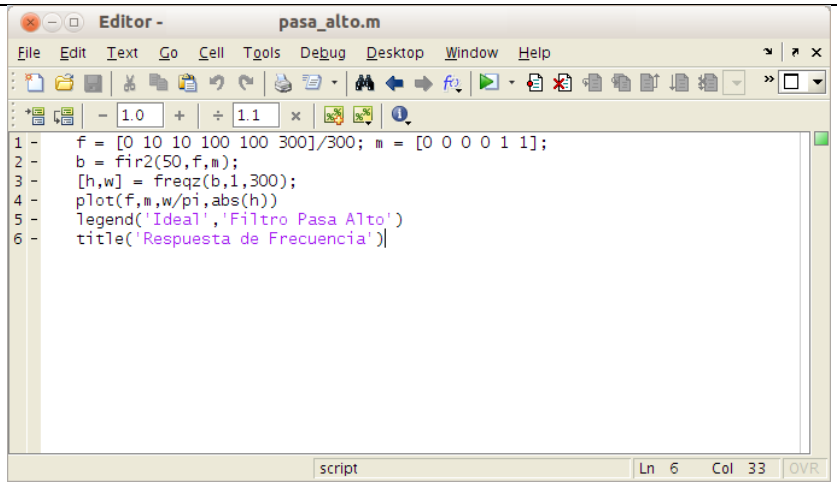
La Figura 2-27 presenta los filtros diseñados en Matlab[68] mientras que la Figura 2-28 la representación gráfica de los mismos.

[67] Finite Impulse Response. Respuesta Finita al Impulso. Es un tipo de filtro digital en el que su respuesta depende únicamente de la señal de entrada y no de valores anteriores de la señal de salida. En otras palabras este tipo de filtro tiene una respuesta finita ya que no son realimentados.

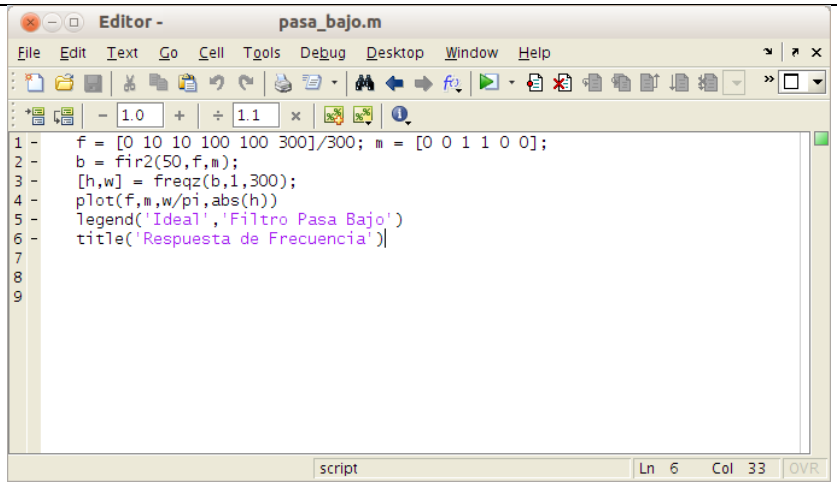
[68] Herramienta de software, distribuida por MathWorks, que integra un lenguaje de alto nivel y un entorno para el cálculo numérico [77].



Pasa banda completa

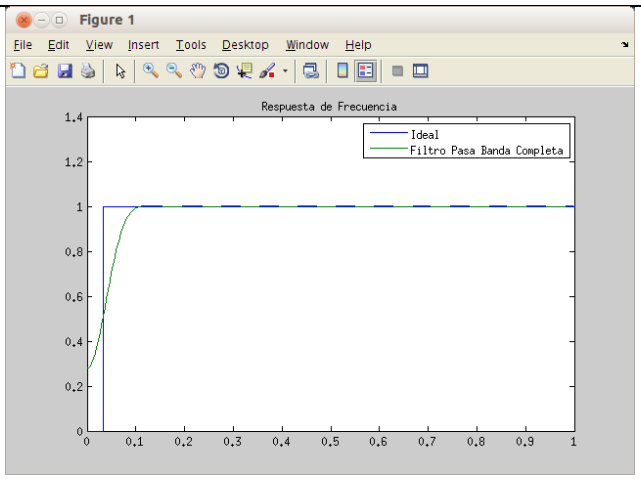


Pasa alto

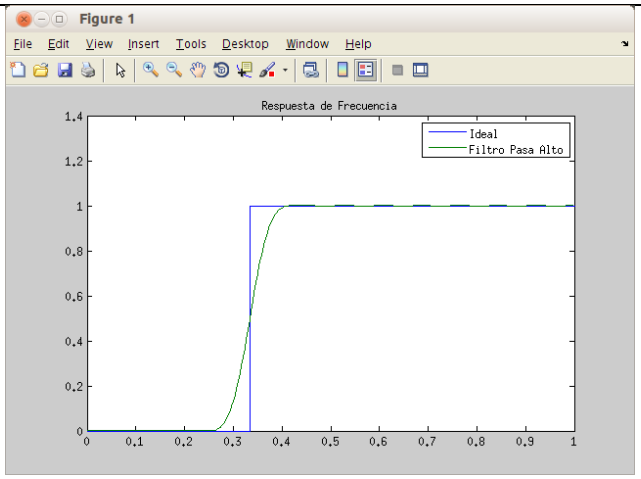


Paso bajo

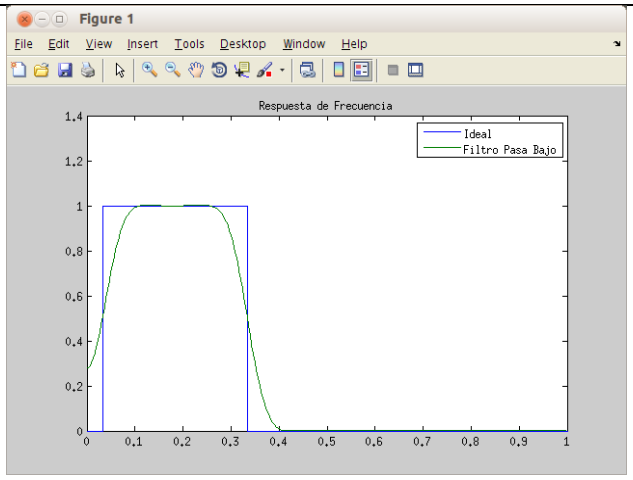
Fig. 2-27 Código de los filtros pasa banda completa, pasa alto y pasa bajo



Pasa banda completa



Pasa alto



Pasa bajo

Fig. 2-28 Gráfica de la respuesta de frecuencia de los filtros pasa banda completa, pasa alto y pasa bajo

Una vez diseñados los filtros, se extrae por medio de Matlab los coeficientes que definen cada uno de los filtros. Tales coeficientes serán aplicados a la señal de entrada, que en este caso será el conjunto de valores devueltos por el proceso de muestreo, para finalmente obtener la señal de salida en base a la expresión matemática para filtros de tipo FIR. Todo este procesamiento constituye la tarea principal del Módulo Sísmico.

Con respecto a la programación del Módulo Sísmico, se codificó un programa que incluye todas las tareas y/o requerimientos planteados para el módulo durante la etapa de diseño. El código fuente del programa se halla incluido en los sección Anexos, sub sección "Código fuente de programas para módulos", bajo el nombre "Seismic Station".

Para la construcción del Módulo Sísmico se replicó el diseño del Módulo Base salvo el número de puertos disponibles. El diseño del Módulo Sísmico establece la instalación de un puerto de entrada para el geófono y un puerto de alimentación de corriente continua. Con el objetivo de normalizar el diseño y la construcción del Módulo Sísmico se adoptaron las normas de construcción especificadas en el documento "Normas de construcción de módulos", el cual se halla en la sección Anexos. La Figura 2-29 presenta algunas imágenes del diseño final del Módulo Sísmico.

Fig. 2-29 Diseño final del Módulo Sísmico

2.2.2.4 Módulo de Visualización

El Módulo de Visualización hace uso de la IPR2400 y de una variante de la tarjeta IEF100 denominada IEF101. Las tareas de este módulo son: mantenerse a la escucha de mensajes de alerta enviados por el Módulo Base e interactuar con una cámara de video de alta resolución y un iluminador infrarrojo a través de uno de los puertos de entrada/salida de uno de los conectores avanzados que posee la IEF101.

Los módulos que conforman la red envían alertas hacia el Módulo Base, el cual las gestiona y dependiendo del caso son retransmitidas hacia el Módulo de Visualización. El Módulo de Visualización dependiendo del estado de la alerta envía una señal eléctrica hacia un puerto de entrada/salida de la cámara. La cámara a través de un script, verifica el estado del puerto y activa o desactiva un relé. Tal relé a su vez hace uso de un pequeño circuito electrónico que enciende o apaga el iluminador infrarrojo. El estado del puerto a su vez determina que la cámara mantenga o no su frecuencia por defecto de transmisión de imágenes.

La cámara usada para este prototipo es la NetCam SC del fabricante StarDot. NetCam SC tiene embebida una versión de uCLinux[69], la cual permite la ejecución de scripts con el objetivo de modificar el comportamiento del dispositivo. NetCam SC también integra un servidor web que permite la configuración del dispositivo; a través de esta interfaz web se definió por ejemplo la frecuencia de transmisión de imágenes y la configuración del servidor FTP al cual serán enviadas las imágenes. El cambio de frecuencia de transmisión de imágenes ante cambios en el puerto entrada/salida de la cámara será controlado a través de dos *bash scripts*. NetCam SC cuenta con un software de visualización denominado StarDot DVR100. Entre sus principales características están el soporte para diferentes modelos de cámaras de la marca StarDot, grabación de video y la capacidad de incluir señales de hasta 100 cámaras.

Con respecto a la programación del Módulo de Visualización, se codificó un programa que incluye todas las tareas y/o requerimientos planteados para el módulo durante la etapa de diseño. El código fuente del programa se halla incluido en la sección Anexos, sub sección “Código

[69] Versión portable de Linux para micro controladores sin unidades de gestión de memoria (MMUs) [83].

fuente de programas para módulos" bajo el nombre "Visualization Station". Además, se codificaron dos scripts para la lectura del puerto, encendido del relé y cambio de frecuencia de transmisión de imágenes de la cámara de video. El código fuente de los scripts se hallan incluidos en la sección Anexos, sub sección "Código fuente de programas para cámara de video", bajo los nombres "active_relay" y "trigger_ftp_relay".

Para la construcción del Módulo de Visualización se replicó el diseño del Módulo Base salvo el número de puertos disponibles. El diseño del Módulo de Visualización establece la instalación de dos puertos de salida, uno para el relé de la cámara y del iluminador y un puerto de alimentación de corriente continua. Con el objetivo de normalizar el diseño y la construcción del Módulo de Visualización se adoptaron las normas de construcción especificadas en el documento "Normas de construcción de módulos", el cual se halla en la sección Anexos. La Figura 2-30 presenta algunas imágenes del diseño final del Módulo de Visualización.

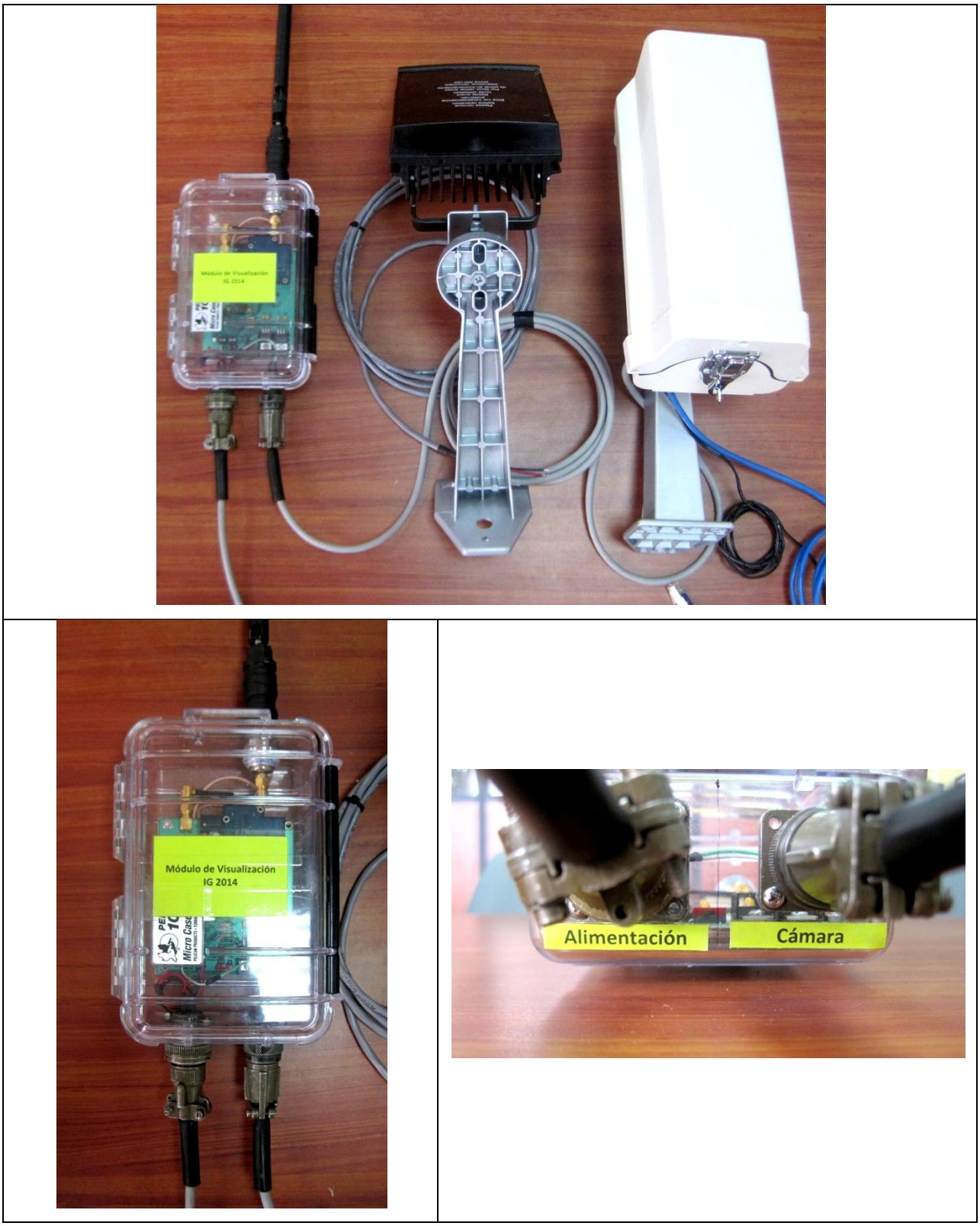


Fig. 2-30 Diseño final del Módulo de Visualización

2.3 Diseño de la red de acceso

La red de acceso (RA) haría uso de la tecnología de espectro expandido para establecer los enlaces de radio necesarios para alcanzar la RM del IG. Para tal objetivo, se planteó el uso de radio módems fabricados por la empresa Freewave, específicamente con su modelo FGR2-PE. Este modelo opera en la banda UHF no licenciada de 902 a 928 MHz, ofrece velocidades de hasta 154 Kbps y alcanza distancias punto a punto con línea de vista de 96 km. El modelo FGR2-PE incluye además un conversor de medios, el cual permite convertir interfaces RS232 a Ethernet[70], Esta funcionalidad es de gran utilidad ya que el puerto de comunicación en el módulo base es de tipo serial y la red microondas del IG provee tráfico Ethernet. El radio modem por lo tanto cumpliría la función de conversor de medios y transmisor/receptor. La Figura 2-31 presenta las interfaces con que cuenta el modelo FGR2-PE.

Fig. 2-31 Vista frontal del radio modem FGR2-PE

Fuente: [39]

La RM del IG maneja grandes volúmenes de tráfico de datos debido a que transporta información de un gran número de dispositivos y estaciones de distinto tipo. Teniendo en cuenta este hecho, se planteó crear una VLAN para la RA, la cual reducirá el dominio de difusión, reducirá el tráfico de la red y por ende mejorará el rendimiento de la misma. Por otro lado, se realizó por parte de los técnicos y geólogos del IG un estudio de campo que llegó a establecer los

[70] Tecnología de conmutación de paquetes en redes de area local.

posibles sitios de instalación de las sRIS. La Tabla 2-3 presenta los posibles sitios de instalación y sus coordenadas, así como algunas de las estaciones de repetición que conforman la RM del IG.

Nombre	Coordenadas	Altura (m)	Tipo	Red
Catalina	0°42'1.30" S 78°34'12.90" O	3186	Estación Final	Cotopaxi
San Elías	0°45'11.31" S 78°32'31.55" O	3199.5	Estación Final	Cotopaxi
Barrancas	0°46'53.92" S 78°30'3.25" O	3438.7	Estación Final	Cotopaxi
Bocatoma	0°28'47.75" S 78°26'36.91" O	3622	Estación Final	Cotopaxi
Rumipamba	0°27'15.9" S 78°25'18.6" O	3139.9	Estación Final	Cotopaxi
Campamento Proaño	0°29'21.88" S 78°25'20.46" O	3432.4	Estación Final	Cotopaxi
Juive	1°25'10.3" S 78°27'44.14" O	2494.5	Estación Final / Repetidora	Tungurahua
Mapayacu	1°30'5" S 78°29'15" O	2753.3	Estación Final	Tungurahua
Vazcún	1°25'34" S 78°25'52" O	1613.2	Estación Final	Tungurahua
Pondoa	1°25'29.9" S 78°27'1.6" O	2870.7	Estación Final	Tungurahua
Ulba	1°24'52" S 78°24'22" O	1379.9	Estación Final	Tungurahua
IG	0°12’55.1” S 78°29’36.5” O	2862.2	Repetidora	IG
Monjas	0°14’12.6” S 78°28’42.2” O	3033.6	Repetidora	IG
Ilinizas	0°39’12.8” S 78°40’18.7”O	3761.2	Repetidora	IG
Sincholagua	0°33’24.24”S 78°24’32.8”O	3962.6	Repetidora	IG

Pilisurco	1°09’17.0” S 78°39’58.0” O	4096.8	Repetidora	IG
Putzalagua	0°57’54.8” S 78°33’43.6” O	3480.2	Repetidora	IG
Loma Grande	1°22’26.3”S 78°27’38.1” O	2962.6	Repetidora	IG
OVT	1°21’29.1” S 78°29’45.3” O	1999.4	Repetidora	IG
Igualata	1°29’24.0”S 78°38’24.0” O	4241.5	Repetidora	IG

Tabla 2-3 Posibles sitios de instalación de las sRIS y estaciones de repetición de la RM

Por medio del software Radio Mobile[71], se realizó el estudio de radio propagación de los posibles sitios de instalación. El estudio determinó los enlaces de radio puno-punto a instalarse para que los sitios alcancen la RM del IG. Para la configuración del sistema de radio fue considerada una antena BMYD890M tipo yagi del fabricante Pctel. La antena trabaja en la banda de 890-960 MHz y su ganancia nominal es 12 dBd. La Figura 2-32 presenta la pantalla de configuración del sistema de radio y la Figura 2-33 presenta la pantalla de parámetros de la red.

[71] Software de simulación de radio propagación desarrollado por Roger Coudé [82].

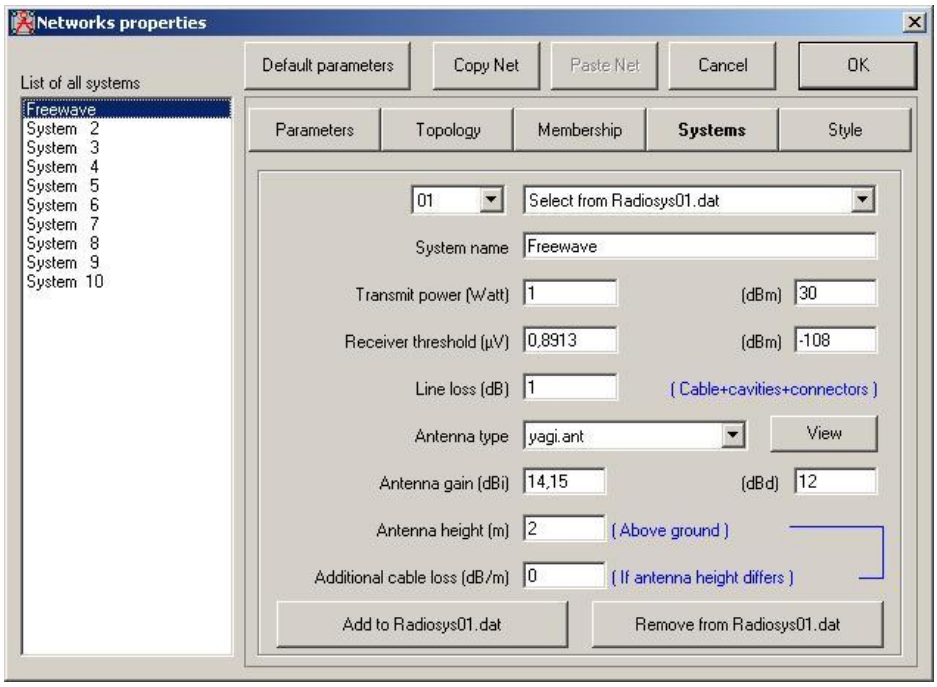
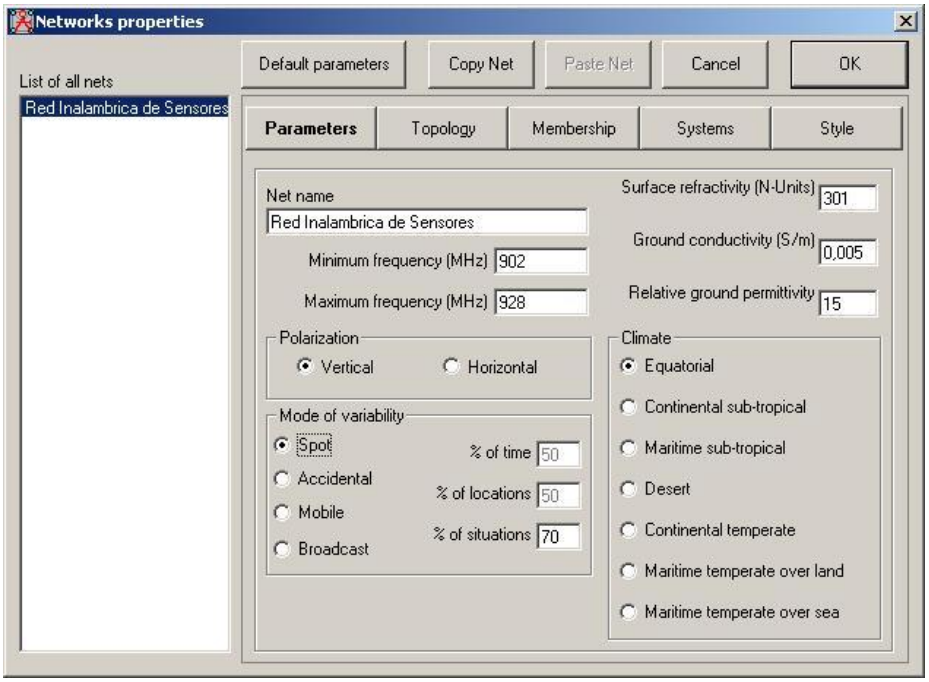


Fig. 2-32 Pantalla de configuración del sistema de radio de Radio Mobile



2-33 Pantalla de parámetros de la red de Radio Mobile

La Figura 2-34 presenta un mapa con las ubicaciones de los posibles sitios de instalación y la Figura 2-35 los enlaces de radio obtenidos del estudio.

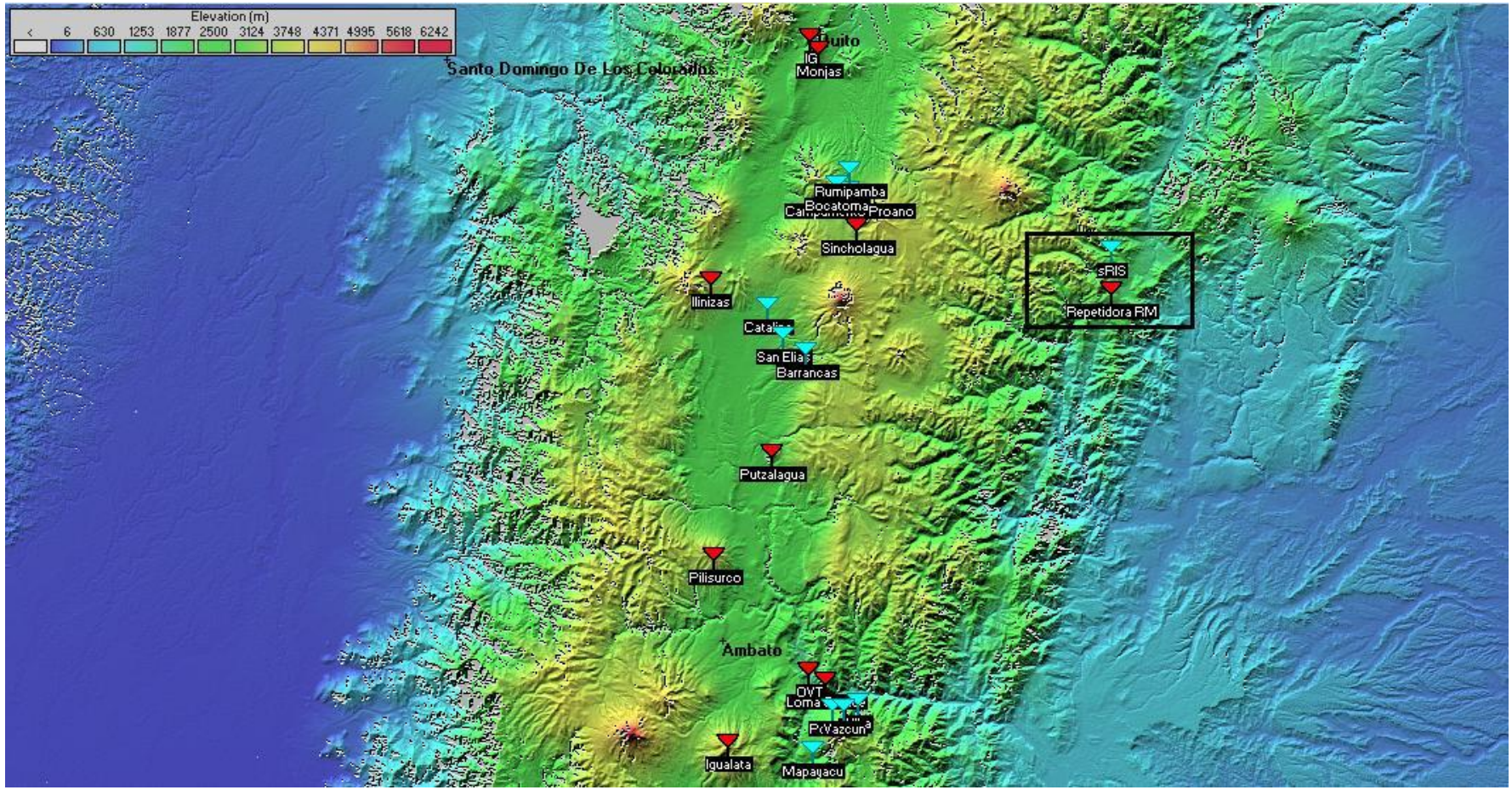
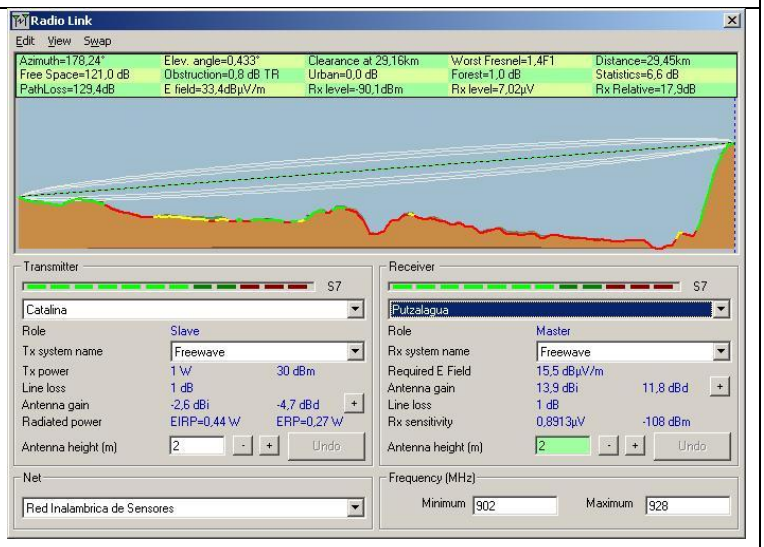
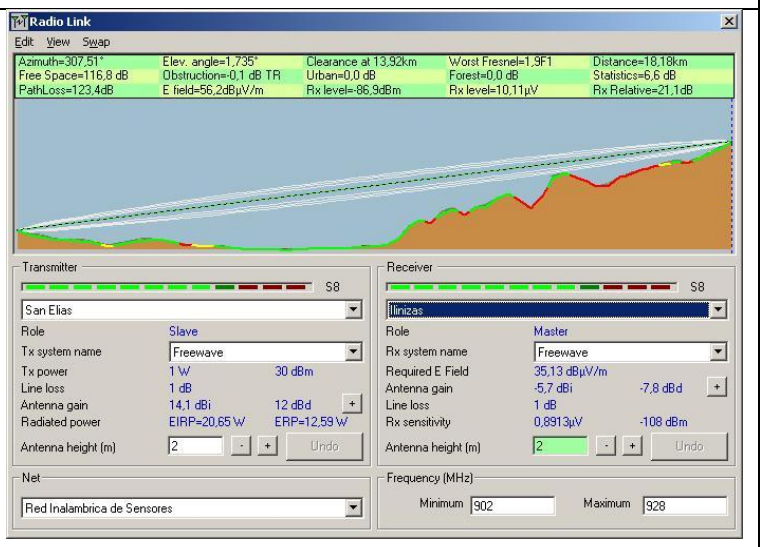
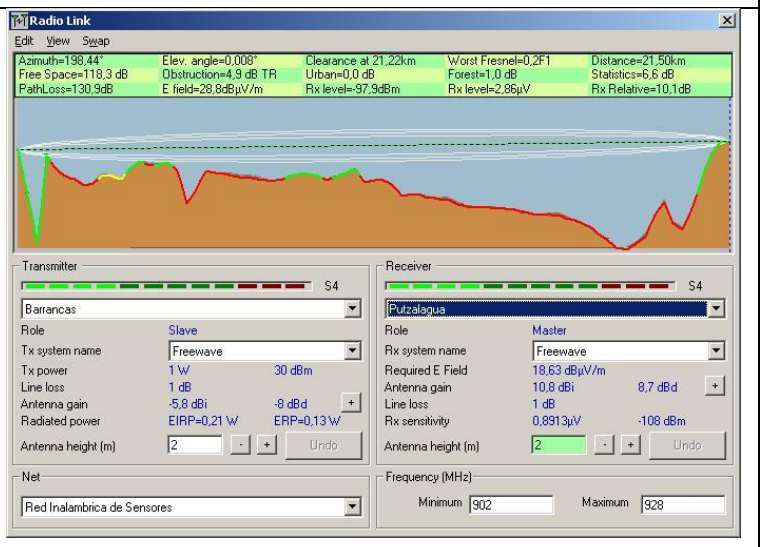
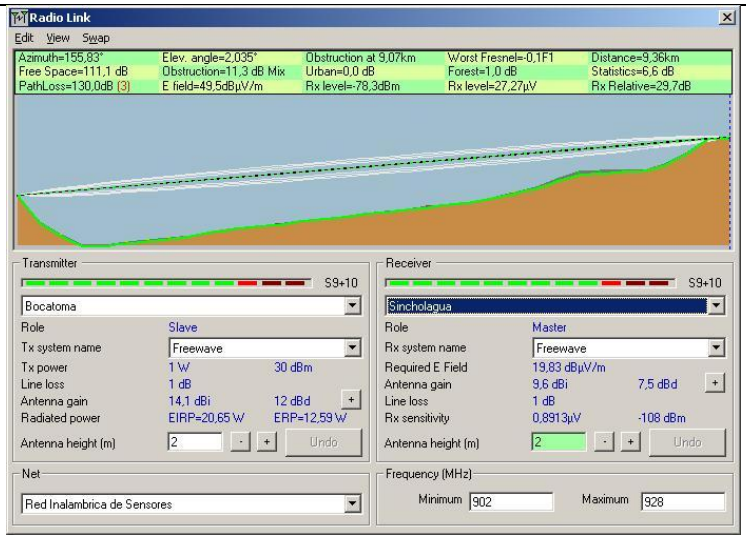
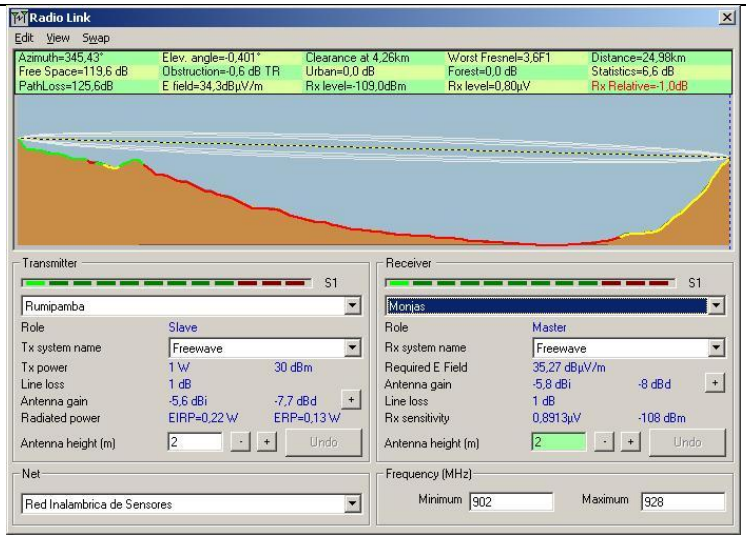
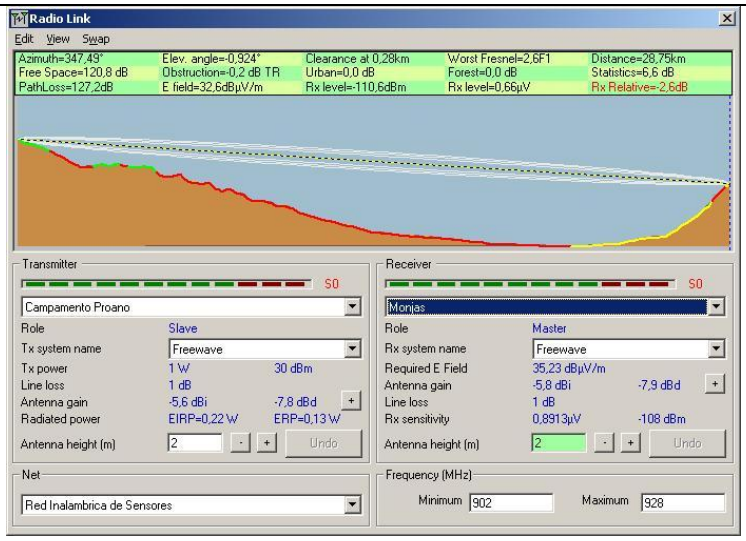
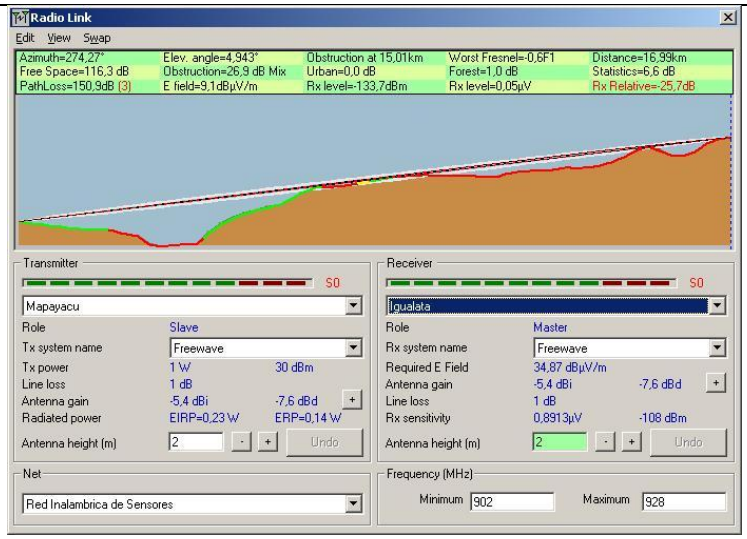
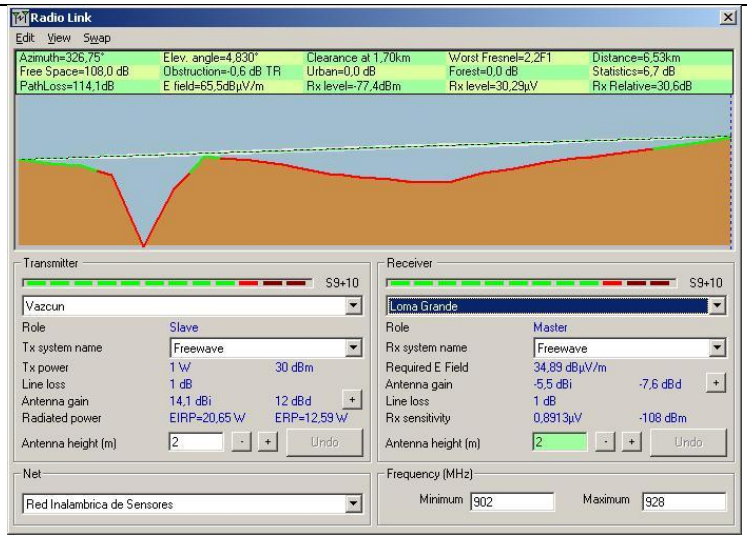
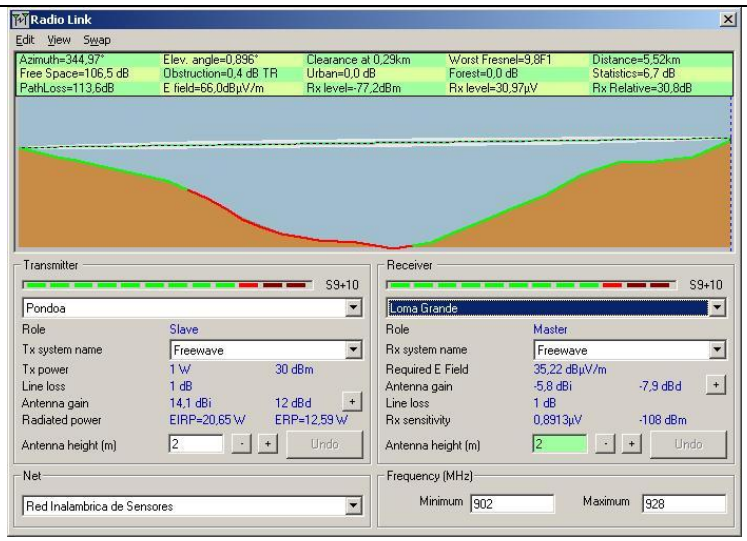


Fig. 2-34 Mapa con las ubicaciones de los posibles sitios de instalación

Nombre	Enlace de radio
Catalina (sRIS) – Putzalagua (R)	
San Elías (sRIS) – Ilinizas (R)	
Barrancas – Putzalagua (R)	

Bocatoma (sRIS) – Sincholagua (R)	
Rumipamba (sRIS) – Monjas (R)	
Campamento Proaño (sRIS) – Monjas (R)	

Mapayacu (sRIS) – Pilisurco (R)	
Vazcún (sRIS) – Loma Grande (R)	
Pondoa (sRIS) - Loma Grande (R)	

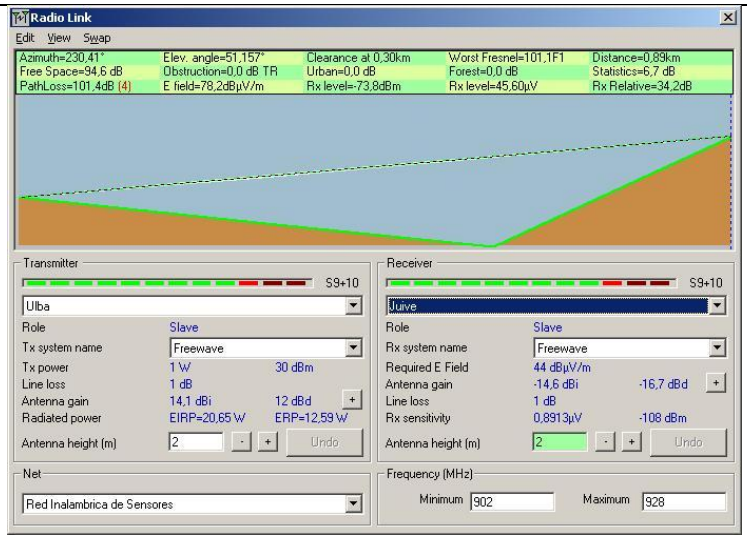
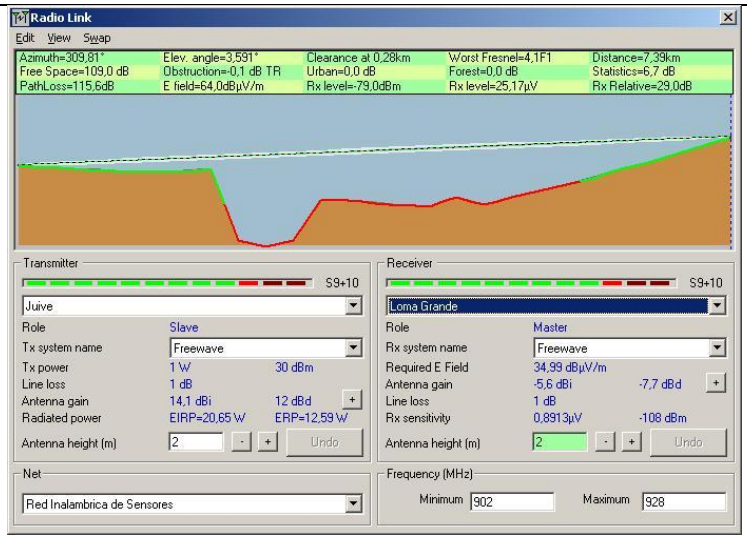
Ulba (sRIS) – Juive (sRIS)	
Juive (sRIS) – Loma Grande (R)	

Fig. 2-35 Enlaces de radio obtenidos a partir del estudio de radio propagación

Considerando los enlaces de radio propuestos, el diseño para la RA es el presentado en la Figura 2-36.

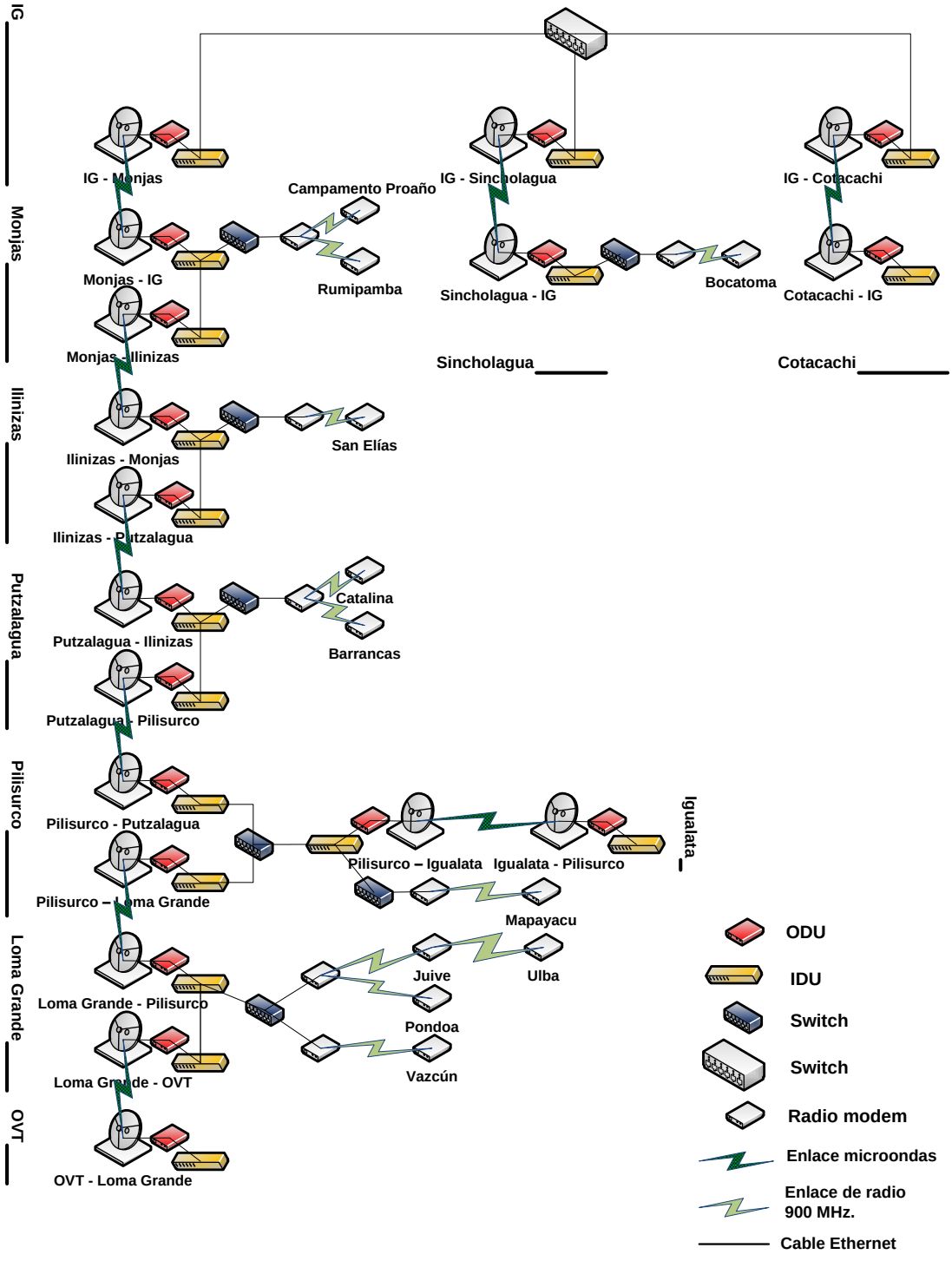


Fig. 2-36 Diseño de la RA

CAPÍTULO 3

SOFTWARE DE ADQUISICIÓN Y MONITOREO

3.1 Sistema de Adquisición y Monitoreo Integrado (SAMI)

SAMI es un software de código abierto y libre distribución, desarrollado por Pablo Marcillo como trabajo de investigación de pregrado [40], y que actualmente es usado por el IG como herramienta de adquisición de estaciones inclinométricas y pluviométricas. SAMI está basado en el lenguaje de programación Java[72] y el motor de base de datos PostgreSQL[73]. SAMI puede ser ejecutado en ambientes Windows, Linux y Mac OS X. Debido a su condición de software de código abierto ha sido posible su modificación, lo cual ha permitido suplir nuevas necesidades y/o requerimientos por parte del IG. El presente trabajo de investigación plantea el uso de SAMI como base para la implementación de un complemento que sirva para adquirir y monitorear información proveniente de la RIS. El diseño y la implementación de tal complemento de software serán abordados en la sección 3.2. A continuación se detallará brevemente las características y funcionalidades más importantes de SAMI, herramienta que será reutilizada para satisfacer uno de los objetivos de este proyecto de investigación.

Una característica importante es que fue programado de forma modular lo cual sin duda ha sido de gran ayuda al momento de integrar nuevos módulos de software. SAMI está conformado por: a) el módulo de adquisición, que es el encargado de discernir la información recibida por los puertos de comunicación; b) el módulo de visualización, que se encarga de presentar la información en pantallas; y, c) el módulo de gestión, que es el encargado de gestionar redes, estaciones, usuarios, dispositivos y puertos de comunicación.

[72] Lenguaje de programación orientado a objetos y distribuido por Oracle Inc [86].
[73] Motor de base de datos de código abierto [87].

SAMI hace uso de APIs[74] comúnmente conocidas como librerías, para implementar nuevas funcionalidades. Destacan de este conjunto la Java Communications API que permite gestionar y manipular interfaces seriales y paralelas; y JFreeChart que proporciona una gama variada de componentes gráficos usados para crear graficas interactivas basadas en la información de la RIS. La Tabla 3-1 presenta una lista de las APIs usadas por SAMI, mientras que la Figura 3-1 presenta la interfaz gráfica de SAMI.

Nombre	Detalle
Java Communications API	Provee la implementación para manipular interfaces basadas en el estándar RS232 e IEEE-1284.
JFreeChart	Proporciona una variedad de componentes gráficos para implementar graficas interactivas y no interactivas.
Java Mail API	Provee la implementación para crear aplicaciones cliente de correo electrónico.
Java Persistence API (JPA)	Provee un modelo de persistencia para el mapeo objeto - relación.

Tabla 3-1 APIs usadas por SAMI

[74] Application Programming Interface. Grupo de funciones o rutinas que permiten interactuar con componentes de software abstrayéndose de la programación de los mismos.

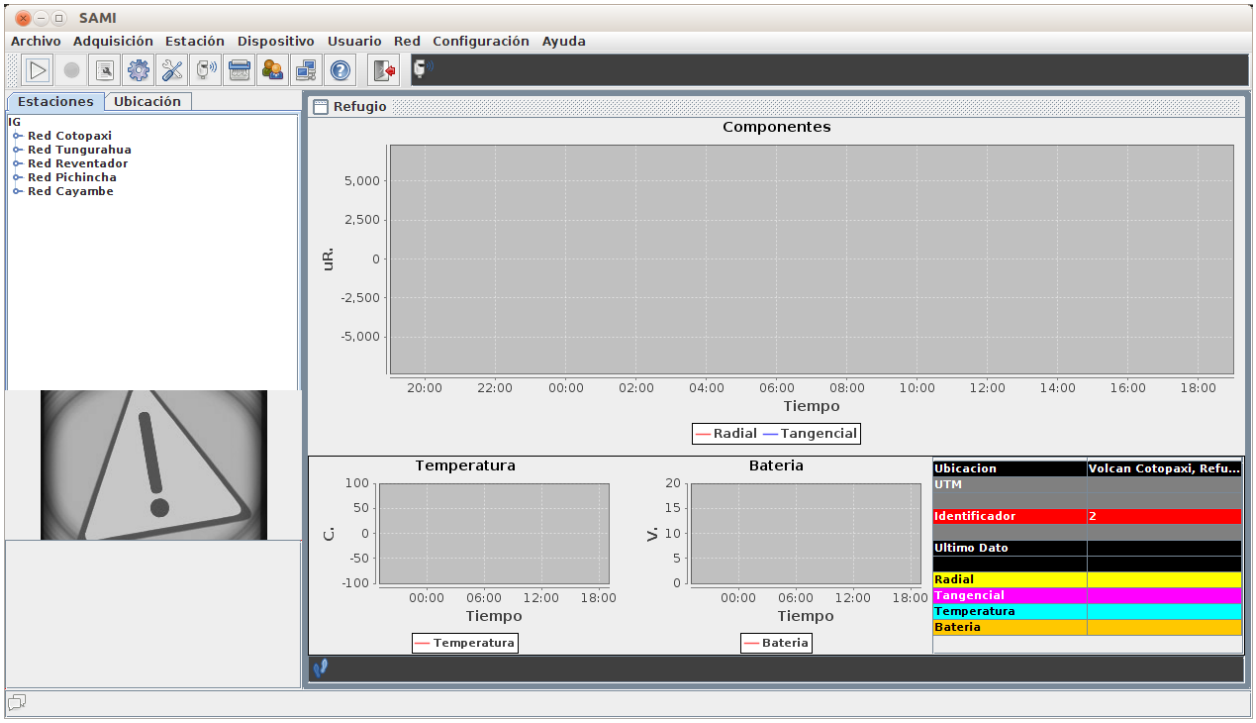


Fig. 3-1 Interfaz gráfica en Linux de SAMI

Entre las herramientas más importantes que provee SAMI, se halla la denominada “Puerto” la cual se encarga de gestionar conexiones y puertos de comunicación. SAMI usa el estándar de comunicación RS232 para llevar a cabo el proceso de adquisición de información. La Figura 3-2 presenta la herramienta “Puerto” provista por SAMI.

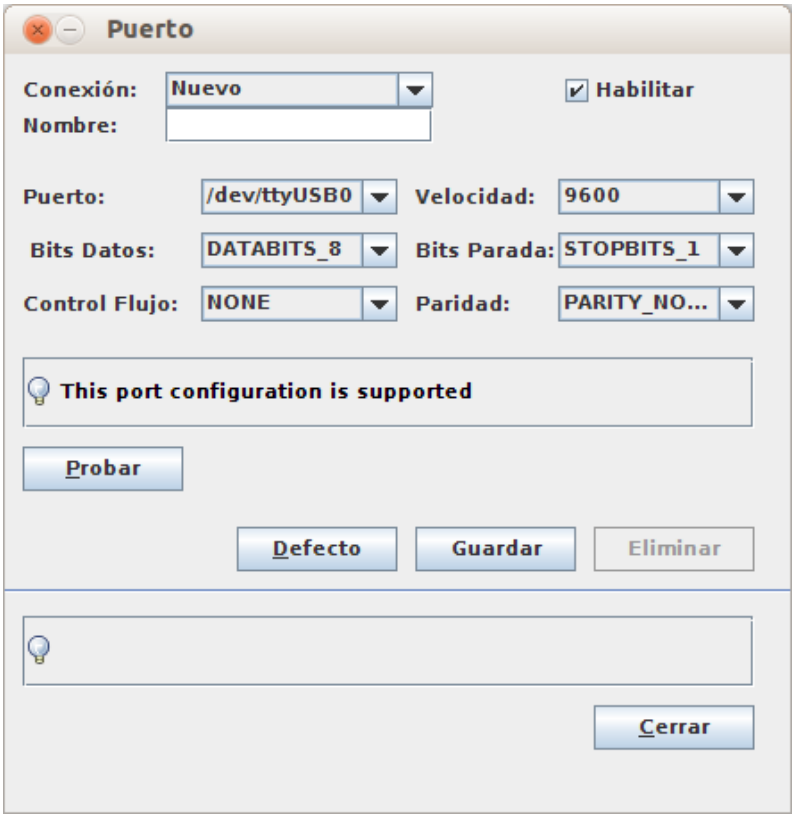


Fig. 3-2 Herramienta "Puerto" de SAMI

3.2 Módulo de software para la RIS

El módulo de software para la RIS hace uso de la implementación para Java del sistema operativo TinyOS. Los paquetes son construidos y empaquetados por el Módulo Base usando las estructuras de datos mencionadas en la Tabla 2-2. La implementación provee un conjunto de clases que permiten discriminar y validar paquetes de datos enviados a través del puerto serial. La primera tarea que se llevó a cabo fue el mapeo de las estructuras de datos a clases Java. TinyOS cuenta con un generador de mensajes para el lenguaje de programación NesC denominada mig[75], que permite realizar esta tarea únicamente especificando las estructuras en un archivo de cabecera (.h). Al ser mig una herramienta para NesC, el archivo de cabecera debe ser definido usando la sintaxis propia del lenguaje.

[75] Generador de mensajes de interfaz para NesC, http://www.tinyos.net/tinyos-1.x/doc/nesc/mig.html.

El archivo de cabecera debe incluir una enumeración que contenga el identificador de cada una de las estructuras de datos seguida del detalle de las mismas. La Figura 3-3 presenta el formato de este archivo de cabecera y la Figura 3-4 presenta un ejemplo de una clase Java generada por la herramienta mig. Las clases resultantes de esta tarea fueron incluidas dentro del paquete *ec.edu.igepn.sami.messages*. La clase MoteIF, provista en el paquete *net.tinyos.message* que se halla disponible en el API de TinyOS, se encarga de registrar un oyente para cada uno de los objetos. Tales oyentes serán invocados cuando los mensajes arriben. Por su parte, el método *messageReceived* es llamado para anunciar la recepción de un mensaje. La Figura 3-5 presenta el código usado para registrar los oyentes y la manera en que los mensajes son recuperados.

```
enum {
    AM_TMPHMDMSG = 1,
    AM_HGHMSG = 2,
    AM_SSMMSG = 3,
    AM_CMRMSG = 4,
};

typedef nx_struct TmpHmdMsg {
    nx_uint8_t alrt;
    nx_uint8_t snid;
    nx_uint16_t tmpr;
    nx_uint16_t hmdt;
} TmpHmdMsg;

...
...
...
```

Fig. 3-3 Formato del archivo de cabecera

```java
package ec.edu.igepn.sami.messages;
/**
 * This class is automatically generated by mig. DO NOT EDIT THIS FILE.
 * This class implements a Java interface to the 'TmpHmdPck'
 * message type.
 */

public class TmpHmdPck extends net.tinyos.message.Message {

    /** The default size of this message type in bytes. */
    public static final int DEFAULT_MESSAGE_SIZE = 14;

    /** The Active Message type associated with this message. */
    public static final int AM_TYPE = 1;

    /** Create a new TmpHmdPck of size 14. */
    public TmpHmdPck() {
        super(DEFAULT_MESSAGE_SIZE);
        amTypeSet(AM_TYPE);
    }
    ...
    ...
    ...
}
```

Fig. 3-4 Ejemplo de una clase Java generada por la herramienta mig

```java
PhoenixSource phoenixSource = BuildSource.makePhoenix(source, null);

MoteIF moteIF = new MoteIF(phoenixSource);
moteIF.registerListener(new HghPck(), this); //registro de un oyente para una instancia de la clase HghPck
moteIF.registerListener(new TmpHmdPck(), this); //registro de un oyente para una instancia de la clase TmpHmdPck
moteIF.registerListener(new SsmPck(), this); //registro de un oyente para una instancia de la clase SsmPck
...
...
...
public void messageReceived(int i, Message msg) {  //metodo que es llamado para anunciar la recepcion de un mensaje
   if (msg.amType() == 1) {
        TmpHmdPck tmpHmdPck = (TmpHmdPck) msg;
   } else if (msg.amType() == 2) {
        HghPck hghPck = (HghPck) msg;
   } else if (msg.amType() == 3) {
        SsmPck ssmPck = (SsmPck) msg;
   }
}
```

Fig. 3-5 Código para registrar oyentes a través de la clase MoteIF y para recuperar los mensajes a través del método messageReceived

Para visualizar la información se diseñó una pantalla que incluye los parámetros generados por la sRIS. La pantalla integra gráficas para las componentes full, high, low; para los parámetros altura, temperatura y humedad; y, una tabla para mostrar información adicional como hora, fecha y número de satélites del receptor GPS. Se establecieron rangos y unidades para el eje vertical para cada una de las gráficas, así para la gráfica relacionada a los componentes full, low y high el rango va de 0 a 1000 cuentas; la del radar de 0 a 40 m;

la del parámetro temperatura de -25 a 75 °C; y finalmente el parámetro humedad de 0 a 100%. Con respecto al eje horizontal, las gráficas presentan intervalos de tiempo de 10 min. La Figura 3-6 presenta la interfaz gráfica de la pantalla mencionada.

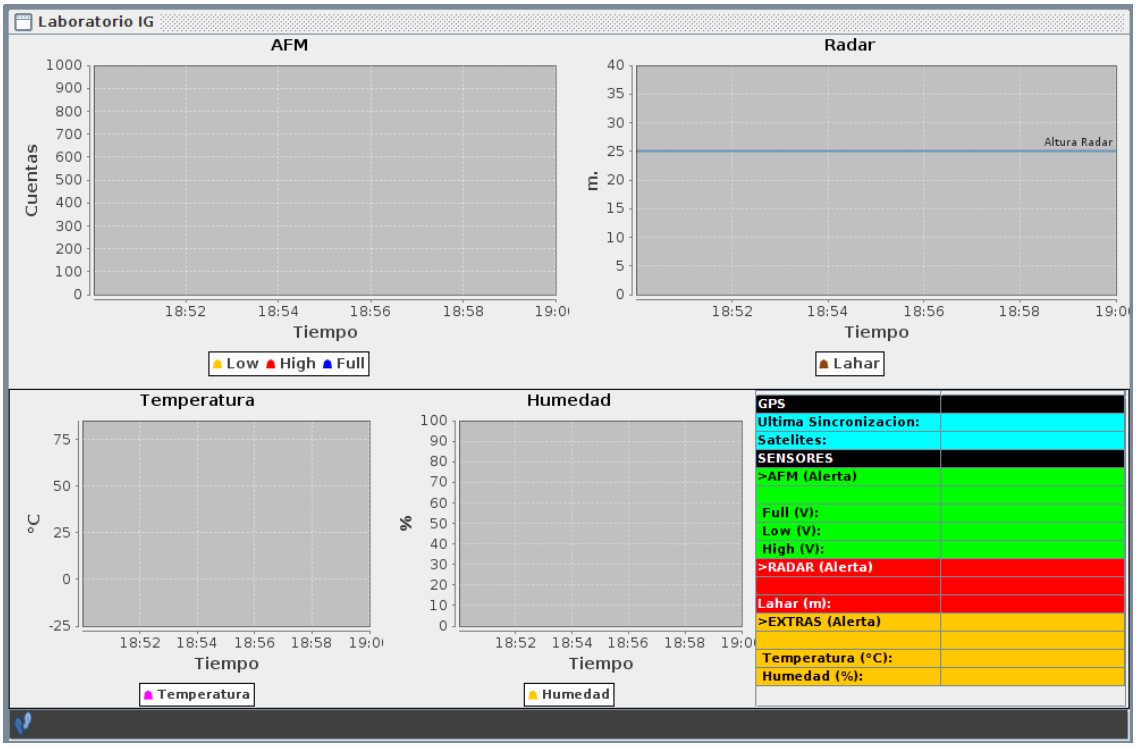


Fig. 3-6 Interfaz gráfica en Linux de la pantalla para la sRIS

El módulo de software debe realizar ciertas conversiones para obtener el valor real de algunos parámetros. Algunas de estas conversiones ya se encuentran definidas, por ejemplo para el sensor de temperatura y humedad se usaron las conversiones provistas en [40].

Para la temperatura se usó la siguiente:

$$temperatura(°C) = cuentas * 0.01 - 39.6 \qquad (4)$$

Mientras que para la humedad se usó la siguiente:

$$humedad(\%) = -0.0000028 * cuentas^2 + 0.04 * cuentas - 4 \qquad (5)$$

La conversión para el sensor de altura fue definida a partir de los datos obtenidos de las pruebas de laboratorio. Como se puede observar en la Figura 3-7, los valores digitales o cuentas mostraron una relación inversamente proporcional a la altura y al voltaje de salida.

								Cuentas		
Altura (m)	Voltios (V)	Cuentas		1 m.	2 m.	3 m.	4 m.	5 m.	6 m.	7 m.
1	2.827	4095		4095	4007	3734	3658	3430	3245	3068
2	2.749	4007		4095	4007	3769	3657	3422	3280	3070
3	2.584	3764		4095	4007	3774	3656	3424	3256	3070
4	2.509	3658		4095	4009	3768	3655	3422	3245	3069
5	2.349	3423		4095	4007	3776	3657	3425	3265	3076
6	2.226	3245		4095	4027	3765	3667	3424	3246	3072
7	2.107	3070		4095	4009	3768	3656	3420	3243	3069
8	2.03	2958		4095	4006	3767	3644	3422	3242	3076
9	1.868	2723		4095	4005	3765	3658	3413	3242	3070
10	1.79	2608		4095	4002	3731	3657	3423	3246	3073
11	1.626	2369		4095	4006	3758	3658	3423	3246	3070
12	1.549	2257		4095	4005	3767	3660	3420	3245	3071
14	1.309	1909		4095	4006	3772	3658	3428	3242	3069
				4095	4004	3763	3663	3424	3247	3070
				4095	4008	3765	3656	3423	3245	3068
				4095	4005	3767	3655	3424	3247	3069
				4095	4004	3764	3660	3424	3244	3070
				4095	4004	3764	3658	3423	3249	3072
				4095	4010	3764	3659	3437	3245	3063
				4095	4005	3764	3656	3447	3251	3067
				4095	4006	3765	3658	3434	3244	3070
				4095	4007	3763	3657	3421	3245	3067
				4095	4007	3766	3658	3422	3245	3070
				4095	4010	3768	3658	3423	3245	3069
				4095	4006	3745	3656	3423	3248	3071

Fig. 3-7 Datos obtenidos a partir de pruebas de laboratorio

Por medio de la herramienta Basic Fitting de Matlab se realizó el ajuste lineal de la relación Altura – Cuentas, con lo cual se evitó posibles errores de dispersión de datos. La Fig. 3-8 presenta la obtención de la ecuación de la recta a través de la herramienta Basic Fitting y la Figura 3-9 presenta una gráfica compartida de la relación Altura – Cuentas y del ajuste lineal realizada a la relación. De esta manera se obtuvo la siguiente conversión para el sensor de altura:

$$altura(m) = \frac{4298.3 - cuentas}{171.64} \qquad (6)$$

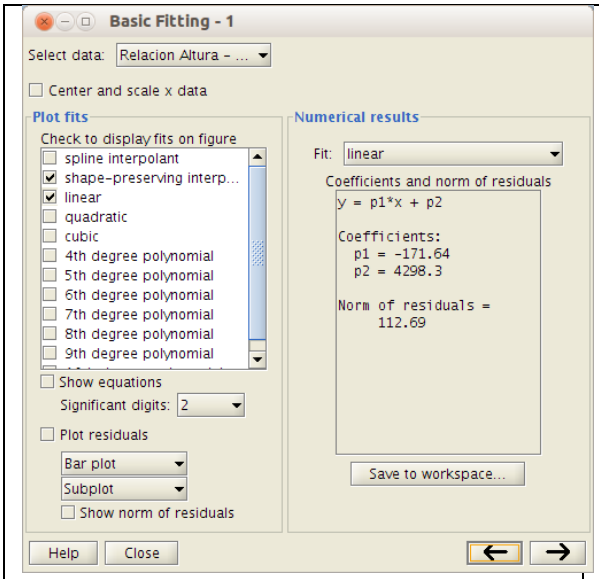


Fig. 3-8 Obtención de la ecuación de la recta de la relación Altura – Cuentas

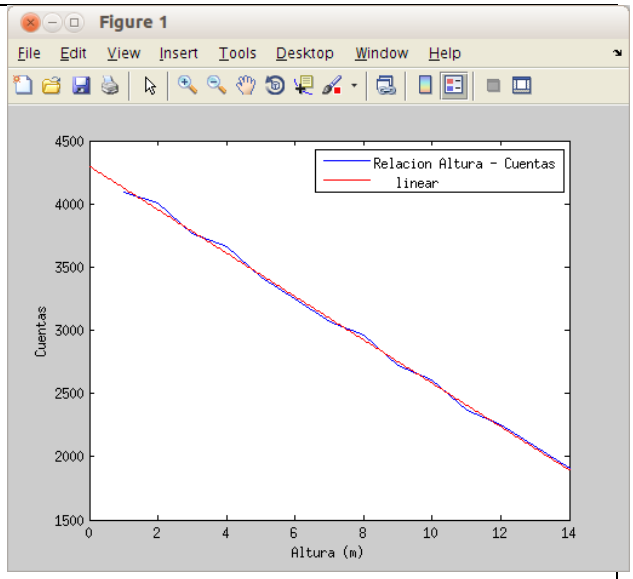


Fig. 3-9 Relación Altura – Cuentas y su ajuste lineal

3.3 Integración del módulo de software al SAMI

Como se mencionó en la sección 3.1, SAMI está conformado por el módulo de adquisición, el módulo de visualización y el módulo de gestión. Dentro del módulo de gestión, SAMI provee una herramienta que permite administrar cada una de las estaciones a ser monitoreadas. La herramienta en cuestión se denomina "Estación"; en un principio tal herramienta permitía únicamente el registro de estaciones de tipo "INCLINOMETRO" y "PLUVIOMETRO". Para este proyecto se incluyó un nuevo tipo de estación denominado "MULTIPARAMETRO". Se lo definió de esta manera debido a que los parámetros medibles de la sRIS provienen de distintos tipos de sensores. La Figura 3-10 presenta la herramienta "Estación" de SAMI. El campo que debe ser considerado sobremanera de la herramienta "Estación" es el definido como *Id. Hardware*, el cual define el identificador de la estación a ser monitoreada. Para el caso de la RIS, el valor del *Id. Hardware* debe coincidir con el identificador de la sRIS. El identificador de la sRIS se halla definido en todas las estructuras de datos de la Tabla 2-2 bajo el nombre *snid*.

Fig. 3-10 Herramienta "Estación" de SAMI

Además se requirió añadir los diferentes componentes que integran una estación de tipo "MULTIPARAMETRO". SAMI interpreta los componentes como los parámetros medibles de un tipo de estación. Para visualizar de mejor manera este concepto se podría citar los componentes de cada uno de los tipos de estación; así, los componentes de una estación tipo "INCLINOMETRO" son: Ejes Radial y Tangencial, Temperatura y Voltaje; para una estación tipo "PLUVIOMETRO" son: Nivel y Voltaje; mientras que para una estación tipo "MULTIPARAMETRO" son: Full, High, Low, Altura, Temperatura y Humedad. La Figura 3-11 presenta el código usado para incluir el nuevo tipo de estación y sus componentes en la clase Java "MainProgram".

```
...
...
...
//clase principal de SAMI
public class MainProgram extends javax.swing.JFrame {
...
...
...
//inclusion del tipo de estacion MULTIPARAMETRO en el arreglo types
types = new String[][]{{Enums.StationType.INCLINOMETRO.name(),String.valueOf(Enums.StationType.INCLINOMETRO.id)},
                       {Enums.StationType.PLUVIOMETRO.name(),String.valueOf(Enums.StationType.PLUVIOMETRO.id)},
                       //el nuevo tipo de estacion MULTIPARAMETRO fue incluido en la enumeracion StationType
                       {Enums.StationType.MULTIPARAMETRO.name(),String.valueOf(Enums.StationType.MULTIPARAMETRO.id)}};

//inclusion de los componentes Full, Low, High, Altura, Temperatura y Humedad en el arreglo components
components = new String[][]{
                            //estacion tipo INCLINOMETRO
                            {String.valueOf(Enums.StationType.INCLINOMETRO.id), "Radial"},
                            {String.valueOf(Enums.StationType.INCLINOMETRO.id), "Tangencial"},
                            {String.valueOf(Enums.StationType.INCLINOMETRO.id), "Temperatura"},
                            {String.valueOf(Enums.StationType.INCLINOMETRO.id), "Voltaje"},
                            //estacion tipo PLUVIOMETRO
                            {String.valueOf(Enums.StationType.PLUVIOMETRO.id), "Nivel"},
                            {String.valueOf(Enums.StationType.PLUVIOMETRO.id), "Voltaje"},
                            //estacion tipo MULTIPARAMETRO
                            {String.valueOf(Enums.StationType.MULTIPARAMETRO.id), "Full"}, //componente Full
                            {String.valueOf(Enums.StationType.MULTIPARAMETRO.id), "Low"}, //componente Low
                            {String.valueOf(Enums.StationType.MULTIPARAMETRO.id), "High"}, //componente High
                            {String.valueOf(Enums.StationType.MULTIPARAMETRO.id), "Altura"}, //componente Altura
                            {String.valueOf(Enums.StationType.MULTIPARAMETRO.id), "Temperatura"}, //componente Temperatura
                            {String.valueOf(Enums.StationType.MULTIPARAMETRO.id), "Humedad"}}; //componente Humedad
...
...
...
}
```

Fig. 3-11 Código usado para incluir el nuevo tipo de estación y sus componentes

Las modificaciones mencionadas en esta sección permitieron la completa integración del módulo de software para la RIS al SAMI. La Figura 3-12 presenta la forma en que las estaciones de tipo “MULTIPARAMETRO” se integraron al árbol de estaciones que provee SAMI.

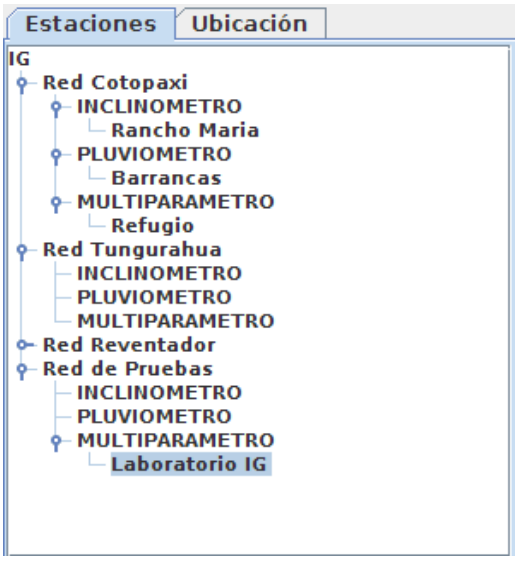


Fig. 3-12 Árbol de estaciones provista por SAMI

CAPÍTULO 4

PRUEBAS Y RESULTADOS

4.1 Pruebas de Laboratorio

El funcionamiento del sistema requirió de pruebas realizadas a cada uno de los módulos que integran la sRIS y de pruebas en conjunto. Se establecieron valores por defecto o valores experimentales para los programas que fueron cargados en cada uno de los módulos que, en su momento, deberán ser analizados y modificados por los científicos y técnicos a cargo. De la misma manera se establecieron parámetros para la cámara de video que integra el Módulo de Visualización. Estos parámetros fueron configurados en los scripts *bash* creados para cambiar la frecuencia de transmisión de imágenes. Se podría citar, por ejemplo, los valores umbrales establecidos para los parámetros de temperatura, humedad y altura; de estos valores umbrales dependerá la generación de alertas en el sistema. La Tabla 4-1 presenta una lista con los parámetros establecidos y sus valores por defecto.

Parámetro	Definición	Valor
ID_SRIS	Identificador de la sRIS. Cada uno de los módulos que integren una sRIS, deberá ser configurado con el mismo valor.	3
THRESHOLD_TEMP	Valor umbral del parámetro temperatura	7960 //40°C
THRESHOLD_HUM	Valor umbral del parámetro humedad	1510 //50%
THRESHOLD_HEIGHT	Valor umbral del parámetro altura	2572 //10 m.
THRESHOLD	Valor umbral de la componente "low"	300 //número de cuentas
NORMAL_DELAY	Intervalo de tiempo establecido para envío de mensajes en modo normal.	10000 o 20000 //10 o 20 s.
ALERT_DELAY	Intervalo de tiempo establecido para envío de mensajes en modo alerta.	1000 o 5000 //1 o 5 s.
SAMPLE_PERIOD	Frecuencia de muestreo para el geófono	2 //2 ms -> 500 Hz.
COEFFICIENTS_NUMBER	Número de coeficientes usados en los filtros digitales	21
period_of_alert	Periodo de tiempo que dura el estado de alerta	30 //30 s
Delay	Periodo de tiempo de espera entre cada transmisión de imagen	2 //2 s

Tabla 4-1 Lista de parámetros con sus valores por defecto, usados por los módulos

4.1.1 Pruebas al Módulo de Sensores

Las pruebas al Módulo de Sensores, específicamente al sensor de altura, requirieron el uso de un obstáculo móvil de la misma manera que se lo había hecho anteriormente para efectuar las mediciones usadas en la etapa de diseño. La superficie del obstáculo móvil intentó simular la superficie de un flujo de lodo y escombros. Tales pruebas consistieron en tomar medidas de voltaje y número de cuentas al módulo, mientras el obstáculo móvil era deslizado por el suelo. Las medidas fueron tomadas sucesivamente a intervalos de 1 metro, hasta cubrir una distancia de 13 metros. Con respecto a los sensores de temperatura y humedad, las pruebas no conllevaron mayor desafío y simplemente se verificó su respuesta

ante cambios de temperatura. La Figura 4-1 presenta la manera en que el sensor de altura fue probado, mientras que las Figuras 4-2 y 4-3 presentan el comportamiento de los sensores de altura, temperatura y humedad ante los diferentes cambios realizados durante la ejecución de las pruebas.

Fig. 4-1 Técnica usada para realizar mediciones con el sensor de altura

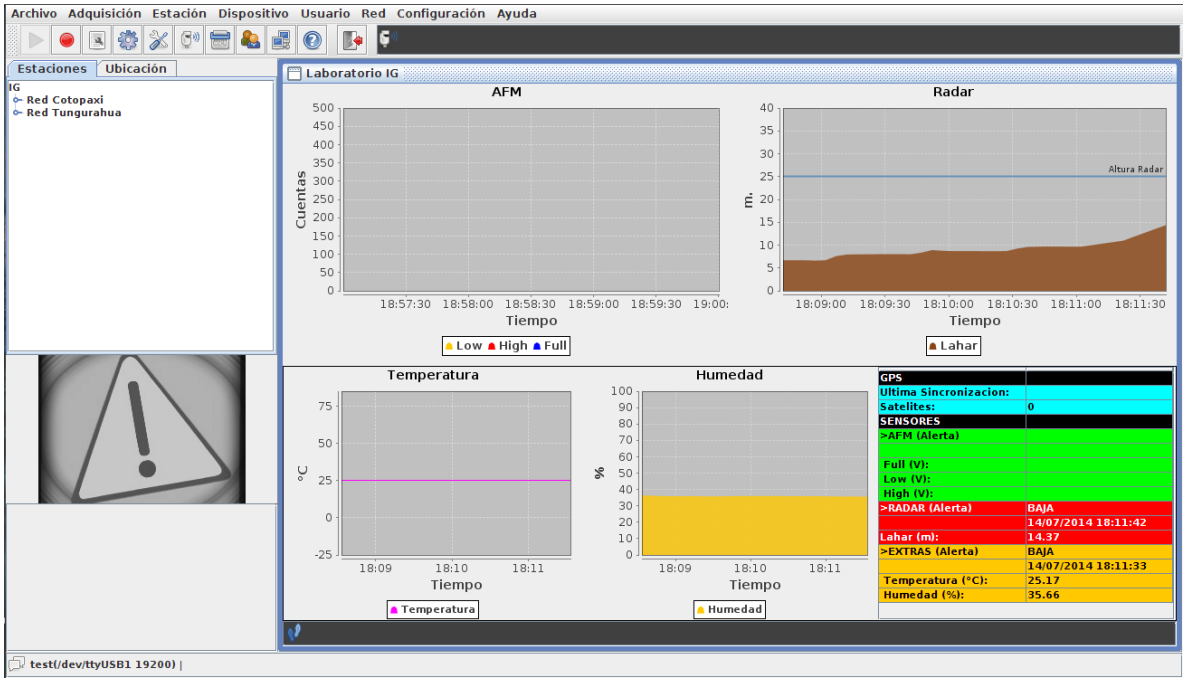
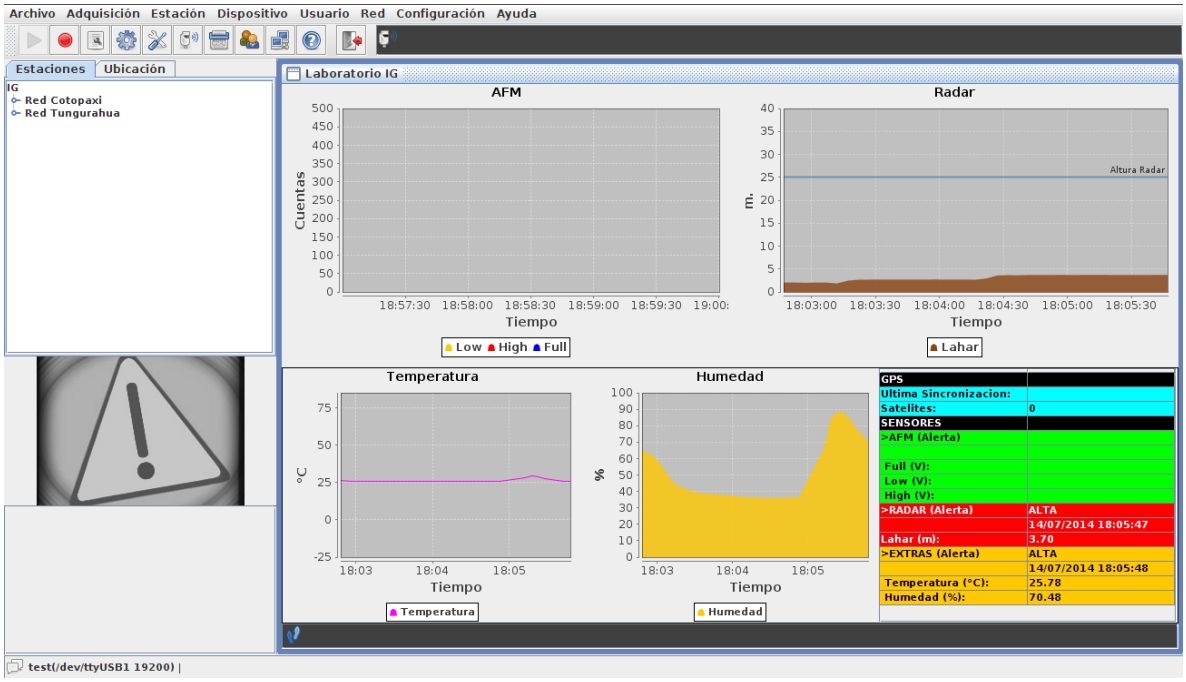


Fig. 4-2 Comportamiento del sensor de altura durante la ejecución de las pruebas (I)

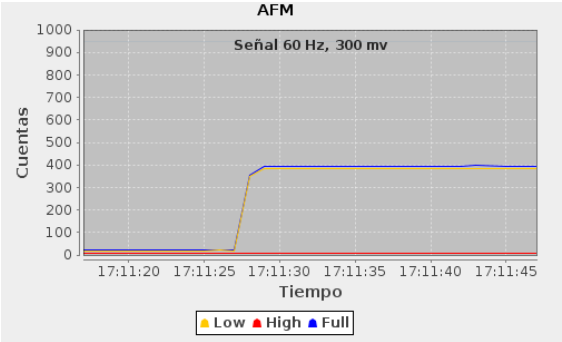
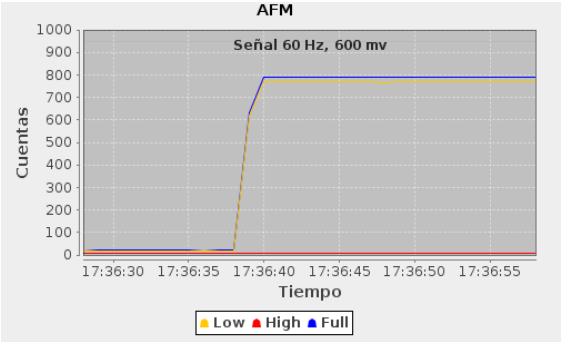
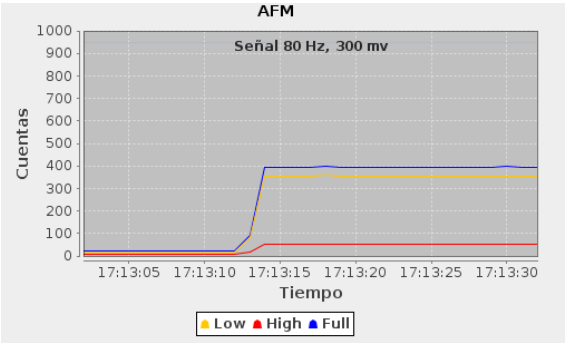
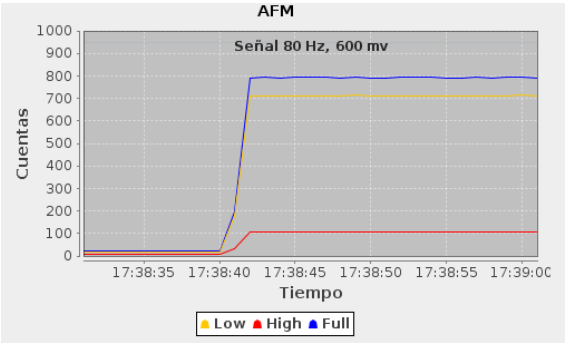
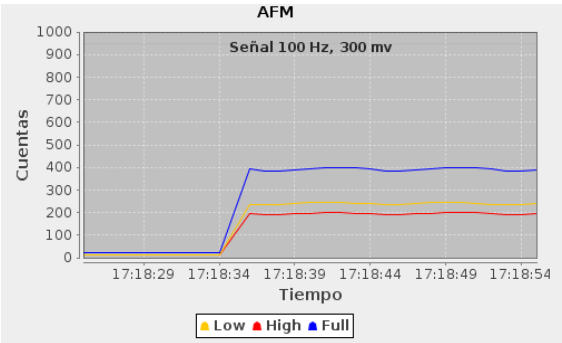
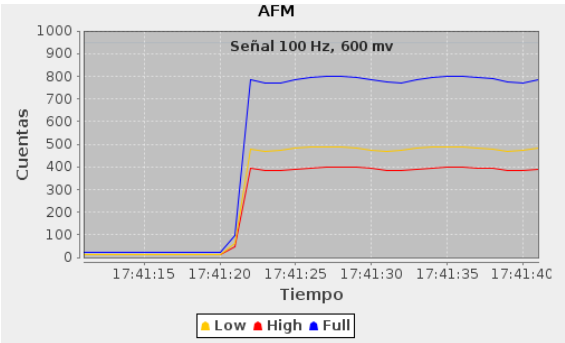


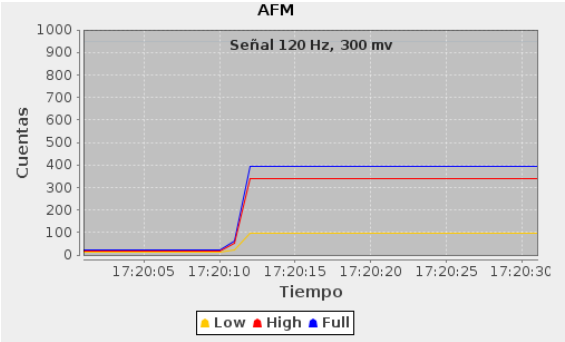
4-3 Comportamiento de los sensores de temperatura y humedad durante la ejecución de las pruebas (II)

4.1.2 Pruebas al Módulo Sísmico

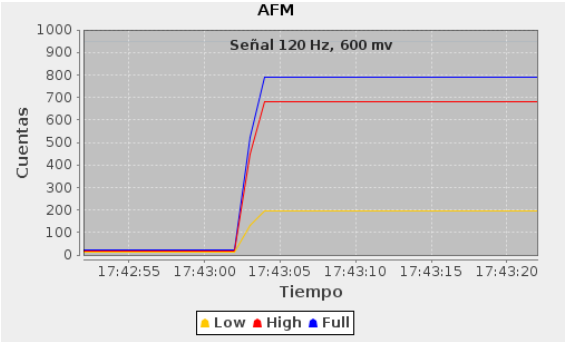
Las pruebas al Módulo Sísmico consistieron en reemplazar el geófono por un generador de funciones, y así generar señales sinusoidales de diferente frecuencia y amplitud. Las pruebas conllevaron la generación de señales sinusoidales de 60, 80, 100, 120, 170 y 220 Hz con variaciones de amplitud de 300 y 600 mV. Como se puede apreciar en la Figura 4-5, y de acuerdo a lo establecido en el fase de diseño, la componente *Low* presenta cambios significativos ante señales de 60, 80 y 100 Hz; mientras que la componente *High* presenta cambios significativos ante señales de 120, 170 y 220 Hz. Este comportamiento demuestra el correcto funcionamiento de los filtros digitales, que en el caso del filtro pasa bajo (*Low*) deja pasar señales que se hallan en el rango de 10 a 100 Hz y descarta señales en el rango de 100 a 300 Hz; a diferencia del pasa alto (*High*), el cual descarta señales que se hallan en el rango de 10 a 100 Hz y deja pasar señales en el rango de 100 a 300 Hz. Por otro lado, el comportamiento de la componente *Full* refleja cambios ante todas las señales generadas, lo cual demuestra el correcto funcionamiento del filtro pasa banda completa (*Full*), al dejar pasar señales que se hallan en los rangos de 10 a 100 Hz (Low) y de 100 a 300 Hz (High). La Figura 4-4 presenta una imagen del Módulo Sísmico durante la fase de pruebas y la Figura 4-5 presenta el comportamiento de cada una de las componentes del AFM ante señales de alta y baja frecuencia y distinto nivel de amplitud.

Fig. 4-4 Módulo Sísmico durante la fase de pruebas

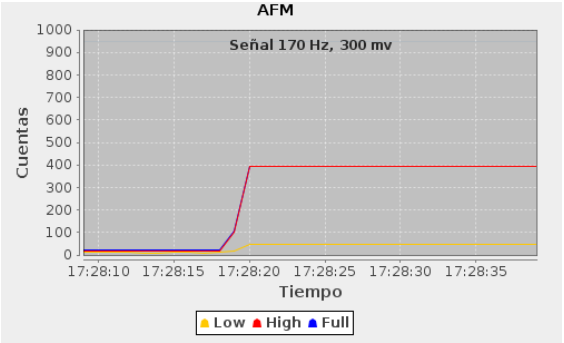
	
Señal 60 Hz, 300 mV	Señal 60 Hz, 600 mV
	
Señal 80 Hz, 300 mV	Señal 80 Hz, 600 mV
	
Señal 100 Hz, 300 mV	Señal 100 Hz, 600 mV



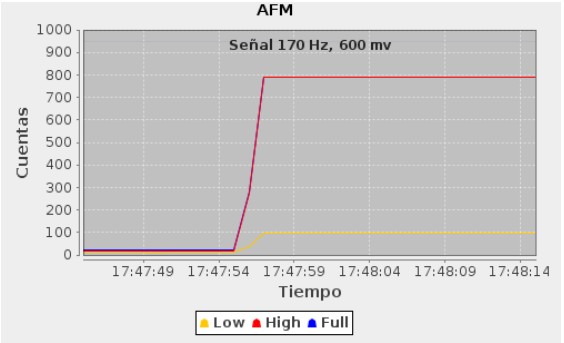
Señal 120 Hz, 300 mV



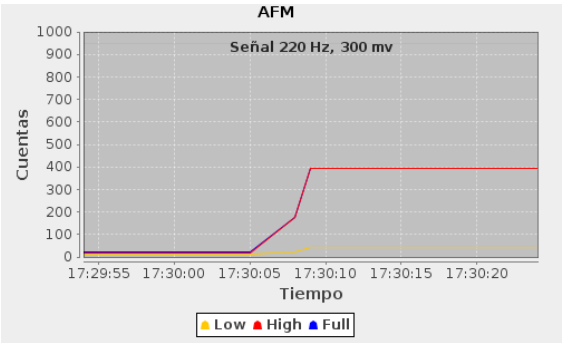
Señal 120 Hz, 600 mV



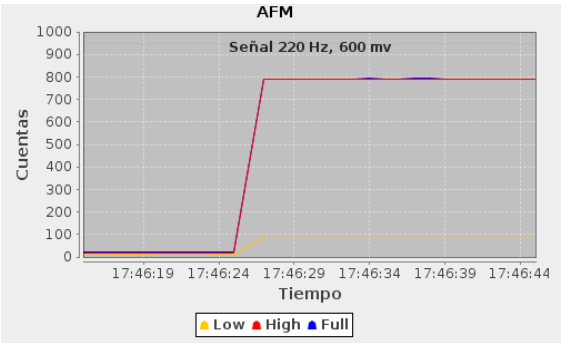
Señal 170 Hz, 300 mV



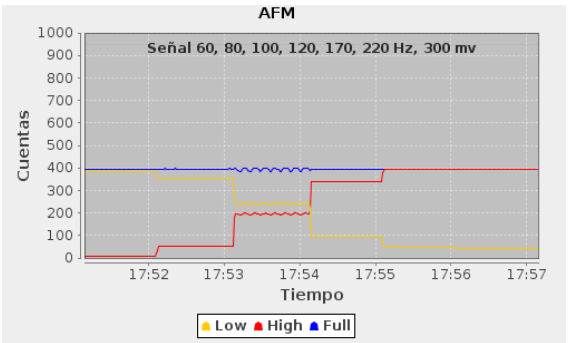
Señal 170 Hz, 600 mV



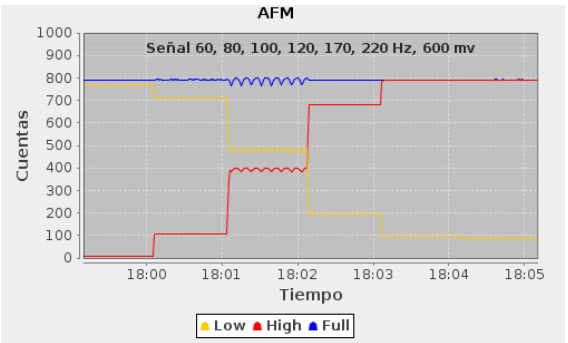
Señal 220 Hz, 300 mV



Señal 220 Hz, 600 mV



Señal 60, 80, 100, 120, 170, 220 Hz, 300 mV



Señal 60, 80, 100, 120, 170, 220 Hz, 600 mV

Fig. 4-5 Comportamiento de las componentes del AFM ante señales de baja y alta frecuencia y distinto nivel de amplitud

4.1.3 Pruebas al Módulo de Visualización

Las pruebas al Módulo de Visualización consistieron en generar alertas por medio del Módulo de Sensores. Los mensajes de alerta fueron retransmitidos hacia el Módulo de Visualización y se generaron los pulsos esperados que por un lado encendieron y apagaron el iluminador infrarrojo y por otro permitieron el cambio de la frecuencia de transmisión de imágenes. Teniendo en cuenta la dificultad de comprobar el encendido y apagado del iluminador infrarrojo durante el día debido al hecho de que el iluminador en cuestión no posee un indicador o *led* de encendido y apagado, se usó un multímetro para comprobar efectivamente la subida y caída de voltaje durante la ejecución de las pruebas. En lo que respecta a las imágenes transmitidas durante el estado de alerta, se usó el servidor FTP del IG para almacenarlas. Como puede ser apreciado en la Figura 4-6, las imágenes han sido nombradas siguiendo el formato *año/mes/día-hora/minuto/segundo*; y almacenadas dentro del directorio *sRIS/backup*. La Figura 4-6 presenta la interfaz gráfica del cliente FTP

durante las pruebas al módulo. La Figura 4-7 presenta una imagen del Módulo de Visualización y del Módulo de Sensores durante la fase de pruebas.

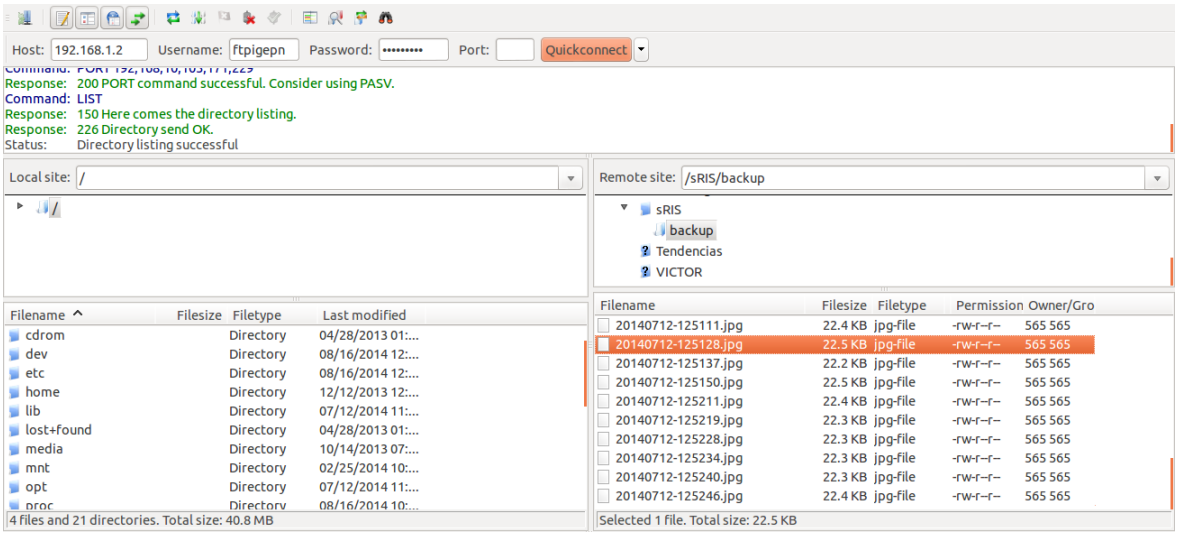


Fig. 4-6 Interfaz gráfica del cliente FTP durante la fase de pruebas

Fig. 4-7 Módulo de Visualización durante la fase de pruebas

4.1.4 Pruebas en conjunto

Se realizó una prueba de funcionamiento en conjunto en la que intervinieron cada uno de los módulos que integran la sRIS. La prueba fue enfocada especialmente al funcionamiento del Módulo Base y su interacción con el resto de módulos. En una primera instancia, se

comprobó el correcto funcionamiento del puerto de depuración tal como se presenta en la Figura 4-8; el puerto de depuración arrojó información relacionada al arribo de mensajes de los distintos módulos e información de los procesos de enganche y desenganche del GPS y de lectura del mismo. Una segunda instancia comprobó el correcto funcionamiento del puerto de comunicaciones del Módulo Base, que puede ser apreciado en la Figura 4-9; además, para comprobar otras funcionalidades del Módulo Base, se simuló un escenario en que todos los sensores que integran la sRIS generaron alertas en el mismo periodo de tiempo. Esto determinó también el correcto funcionamiento del Módulo Base gestionando las alertas del resto de módulos.

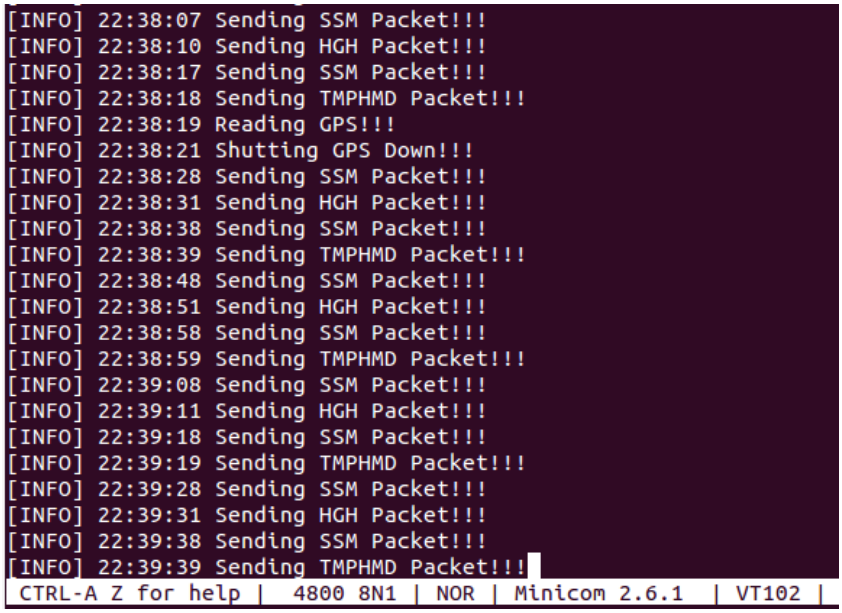


Fig. 4-8 Muestra de la información generada por el puerto de depuración

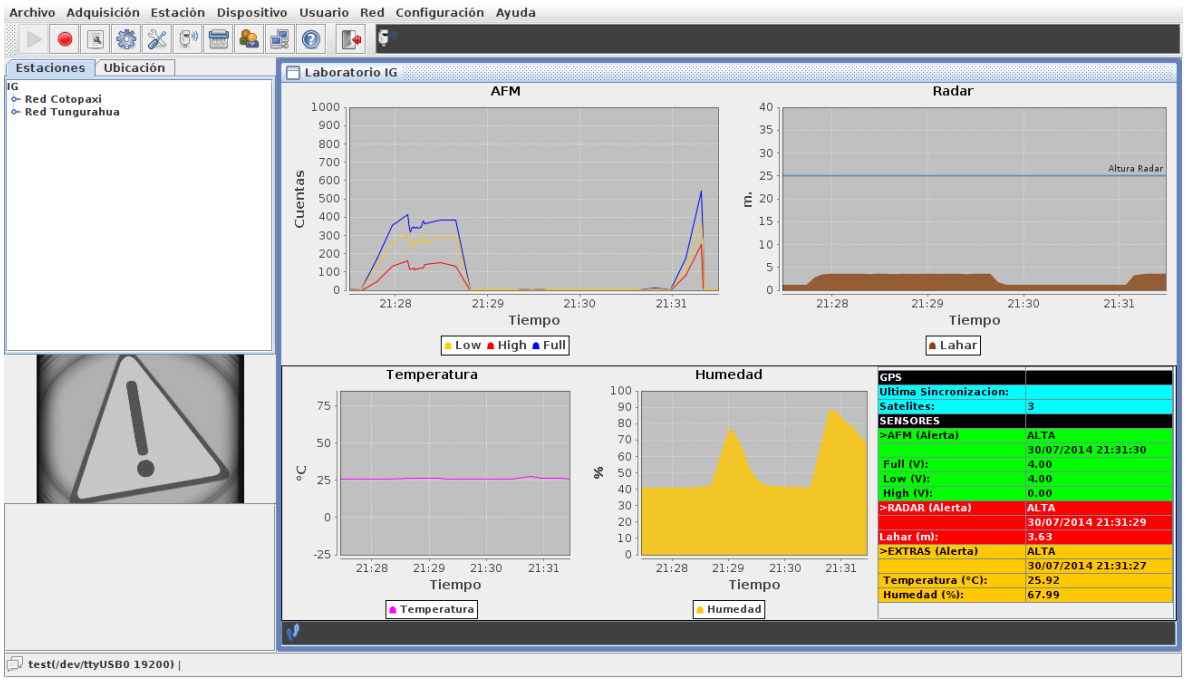


Fig. 4-9 Comportamiento de todos los sensores que integran la sRIS

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- El SMYCF constituye para el IG la primera solución de monitoreo basado en el estándar de comunicaciones IEEE 802.15.4. Por lo tanto, todo lo que conllevó el diseño e implementación del SMYCF servirá como punto base o punto de partida para nuevos proyectos e iniciativas basadas en redes inalámbricas de sensores.

- Existen limitaciones en la sRIS en cuanto a seguridad. Tales limitaciones se deben al hecho de que el diseño de la sRIS esta basada en el estándar de comunicaciones IEEE 802.15.4 y no en especificaciones como por ejemplo ZigBee, que ofrece servicios como ruteo, encriptación, asociación y autenticación, y otros servicios disponibles en su Capa de Aplicaciones.

- La tarjeta de expansión IEF100 que forma parte del Módulo Base, provee funcionalidades de hardware muy específicas que no han sido encontradas en tarjetas de expansión disponibles en el mercado, tal ventaja la convierte en una excelente opción para suplir los requerimientos definidos para tal módulo.

- Se concluyó que el SMYCF en conjunto posee ciertas limitaciones y deficiencias en cuanto a su diseño, funcionamiento, seguridad y autonomía por lo que es necesario que el sistema se halle bajo un continuo desarrollo para así alcanzar la etiqueta de sistema robusto y confiable. En la sección de Recomendaciones se detallan tales limitaciones y sus posibles soluciones.

- Se estableció que la información generada por el SMYCF constituirá un parámetro de calibración del modelo matemático que deberá ser implementado por el IG y que

permitirá la cuantificación de flujos de lodo; mientras que la topografía del terreno, la geometría de las quebradas, los hidrogramas de entrada y los parámetros reológicos[76] constituirán los parámetros de entrada que alimentarán tal modelo.

- La señal analógica del geófono requirió ser acondicionada antes de ser digitalizada por el conversor A/D de 2 maneras distintas. El primer acondicionamiento abarcó la integración de un amplificador de señal. El diseño del preamplificador fue el diseñado y construido en el proyecto de titulación [41] y que está incluido en la sección de Anexos. El segundo acondicionamiento abarcó la integración de un divisor de voltaje o tensión. Una configuración inicial estableció la conexión de los terminales del sensor a la entrada 0 del conversor A/D y a GND, respectivamente. En esta primera configuración la señal oscilaba en 0 V. y por lo tanto el conversor A/D asignó un valor de 0 a toda la parte negativa de la señal. Una segunda configuración estableció el uso de un divisor de voltaje o tensión para que la señal oscile en $V_{max}\,/2$ y de esta manera la parte negativa de la señal no sea descartada. En esta segunda configuración, los terminales del sensor fueron conectados a la entrada 0 del conversor A/D y a la salida del divisor de voltaje, respectivamente. El diseño del divisor de voltaje está incluido en la sección de Anexos.

- Un mayor número de coeficientes para un filtro digital implica mayor carga de procesamiento para el microprocesador, pero por otro lado mejora la calidad del filtro. En un principio se definieron filtros digitales de 51 coeficientes lo cual provocó retardos muy grandes y errores de cálculo. Es así que se optó por disminuir la calidad del filtro para ganar tiempo de procesamiento y así evitar errores de cálculo. El escenario ideal seria trabajar con filtros digitales de calidad aceptable que no provoquen carga de procesamiento excesiva. En la fase de desarrollo e implementación se manejaron tres filtros digitales de 21 coeficientes cada uno y una frecuencia de muestreo de 1 KHz. Esto se transforma en alrededor de 140 operaciones aritméticas entre sumas, multiplicaciones y divisiones sucesivas en intervalos de 1 ms.

[76] La Reologia es una rama de la Física que se encarga del estudio y el flujo del material. La viscosidad es considerado un parámetro reológico.

- El uso de herramientas como Radio Mobile, Google Maps [42], Matlab, Eclipse [43] y Proteus [44] aportaron y ayudaron en gran medida al desarrollo de este proyecto.

5.2 Recomendaciones

- El diseño de la sRIS permite que módulos adicionales puedan ser integrados a la misma. En el caso del IG, se podría requerir por ejemplo, la integración de módulos que interactúen con pluviómetros[77], o módulos que incluyan sensores de gases como CO_2 o SO_2. Estos gases se hallan muy presentes en las emisiones generadas por erupciones volcánicas.

- El consumo de corriente del iluminador infrarrojo es aproximadamente 10 vatios, tal consumo requiere de alimentación adicional; por lo tanto, se recomienda implementar un sistema de alimentación basado en arreglos de baterías y paneles solares.

- Implementar paneles solares para cada uno de los módulos que conforman la sRIS. Algunos modelos de paneles solares para plataformas inalámbricas están disponibles en [45].

- Dotar a los módulos inalámbricos que conforman la sRIS de antenas omnidireccionales de mayor ganancia. Una opción es la antena omnidireccional de 8 dBi del fabricante *Pctel* [46].

- Dotar a la tarjeta IEF100 de un puerto de comunicación tipo Ethernet. Una solución rápida y económica, podría ser la implementación de tal interfaz en base a la ya existente interfaz RS232, a través del dispositivo denominado *XPort* del fabricante *Lantronix* [47].

- Proveer al Módulo de Gestión del SAMI de comunicación tipo Ethernet. Existe un API para el lenguaje de programación Java, para envío y captura de paquetes de red denominada *Jpcap* [48].

[77] Instrumento usado para medir la cantidad de agua precipitada en un cierto lugar. Es muy usado en estaciones meteorológicas.

- La instalación del Módulo de Sensores en los diferentes drenajes conllevará grandes retos debido principalmente a la anchura de las quebradas y al peso que posee el sensor de altura o radar. Es así que se recomienda adquirir radares que incluyan un cable umbilical (comúnmente conocido como extensión de cable), el cual une la parte electrónica y la antena del sensor. Al momento esta opción está disponible para los radares de la serie VF [49].

- En vista de la acogida que podría tener la tarjeta de expansión IEF100, se plantea un rediseño de la misma para su futura distribución y comercialización. Se recomienda que tal rediseño tome en consideración el uso de componentes de montaje superficial en cuanto sea posible, y la inclusión de otro par de conectores avanzados para así mantener la compatibilidad con otras tarjetas de expansión.

- Existe la posibilidad que la sRIS deba coexistir con redes basadas en el estándar IEEE 802.11 b/g (WiFi), así que se recomienda el uso de un analizador de espectro para evitar posibles problemas de interferencia y pérdida de información. En el mercado se halla disponible un dispositivo portátil de bajo consumo denominado Wi-Spy [50] que funciona como un analizador de espectro para las bandas de frecuencia de 2.4 y 5 GHz.

- Dependiendo de la necesidad de establecer un esquema de seguridad en la sRIS, se recomienda el diseño y construcción de una tarjeta de expansión que permita la comunicación inalámbrica bajo la especificación ZigBee. La empresa Digi [29] distribuye un módulo de comunicación basado en la especificación ZigBee denominado XBee.

- Los sensores de altura o radares de los que dispone el Departamento de Instrumentación del IG han sido configurados con valores por defecto, lo cual conlleva limitaciones al usarlos; es por eso que es imprescindible la adquisición de una herramienta universal de configuración o de un módem con tecnología HART para suplir el proceso de configuración, diagnóstico y corrección de errores del radar. Dispositivos con tecnología HART pueden ser encontrados en [51].

- En el Capítulo 2 se mencionó la posibilidad de adoptar una topología tipo malla para la sRIS, en la cual nodos finales actúen también como repetidores. Esta nueva funcionalidad requiere de la implementación y optimización de código. En el caso del Módulo Sísmico, la carga de procesamiento es tan alta que se requiere previamente la optimización del código ya existente para la inclusión de uno nuevo para tareas adicionales. Por ejemplo, personal del IG comentaron el interés de instalar una sRIS basada en un arreglo de Módulos Sísmicos instalados a lo largo de una quebrada, con el objetivo de estudiar el comportamiento de los flujos de lodo en quebradas especificas. El no contar con la funcionalidad mencionada anteriormente haría que la aplicación propuesta por el personal del IG demande la instalación de Módulos Base cada cierta distancia en lugar de Módulos Sísmicos que actúen como repetidores hasta alcanzar un solo Módulo Base.

- La RM del IG maneja gran cantidad de información por lo que se recomienda configurar una VLAN para la RA del SMYCF y así reducir el dominio de difusión. Tal reducción implicará una mejora significativa en el rendimiento de la red.

- Para la construcción de cada uno de los módulos que integran la sRIS se usaron conectores tipo militar de diferente número de pines y además se definió un patrón de colores para los cables. Por lo tanto, se recomienda el uso de estas normas en la construcción de los nuevos módulos. Tales normas están definidas en el documento denominado “Normas para la construcción de módulos”, el cual se halla incluido en la sección Anexos.

- La primera versión de la tarjeta de interfaces IEF100 contemplaba el uso de las 3 interfaces seriales de alta velocidad (STUART, BTUART y FFUART) para implementar los puertos de comunicación, depuración y sincronización. Sin embargo la interfaz BTUART no funcionó de acuerdo a lo esperado, y es así que se improvisó de manera que la línea de transmisión disponible del puerto de sincronización (FFUART) fue usada como puerto de depuración. Algún requerimiento muy específico podría

requerir el uso y disponibilidad de tal interfaz por lo que se recomienda realizar una exhaustiva investigación en búsqueda del problema y su posible solución.

- Se recomienda hacer uso de la característica denominada Escalamiento Dinámico de Voltaje incluida en la plataforma imote2. En el caso del Módulo Base que tiene a su cargo la ejecución de varias tareas simultáneas, el contar con un microprocesador que procese información y libere recursos de hardware más rápidamente sería fundamental. En la fase de diseño, el Módulo Base no requirió el uso de esta característica, sin embargo requerimientos propios del sistema o la integración de módulos adicionales provocarían una caída en el desempeño de la sRIS.

- Se recomienda visitar el grupo de desarrollo Intel Mote2 Community [52]. Este grupo de desarrollo provee gran cantidad de información y soporte para la plataforma imote2. Otra fuente de soporte técnico es la provista por el grupo TinyOS Community Forum [53] a través de sus contribuciones de código. Este grupo por su parte provee aplicaciones y soporte para distintas plataformas inalámbricas de sensores.

REFERENCIAS BIBLIOGRÁFICAS

[1] SciDevNet. [Online]. http://www.scidev.net/america-latina/comunicacion/especial/alerta-temprana-contra-desastres-hechos-y-cifras.html

[2] Daniel et al. Andrade. (2006) Lahar early warning systems based on acoustic flow monitors in Ecuador.

[3] Daniel Andrade, Patricio Ramón, and et al. (2005) Sistemas de Detección de Lahares y Alertas Tempranas: Aplicaciones y Resultados en los volcanes del Ecuador.

[4] IEEE. (2012) IEE 802.15 Grouping Group for WPAN. [Online]. http://grouper.ieee.org/groups/802/15/

[5] Greg Hackmann. (2006, Marzo) [Online]. http://www.cse.wustl.edu/~jain/cse574-06/ftp/wpans/index.html

[6] Laboratorio Didattico di Informatica Avanzata, Politecnico di Torino. [Online]. http://www.labinf.polito.it/~s100675/bt_air.html

[7] IEEE Computer Society, "Part 15.2: Coexistence of Wireless Personal Area Networks with Other Wireless Devices Operating in Unlicensed Frequency Bands," *IEEE-SA Standards Board*, Agosto 2003.

[8] ICPDAS-USA. [Online]. http://www.icpdas-usa.com/zigbeeintro.php

[9] Prism Model Checker. [Online]. http://www.prismmodelchecker.org/casestudies/zigbee.php

[10] Claudio Rossi and et al., "Wireless Sensor Networks for Planetary Exploration: Experimental Assessment of Communication and Deployment," *Advances in Space Research*, 2013.

[11] Memsic Inc. [Online]. http://www.memsic.com/

[12] Libelium. [Online]. http://www.libelium.com/

[13] Memsic. TelosB.

[14] Memsic. IRIS.

[15] Libelium. Waspmote Datasheet.

[16] FreeRTOS. [Online]. http://www.freertos.org/

[17] eCOS. [Online]. http://ecos.sourceware.org/

[18] ArdOS. [Online]. https://bitbucket.org/ctank/ardos/wiki/Home

[19] Contiki. [Online]. http://www.contiki-os.org/

[20] TinyOS. [Online]. http://www.tinyos.net/

[21] Philip Levis and et al., "TinyOS: An Operating System for Sensor Networks".

[22] Contiki Wiki. [Online]. https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja

[23] School of Engineering of Zurich University. [Online]. http://www.engineering.zhaw.ch/en/engineering/institutes-centres/ines/research/wireless/wireless-technologies-wpan/zigbee.html

[24] Zigbee Alliance. ZigBee Alliance. [Online]. http://www.zigbee.org/

[25] Sinem Coleri Ergen. (2004, Septiembre) ZigBee/IEEE 802.15.4 Summary.

[26] ZigBee AFG Chair Phil Jamieson. (2007, Septiembre) ZigBee Application Profiles.

[27] Alberto Bielsa. (2012, Enero) Smart City project in Serbia for enviromental monitoring by Public Transportation.

[28] Javier Solobera. (2010, Septiembre) Detenting Forest Fires with Wireless Sensor Networks.

[29] Geoffrey Werner-Allen, Konrad Lorincz, Matt Welsh, and et al., "Deploying a Wireless Sensor Network on an active volcano," *IEEE Internet Computing*, p. 25, Marzo 2006.

[30] Der Richtfunk Shop. [Online]. http://www.richtfunk-shop.com/

[31] Crossbow. (2007, Septiembre) Imote2 Hardware Reference Manual.

[32] B.F. Spencer. (2011, Octubre) Wireless Monitoring of Civil Infrastructure Comes of Age. [Online]. http://www.structuremag.org/article.aspx?articleID=1328

[33] Université de Pau et des Pays de l'Adour. [Online]. http://web.univ-pau.fr/~cpham/WSN/InstallationTinyOS.html.

[34] Joe Mehaffey and et al. [Online]. http://www.gpsinformation.org/dale/nmea.htm

[35] TinyOS. (2013, Enero) [Online]. http://www.tinyos.net/tinyos-2.1.0/doc/

[36] Switches International. [Online]. www.switches.co.za

[37] Krohne. [Online]. http://krohne.com/en/industries/marine-industry/tank-monitoring-systems/optiwave-cargo-level-radar/measuring-principle-fmcw/

[38] USGS. [Online]. http://volcanoes.usgs.gov/activity/methods/hydrologic/lahardetection.php

[39] Freewave Technologies. [Online]. https://www.freewave.com

[40] Pablo Marcillo. (2006, Abril) Sistema Informático para Adquisición, Almacenamiento y Análisis de datos de deformaciones volcánicas. [Online]. http://repositorio.puce.edu.ec/handle/22000/1048

[41] Crossbow. (2007, Septiembre) Imote2 Buider Kit Manual.

[42] Carlos Macias. (2012, Agosto) Diseño de un sistema digital de monitoreo de lahares.

[43] Google Inc. [Online]. https://maps.google.com/

[44] Eclipse. [Online]. http://www.eclipse.org/

[45] Labcenter Electronics. [Online]. http://www.labcenter.com/index.cfm

[46] Libelium. Cooking Hacks. [Online]. http://www.cooking-hacks.com/

[47] Pctel. [Online]. http://www.antenna.com/

[48] Lantronix. [Online]. http://www.lantronix.com

[49] SourceForge. [Online]. http://jpcap.sourceforge.net/

[50] HYCONTROL. FMCW Radar Technology and Time Domain Reflectometry (TDR).

[51] Metageek. [Online]. http://www.metageek.net/

[52] Hart Communication Foundation. Hart Communication Foundation. [Online]. http://www.hartcomm.org/

[53] Intel Mote2 Community. [Online]. https://groups.yahoo.com/neo/groups/intel-mote2-community/info

[54] TinyOS Community Forum. [Online]. http://webs.cs.berkeley.edu/tos/contrib.html

[55] Mark Perillo and Wendi Heinzelman. Wireless Sensor Network Protocols.

[56] Jacobo Ocharan, "Sistemas de Alerta Temprana. Fotografía actual y retos.," *Cuadernos Internaciones de Tecnología para el Desarrollo Humano*, pp. 1-6, 2007.

[57] Steven Kosmerchock. Wireless Sensor Network Topologies.

[58] Minard L. Hall, "Chronology of the principal scientific and governmental actions leading up to the November 13, 1985 eruption of Nevado del Ruiz Colombia," *ELSEVIER*, pp. 101-115, 2010.

[59] Muhammad Omer Farooq and Thomas Kunz, "Operating Systems for Wireless Sensor Networks: A Survey," *Sensors - Open Access Journal*, pp. 5900-5930, 2011.

[60] Wireless Sensor Network Research Group. [Online]. http://www.sensor-networks.org/

[61] The United States Geological Survey (USGS). [Online]. http://volcanoes.usgs.gov/activity/methods/hydrologic/lahardetection.php

[62] The United States Geological Survey (USGS), Acoustic Flow Monitor System, 2005.

[63] Austin Zeideman and Laura Astrid Ramírez Elizalde, "Apocalipsis anunciado: un viraje a la política de riesgo de Colombia a partir de 1985," *Red de Revistas Científicas de América Latina y el Caribe, España y Portugal*, pp. 119-131, 2010.

[64] University of Illinois at Urbana Champaign - Open Systems Laboratory & Smart Structures Technology Laboratory. (2009, Agosto) ISM400 Sensor Board - Advanced User's Guide.

[65] Frank Durda. [Online]. http://www.freebsd.org/doc/en/articles/serial-uart/

[66] Byte Paradigm. [Online]. http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/?/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html

[67] Wireless Sensor Network - Research Group. [Online]. http://www.sensor-networks.org/?page=0823123150

[68] Pablo Pancardo García mmanuel A. González Vazconcelos. Vigilancia de Accesos Físicos empleando una Red Inalambrica de Sensores.

[69] Garmin. (2011, Octubre) GPS 18x Technical Specifications.

[70] Sercel. (2014, Febrero) Analog Seismic Sensors.

[71] Christian Cisneros, Omar Marcillo, and Wilson Enriquez, "Calibrador digital de sensores sísmicos," *XIX Jornadas en Ingeniería Eléctrica y Electrónica*, 2005.

[72] IEEE Computer Society, "Part 15.4: Low-Rate Wireless Personal," *IEEE-SA*

Standards Board, Junio 2011.

[73] HYCONTROL. VG6 & VG7 FMCW Radar Level Gauge.

[74] Klaus Betke. (2001, Agosto) The NMEA 0183 Protocol.

[75] Luis Reyes. [Online]. http://dev4an.blogspot.com/2013/09/introduccion-xbee.html

[76] Microchem. [Online]. http://www.microchem.fr/ayet/gps_maison/nmea_info.htm

[77] Digi International. [Online]. http://www.digi.com/

[78] Mathworks. [Online]. http://www.mathworks.com/products/matlab/

[79] Radar Tutorial. [Online]. http://www.radartutorial.eu/02.basics/Continuous%20Wave%20Radar.en.html

[80] Maxim. [Online]. http://pdfserv.maximintegrated.com/en/ds/MAX3222-MAX3241.pdf

[81] Sparkfun. [Online]. https://learn.sparkfun.com/tutorials/logic-levels/ttl-logic-levels

[82] FreeBSD. [Online]. http://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/

[83] Roger Coude. [Online]. http://www.cplus.org/rmw/english1.html

[84] uCLinux. [Online]. http://www.uclinux.org/

[85] Adi Mallikarjuna, Phani Kumar, and et al. (2007) Operating Systems for Wireless Sensor Networks: A Survey.

[86] Oracle Corporation. [Online]. http://www.oracle.com/technetwork/java/javase/documentation/index.html

[87] PostgreSQL. [Online]. http://www.postgresql.org/docs/

ANEXOS

Código fuente de programas para módulos

```
/*
 *
 * Base Station

 * Copyright(C) 2014 Pablo Marcillo Lara

 * This program is free software; you can redistribute it and/or modify it under the
 * terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later version.

 * This program is distributed in the hope that it will be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public License for more details.cd

 * You should have received a copy of the GNU General Public License along with
 * this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave,
 * Cambridge, MA 02139, USA.

 * Contact me at email:  pablomarcillolara@gmail.com
 *
 */

//#include "printf.h"

configuration BaseStation {

}
implementation {
        components MainC, BaseStationM;
        components new TimerMilliC() as Timer1, new TimerMilliC() as Timer2;
        components LedsC, CrcC;
        components ActiveMessageC;
        components STUARTC;
        components FFUARTC;
        components new AMSenderC(AM_TMPHMDMSG) as TMPHMDSenderC,
                        new AMReceiverC(AM_TMPHMDMSG) as TMPHMDReceiverC;
        components new AMSenderC(AM_HGHMSG) as HGHSenderC,
                        new AMReceiverC(AM_HGHMSG) as HGHReceiverC;
        components new AMSenderC(AM_SSMMSG) as SSMSenderC,
                        new AMReceiverC(AM_SSMMSG) as SSMReceiverC;
        components new AMSenderC(AM_CMRMSG) as CMRSenderC;
        //components SerialStartC;
        //components PrintfC;

        BaseStationM.Boot->MainC.Boot;
        BaseStationM.SplitControl->ActiveMessageC.SplitControl;
        BaseStationM.StdControlSTUART->STUARTC;
        BaseStationM.UartByteSTUART->STUARTC.UartByteSTUART;
        BaseStationM.UartStreamSTUART->STUARTC.UartStreamSTUART;
```

```
	BaseStationM.StdControlFFUART->FFUARTC;
	BaseStationM.UartByteFFUART->FFUARTC.UartByteFFUART;
	BaseStationM.UartStreamFFUART->FFUARTC.UartStreamFFUART;
	BaseStationM.GPS->Timer1;
	BaseStationM.Clock->Timer2;
	BaseStationM.Leds->LedsC;
	BaseStationM.Crc->CrcC;
	BaseStationM.AMPacket->ActiveMessageC;
	BaseStationM.Packet->ActiveMessageC;
	BaseStationM.TMPHMDSend->TMPHMDSenderC;
	BaseStationM.TMPHMDReceive->TMPHMDReceiverC;
	BaseStationM.HGHSend->HGHSenderC;
	BaseStationM.HGHReceive->HGHReceiverC;
	BaseStationM.SSMSend->SSMSenderC;
	BaseStationM.SSMReceive->SSMReceiverC;
	BaseStationM.CMRSend->CMRSenderC;
}


#include "Messages.h"
#include "message.h"
#include "AM.h"
#include "Serial.h"
#include "Packets.h"

module BaseStationM
{
	uses
	{
		interface Boot;
		interface SplitControl;
		interface Leds;
		interface Timer<TMilli> as GPS;
		interface Timer<TMilli> as Clock;
		interface UartStream as UartStreamSTUART;
		interface UartByte as UartByteSTUART;
		interface StdControl as StdControlSTUART;
		interface UartByte as UartByteFFUART;
		interface UartStream as UartStreamFFUART;
		interface StdControl as StdControlFFUART;
		interface Crc;
		interface AMSend as TMPHMDSend;
		interface AMSend as HGHSend;
		interface AMSend as SSMSend;
		interface AMSend as CMRSend;
		interface Receive as TMPHMDReceive;
		interface Receive as HGHReceive;
		interface Receive as SSMReceive;
		interface Receive as CMRReceive;
		interface AMPacket;
		interface Packet;
		interface BuildPacket;
	}
}

implementation
```

```
{
        TmpHmdMsg* tmpHmdMsg;
        SsmMsg* ssmMsg;
        HghMsg* hghMsg;
        CmrMsg* cmrMsg;

        message_t message;

        am_id_t type;
        am_group_t group;
        am_addr_t addr;
        am_addr_t source;
        am_addr_t dest;

        uint8_t PACKET_HEADER = 7;
        uint8_t SERIAL_STACK_HEADER = 4;
        uint8_t SERIAL_STACK_FOOTER = 3;
        uint8_t TOSMSG_HEADER = 6;
        uint8_t UNSCAPED_TOSMSG = 4;
        uint8_t DATA_BITS = 8;
        uint8_t STOP_BITS = 1;
        uint8_t HIGH_ALERT = 2;
        uint8_t LOW_ALERT = 1;
        uint8_t SECONDS_PER_MINUTE = 60;

        uint8_t GPS_START_CHAR = 36, GPS_END_CHAR = 10;
        uint8_t flag_GPGGA = 0, flag_PGRMF = 0, flag_GPRMC = 0;
        uint8_t hour, minute, second, day, month, year;
        uint8_t leap_year = 28;
        uint16_t currentYear;
        uint8_t lttd, ltgr, ltmn, lntt, lngr, lnmn;
        uint8_t stll, altt;
        uint8_t i, j, k;
        uint8_t cnt;
        uint8_t pos;
        uint8_t number;
        uint8_t index = 0;
        uint8_t snid = 0;
        uint8_t packet_type = 0;
        uint8_t alert_delay = 0;
        uint16_t runningCRC = 0;

        char *nmea, *strn, *pcb;
        char *gps_strn;
        char comma[2] = ",", point[2] = ".";
        char *tokens[20];
        char *ltta, *lgga;
        char latitude0[4], latitude1[4], longitude0[5], longitude1[4];
        char ltda[3], ltga[3], ltma[4], lnta[4], lnga[3], lnma[4];
        char daya[3], mtha[3], yara[3], hura[3], mnta[3], scna[3];
        char buffer[100], temp[100], ext[100];
        char chr;

        bool ready_GPS = FALSE;
        bool alert_time_expired = TRUE;
        bool FLOW_CNTL = FALSE;
```

```
	bool packet_send = FALSE;

	uint8_t GPS_ERR_CAST[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32,
67, 97, 115, 116, 32, 69, 114, 114, 111, 114, 33, 33, 33 };		//cast error
	uint8_t GPS_ERR_UKNW[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32,
85, 110, 107, 110, 111, 119, 110, 32, 71, 80, 83, 32, 83, 101, 110, 116, 101, 110, 99, 101, 33, 33, 33 };
//unknown gps sentence
	uint8_t GPS_ERR_DATA[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32,
68, 97, 116, 97, 32, 69, 114, 114, 111, 114, 33, 33, 33 }; //data error
	uint8_t GPS_ERR_SNTC[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32,
66, 65, 68, 95, 70, 82, 68, 95, 83, 78, 84, 78, 67, 69 }; //bad formed sentence
	uint8_t GPS_INFO_RDNG[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32,
82, 101, 97, 100, 105, 110, 103, 32, 71, 80, 83, 33, 33, 33 }; //reading gps
	uint8_t GPS_INFO_DONE[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32,
68, 111, 110, 101, 32, 71, 80, 83, 33, 33, 33 }; //reading gps done
	uint8_t GPS_INFO_TRNG[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32,
84, 117, 114, 110, 105, 110, 103, 32, 71, 80, 83, 32, 79, 110, 33, 33, 33 }; //turning gps on
	uint8_t GPS_INFO_SHTD[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32,
83, 104, 117, 116, 116, 105, 110, 103, 32, 71, 80, 83, 32, 68, 111, 119, 110, 33, 33, 33 }; //shutting gps down
	uint8_t GNRL_INFO[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48, 32};
//info
	uint8_t GNRL_RDNG_MSSG[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 58, 48, 48,
32, 82, 101, 97, 100, 105, 110, 103, 32, 77, 101, 115, 115, 97, 103, 101, 33, 33, 33 }; //reading message
	uint8_t GNRL_SNDG_TMPHMD_PCKT[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 48,
58, 48, 48, 32, 83, 101, 110, 100, 105, 110, 103, 32, 84, 77, 80, 72, 77, 68, 32, 80, 97, 99, 107, 101, 116, 33,
33, 33 }; //sending packet
	uint8_t GNRL_SNDG_HGH_PCKT[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 48, 58,
48, 48, 32, 83, 101, 110, 100, 105, 110, 103, 32, 72, 71, 72, 32, 80, 97, 99, 107, 101, 116, 33, 33, 33 };
//sending packet
	uint8_t GNRL_SNDG_SSM_PCKT[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 48, 58,
48, 48, 32, 83, 101, 110, 100, 105, 110, 103, 32, 83, 83, 77, 32, 80, 97, 99, 107, 101, 116, 33, 33, 33 };
//sending packet
	uint8_t GNRL_SNDG_ALRT[] = { 13, 10, 91, 73, 78, 70, 79, 93, 32, 48, 48, 58, 48, 48, 48, 58, 48, 48,
32, 83, 101, 110, 100, 105, 110, 103, 32, 65, 108, 101, 114, 116, 33, 33, 33 }; //sending alert

	task void sendAlert();

	task void sendAlert(){
		cmrMsg = (CmrMsg * ) call Packet.getPayload(&message,
			sizeof(CmrMsg));
		alert_delay = 0;
		cmrMsg->snid = snid;
		if(alert_time_expired)
			cmrMsg->alrt = LOW_ALERT;
		else
			cmrMsg->alrt = HIGH_ALERT;
		call CMRSend.send(AM_BROADCAST_ADDR, &message, sizeof(CmrMsg));
	}

	event void Boot.booted()
	{
		call UartStreamSTUART.disableReceiveInterrupt();
		call UartStreamFFUART.enableReceiveInterrupt();
		call StdControlSTUART.start();
		call StdControlFFUART.start();
		call SplitControl.start();
```

```
		atomic {
			GPS_INFO_TRNG[9] = hour / 10 + 48;
			GPS_INFO_TRNG[10] = (uint8_t)(hour % 10) + 48;
			GPS_INFO_TRNG[12] = minute / 10 + 48;
			GPS_INFO_TRNG[13] = (uint8_t)(minute % 10) + 48;
			GPS_INFO_TRNG[15] = second / 10 + 48;
			GPS_INFO_TRNG[16] = (uint8_t)(second % 10) + 48;
		}

		call UartStreamFFUART.send(GPS_INFO_TRNG, sizeof(GPS_INFO_TRNG));
		call Clock.startPeriodic(1000);
}

event void GPS.fired()
{
		call Leds.led0Toggle();

		atomic {
			GPS_INFO_RDNG[9] = hour / 10 + 48;
			GPS_INFO_RDNG[10] = (uint8_t)(hour % 10) + 48;
			GPS_INFO_RDNG[12] = minute / 10 + 48;
			GPS_INFO_RDNG[13] = (uint8_t)(minute % 10) + 48;
			GPS_INFO_RDNG[15] = second / 10 + 48;
			GPS_INFO_RDNG[16] = (uint8_t)(second % 10) + 48;
		}

		call UartStreamFFUART.send(GPS_INFO_RDNG, sizeof(GPS_INFO_RDNG));
		call UartStreamFFUART.enableReceiveInterrupt();
}

event void Clock.fired()
{
		call Leds.led0Toggle();

		atomic{

			if(alert_time_expired == FALSE){
				if(alert_delay >= SECONDS_PER_MINUTE){
					alert_delay = 0;
					alert_time_expired = TRUE;
					post sendAlert();
				}
				else
					alert_delay = alert_delay + 1;
			}

			second++;
			if(second >= 60) {
				second = 0;
				minute++;
				if(minute >= 60) {
					minute = 0;
					hour++;
					if(hour >= 24) {
						hour = 0;
```

```
                                                day++;
                                                if(month == 1 || month == 3 || month == 5 || month == 7
|| month == 8 || month == 10 || month == 12) {
                                                        if(day > 31){
                                                                day = 1;
                                                                month++;
                                                        }
                                                }
                                                else if(month == 4 || month == 6 || month == 9 || month
== 11) {
                                                        if(day > 30) {
                                                                day = 1;
                                                                month++;
                                                        }
                                                }
                                                else if(month == 2){
                                                        if(year % 4 == 0  && (year % 100 != 0  ||  year
% 400 == 0)) {
                                                                leap_year = 29;
                                                        }
                                                        else {
                                                                leap_year = 28;
                                                        }
                                                        if(day > leap_year) {
                                                                day = 1;
                                                                month++;
                                                        }
                                                }
                                                if(month > 12) {
                                                        month = 1;
                                                        year++;
                                                }
                                        }
                                }
                        }
                }
        }

        event message_t* TMPHMDReceive.receive(message_t* msg, void* payload, uint8_t len)
        {
                atomic {
                        if(ready_GPS) {
                                uint8_t PACKET_LENGTH = SERIAL_STACK_HEADER +
TOSMSG_HEADER + (uint8_t)TMPHMD_PCK_LENGTH + SERIAL_STACK_FOOTER;
                                uint8_t data[PACKET_LENGTH];
                                uint8_t aux[PACKET_LENGTH - UNSCAPED_TOSMSG];

                                call Leds.led1Toggle();

                                tmpHmdMsg = (TmpHmdMsg*)payload;

                                data[0] = HDLC_FLAG_BYTE;
                                data[1] = SERIAL_PROTO_PACKET_NOACK;
                                data[2] = 0;
                                data[3] = 255;
```

```
                                data[4] = 255;
                                data[5] = 0;
                                data[6] = TOS_AM_ADDRESS;
                                data[7] = TMPHMD_PCK_LENGTH;
                                data[8] = TOS_AM_GROUP;
                                data[9] = AM_TMPHMDMSG;

                                data[10] = tmpHmdMsg->alrt;
                                data[11] = snid = tmpHmdMsg->snid;
                                data[12] = (uint8_t)(tmpHmdMsg->tmpr >> 8);
                                data[13] = (uint8_t)(tmpHmdMsg->tmpr & 0xFF);
                                data[14] = (uint8_t)(tmpHmdMsg->hmdt >> 8);
                                data[15] = (uint8_t)(tmpHmdMsg->hmdt & 0xFF);

                                currentYear = year + 2000;
                                data[16] = (uint8_t)(currentYear >> 8);
                                data[17] = (uint8_t)(currentYear & 0xFF);
                                data[18] = month;
                                data[19] = day;
                                data[20] = hour;
                                data[21] = minute;
                                data[22] = second;
                                data[23] = stll;

                                for(k = 0; k < sizeof(aux); k++) {
                                        aux[k] = data[k + 1];
                                }

                                runningCRC = 0;
                                runningCRC = call Crc.crc16(aux, sizeof(aux));

                                data[24] = (uint8_t)(runningCRC & 0xFF);
                                data[25] = (uint8_t)(runningCRC >> 8);

                                data[26] = HDLC_FLAG_BYTE;
                                packet_type = AM_TMPHMDPCK;

                                if(tmpHmdMsg->alrt == HIGH_ALERT && alert_time_expired){
                                        alert_time_expired = FALSE;
                                        post sendAlert();
                                }
                                else{
                                        if(tmpHmdMsg->alrt == HIGH_ALERT)
                                                alert_delay = 0;
                                }

                                call UartStreamSTUART.send(data, sizeof(data));
                        }
                }
                return msg;
}

event message_t * HGHReceive.receive(message_t *msg, void *payload, uint8_t len)
{
        atomic {
                if(ready_GPS) {
```

```
                                        uint8_t PACKET_LENGTH = SERIAL_STACK_HEADER +
TOSMSG_HEADER + (uint8_t)HGH_PCK_LENGTH + SERIAL_STACK_FOOTER;
                                        uint8_t data[PACKET_LENGTH];
                                        uint8_t aux[PACKET_LENGTH - UNSCAPED_TOSMSG];

                                        call Leds.led1Toggle();

                                        hghMsg = (HghMsg*)payload;

                                        data[0] = HDLC_FLAG_BYTE;
                                        data[1] = SERIAL_PROTO_PACKET_NOACK;
                                        data[2] = 0;
                                        data[3] = 255;

                                        data[4] = 255;
                                        data[5] = 0;
                                        data[6] = TOS_AM_ADDRESS;
                                        data[7] = HGH_PCK_LENGTH;
                                        data[8] = TOS_AM_GROUP;
                                        data[9] = AM_HGHMSG;

                                        data[10] = hghMsg->alrt;
                                        data[11] = snid = hghMsg->snid;
                                        data[12] = (uint8_t)(hghMsg->height >> 8);
                                        data[13] = (uint8_t)(hghMsg->height & 0xFF);

                                        currentYear = year + 2000;
                                        data[14] = (uint8_t)(currentYear >> 8);
                                        data[15] = (uint8_t)(currentYear & 0xFF);
                                        data[16] = month;
                                        data[17] = day;
                                        data[18] = hour;
                                        data[19] = minute;
                                        data[20] = second;
                                        data[21] = stll;

                                        for(k = 0; k < sizeof(aux); k++) {
                                                aux[k] = data[k + 1];
                                        }

                                        runningCRC = 0;
                                        runningCRC = call Crc.crc16(aux, sizeof(aux));

                                        data[22] = (uint8_t)(runningCRC & 0xFF);
                                        data[23] = (uint8_t)(runningCRC >> 8);

                                        data[24] = HDLC_FLAG_BYTE;
                                        packet_type = AM_HGHPCK;

                                        if(hghMsg->alrt == HIGH_ALERT && alert_time_expired){
                                                alert_time_expired = FALSE;
                                                post sendAlert();
                                        }
                                        else{
                                                if(hghMsg->alrt == HIGH_ALERT)
                                                        alert_delay = 0;
```

```
                                        }

                                        call UartStreamSTUART.send(data, sizeof(data));
                                }
                        }
                        return msg;
                }

                event message_t * SSMReceive.receive(message_t *msg, void *payload, uint8_t len)
                {
                        atomic {
                                if(ready_GPS) {
                                        uint8_t PACKET_LENGTH = SERIAL_STACK_HEADER +
TOSMSG_HEADER + (uint8_t)SSM_PCK_LENGTH + SERIAL_STACK_FOOTER;
                                        uint8_t data[PACKET_LENGTH];
                                        uint8_t aux[PACKET_LENGTH - UNSCAPED_TOSMSG];

                                        call Leds.led1Toggle();

                                        ssmMsg = (SsmMsg*)payload;

                                        data[0] = HDLC_FLAG_BYTE;
                                        data[1] = SERIAL_PROTO_PACKET_NOACK;
                                        data[2] = 0;
                                        data[3] = 255;

                                        data[4] = 255;
                                        data[5] = 0;
                                        data[6] = TOS_AM_ADDRESS;
                                        data[7] = SSM_PCK_LENGTH;
                                        data[8] = TOS_AM_GROUP;
                                        data[9] = AM_SSMMSG;
                                        data[10] = ssmMsg->alrt;
                                        data[11] = snid = ssmMsg->snid;
                                        data[12] = (uint8_t)(ssmMsg->full >> 8);
                                        data[13] = (uint8_t)(ssmMsg->full & 0xFF);
                                        data[14] = (uint8_t)(ssmMsg->fthg >> 8);
                                        data[15] = (uint8_t)(ssmMsg->fthg & 0xFF);
                                        data[16] = (uint8_t)(ssmMsg->ftlw >> 8);
                                        data[17] = (uint8_t)(ssmMsg->ftlw & 0xFF);

                                        currentYear = year + 2000;
                                        data[18] = (uint8_t)(currentYear >> 8);
                                        data[19] = (uint8_t)(currentYear & 0xFF);
                                        data[20] = month;
                                        data[21] = day;
                                        data[22] = hour;
                                        data[23] = minute;
                                        data[24] = second;
                                        data[25] = stll;

                                        for(k = 0; k < sizeof(aux); k++) {
                                                aux[k] = data[k + 1];
                                        }

                                        runningCRC = 0;
```

```
                runningCRC = call Crc.crc16(aux, sizeof(aux));

                data[26] = (uint8_t)(runningCRC & 0xFF);
                data[27] = (uint8_t)(runningCRC >> 8);

                data[28] = HDLC_FLAG_BYTE;
                packet_type = AM_SSMPCK;

                if(ssmMsg->alrt == HIGH_ALERT && alert_time_expired){
                        alert_time_expired = FALSE;
                        post sendAlert();
                }
                else{
                        if(ssmMsg->alrt == HIGH_ALERT)
                                alert_delay = 0;
                }

                call UartStreamSTUART.send(data, sizeof(data));
        }
    }
    return msg;
}

event message_t * CMRReceive.receive(message_t *msg, void *payload, uint8_t len){
    return msg;
}

async event void UartStreamFFUART.receivedByte(uint8_t data) {
    chr = (char)data;
    switch(index) {
        case 0:
            if(data == GPS_START_CHAR) {
                gps_strn = "";
                index = 1;
                cnt = 0;
            }
            break;
        case 1:
            if(data == GPS_END_CHAR) {
                strncpy(ext, buffer, cnt);
                ext[cnt] = '\0';
                pcb = strstr(ext, ",,");
                pos = cnt + 1;
                while(pcb != NULL){

                    strncpy(temp, ext, pcb - ext + 1);
                    temp[pcb - ext + 1] = '\0';
                    strcat(temp, "0");

                    strcat(temp, pcb + 1);
                    temp[pos] = '\0';
                    pos = pos + 1;
                    strncpy(ext, temp, pos - 1);
                    ext[pos - 1] = '\0';

                    pcb = strstr(ext, ",,");
```

```
}

index = 0;
nmea = strn = strtok(ext, comma);
i = 0;

while(strn != NULL) {
        tokens[i] = strn;
        i++;
        strn = strtok(NULL, comma);
}

if(strcmp(nmea, "GPGGA") == 0) {
        flag_GPGGA = 0;
        if(strcmp(tokens[1], "0") != 0 && strlen(tokens[1]) >=
6) {
                strncpy(hura, tokens[1], 2);
                hura[2] = '\0';
                strncpy(mnta, tokens[1]+2, 2);
                mnta[2] = '\0';
                strncpy(scna, tokens[1]+4, 2);
                scna[2] = '\0';
                hour = atoi(hura);
                minute = atoi(mnta);
                second = atoi(scna);
        }
        if(strcmp(tokens[2], "0") != 0 && strlen(tokens[2]) >=
8) {
                ltta = strtok(tokens[2], point);
                strncpy(latitude0, ltta, 4);
                ltta = strtok(NULL, "");
                strncpy(latitude1, ltta, 4);

                strncpy(ltda, latitude0, 2);
                ltda[2] = '\0';
                strncpy(ltga, latitude0+2, 2);
                ltga[2] = '\0';
                strncpy(ltma, latitude1, 4);
                ltma[4] = '\0';
                lttd = atoi(ltda);
                ltgr = atoi(ltga);
                ltmn = atoi(ltma);
                ltmn = (uint8_t)(ltmn / 10000.0 * 60.0);
        }
        if(strcmp(tokens[4], "0") != 0 && strlen(tokens[4]) >=
10) {
                lgga = strtok(tokens[4], point);
                strncpy(longitude0, lgga, 5);
                lgga = strtok(NULL, "");
                strncpy(longitude1, lgga, 4);
                strncpy(lnta, longitude0, 3);
                lnta[3] = '\0';
                strncpy(lnga, longitude0+3, 2);
                lnga[2] = '\0';
                strncpy(lnma, longitude1, 4);
                lnma[4] = '\0';
```

```
                lntt = atoi(lnta);
                lngr = atoi(lnga);
                lnmn = atoi(lnma);
                lnmn = (uint8_t)(lnmn / 10000.0 * 60.0);
            }
            stll = atoi(tokens[7]);
            altt = atoi(tokens[9]);
            flag_GPGGA = 1;
            index = 0;
        }
        else if(strcmp(nmea, "GPRMC") == 0) {
            flag_GPRMC = 0;
            if(strcmp(tokens[9], "0") != 0 && strlen(tokens[9]) >=
6) {

                strncpy(daya, tokens[9], 2);
                daya[2] ='\0';
                strncpy(mtha, tokens[9]+2, 2);
                mtha[2] ='\0';
                strncpy(yara, tokens[9]+4, 2);
                yara[2] = '\0';
                day = atoi(daya);
                month = atoi(mtha);
                year = atoi(yara);
            }
            flag_GPRMC = 1;
            index = 0;
        }
        else if(strcmp(nmea, "PGRMF") == 0) {
            flag_PGRMF = 0;
            if(strcmp(tokens[3], "") != 0) {
                strncpy(yara, tokens[3], 2);
                yara[2]='\0';
                strncpy(mtha, tokens[3]+2, 2);
                mtha[2]='\0';
                strncpy(daya, tokens[3]+4, 2);
                daya[2]='\0';
                day = atoi(daya);
                month = atoi(mtha);
                year = atoi(yara);
            }
            flag_PGRMF = 1;
            index = 0;
        }
        else if((strcmp(nmea, "GPGSV") == 0) || (strcmp(nmea,
"GPGSA") == 0)) {

            index = 0;
            break;
        }
        else {
            GPS_ERR_UKNW[9] = hour / 10 + 48;
            GPS_ERR_UKNW[10] = (uint8_t)(hour % 10) + 48;
            GPS_ERR_UKNW[12] = minute / 10 + 48;
            GPS_ERR_UKNW[13] = (uint8_t)(minute % 10) + 48;
            GPS_ERR_UKNW[15] = second / 10 + 48;
            GPS_ERR_UKNW[16] = (uint8_t)(second % 10) + 48;
            call UartStreamFFUART.send(GPS_ERR_UKNW,
```

```
sizeof(GPS_ERR_UKNW));
                                                index = 0;
                                                break;
                                        }
                                        if((flag_GPGGA == 1) && ((flag_GPRMC == 1) ||
(flag_PGRMF == 1))) {
                                                if(stll >= 3) {
                                                        call
UartStreamFFUART.disableReceiveInterrupt();
                                                        if(ready_GPS == FALSE){
                                                                ready_GPS = TRUE;
                                                                call Clock.startPeriodic(1000); //1 s.
                                                                call GPS.startPeriodic(120000); //2
min.
                                                        }
                                                        GPS_INFO_SHTD[9] = hour / 10 + 48;
                                                        GPS_INFO_SHTD[10] = (uint8_t)(hour % 10)
+ 48;
                                                        GPS_INFO_SHTD[12] = minute / 10 + 48;
                                                        GPS_INFO_SHTD[13] = (uint8_t)(minute %
10) + 48;
                                                        GPS_INFO_SHTD[15] = second / 10 + 48;
                                                        GPS_INFO_SHTD[16] = (uint8_t)(second %
10) + 48;
                                                        call
UartStreamFFUART.send(GPS_INFO_SHTD, sizeof(GPS_INFO_SHTD));
                                                }
                                                index = 0;
                                                flag_GPGGA = 0;
                                                flag_GPRMC = 0;
                                                flag_PGRMF = 0;
                                        }
                                }
                                else {
                                        buffer[cnt] = chr;
                                        cnt++;
                                        if(cnt >= sizeof(buffer)) {
                                                GPS_ERR_SNTC[9] = hour / 10 + 48;
                                                GPS_ERR_SNTC[10] = (uint8_t)(hour % 10) + 48;
                                                GPS_ERR_SNTC[12] = minute / 10 + 48;
                                                GPS_ERR_SNTC[13] = (uint8_t)(minute % 10) + 48;
                                                GPS_ERR_SNTC[15] = second / 10 + 48;
                                                GPS_ERR_SNTC[16] = (uint8_t)(second % 10) + 48;
                                                call UartStreamFFUART.send(GPS_ERR_SNTC,
sizeof(GPS_ERR_SNTC));
                                                index = 0;
                                        }
                                }
                                break;
                        }

                return;
        }

        async event void UartStreamSTUART.sendDone( uint8_t* buf, uint16_t len, error_t error ) {
                call Leds.led2Toggle();
```

```
                if(packet_type == AM_TMPHMDPCK) {
                        GNRL_SNDG_TMPHMD_PCKT[9] = hour / 10 + 48;
                        GNRL_SNDG_TMPHMD_PCKT[10] = (uint8_t)(hour % 10) + 48;
                        GNRL_SNDG_TMPHMD_PCKT[12] = minute / 10 + 48;
                        GNRL_SNDG_TMPHMD_PCKT[13] = (uint8_t)(minute % 10) + 48;
                        GNRL_SNDG_TMPHMD_PCKT[15] = second / 10 + 48;
                        GNRL_SNDG_TMPHMD_PCKT[16] = (uint8_t)(second % 10) + 48;
                        call UartStreamFFUART.send(GNRL_SNDG_TMPHMD_PCKT,
sizeof(GNRL_SNDG_TMPHMD_PCKT));
                }
                else if(packet_type == AM_HGHPCK) {
                        GNRL_SNDG_HGH_PCKT[9] = hour / 10 + 48;
                        GNRL_SNDG_HGH_PCKT[10] = (uint8_t)(hour % 10) + 48;
                        GNRL_SNDG_HGH_PCKT[12] = minute / 10 + 48;
                        GNRL_SNDG_HGH_PCKT[13] = (uint8_t)(minute % 10) + 48;
                        GNRL_SNDG_HGH_PCKT[15] = second / 10 + 48;
                        GNRL_SNDG_HGH_PCKT[16] = (uint8_t)(second % 10) + 48;
                        call UartStreamFFUART.send(GNRL_SNDG_HGH_PCKT,
sizeof(GNRL_SNDG_HGH_PCKT));
                }
                else if(packet_type == AM_SSMPCK) {
                        GNRL_SNDG_SSM_PCKT[9] = hour / 10 + 48;
                        GNRL_SNDG_SSM_PCKT[10] = (uint8_t)(hour % 10) + 48;
                        GNRL_SNDG_SSM_PCKT[12] = minute / 10 + 48;
                        GNRL_SNDG_SSM_PCKT[13] = (uint8_t)(minute % 10) + 48;
                        GNRL_SNDG_SSM_PCKT[15] = second / 10 + 48;
                        GNRL_SNDG_SSM_PCKT[16] = (uint8_t)(second % 10) + 48;
                        call UartStreamFFUART.send(GNRL_SNDG_SSM_PCKT,
sizeof(GNRL_SNDG_SSM_PCKT));
                }
                packet_type = 0;
        return;
        }

        async event void UartStreamSTUART.receivedByte(uint8_t data) {
        return;
        }

        async event void UartStreamSTUART.receiveDone( uint8_t* buf, uint16_t len, error_t error ) {
        return;
        }

        async event void UartStreamFFUART.sendDone( uint8_t* buf, uint16_t len, error_t error ) {
                return;
        }

        async event void UartStreamFFUART.receiveDone( uint8_t* buf, uint16_t len, error_t error ) {
        return;
        }

        event void SplitControl.stopDone(error_t error){
                // TODO Auto-generated method stub
        }

        event void SplitControl.startDone(error_t error){
                // TODO Auto-generated method stub
```

```
	}

	event void HGHSend.sendDone(message_t *msg, error_t error){
		// TODO Auto-generated method stub
	}

	event void TMPHMDSend.sendDone(message_t *msg, error_t error){
		// TODO Auto-generated method stub
	}

	event void SSMSend.sendDone(message_t *msg, error_t error){
		// TODO Auto-generated method stub
	}

	event void CMRSend.sendDone(message_t *msg, error_t error){
		call Leds.led2Toggle();
	}
}


/*
 *
 * Seismic Station

 * Copyright(C) 2014 Pablo Marcillo Lara

 * This program is free software; you can redistribute it and/or modify it under the
 * terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later version.

 * This program is distributed in the hope that it will be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public License for more details.

 * You should have received a copy of the GNU General Public License along with
 * this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave,
 * Cambridge, MA 02139, USA.

 * Contact me at email:  pablomarcillolara@gmail.com
 *
 */

//#include "printf.h"

configuration SeismicStation {

}

implementation {
	components MainC, SeismicStationM, LedsC;
	components new TimerMilliC() as Timer1, new TimerMilliC() as Timer2;
	components new MAX136xC() as Sensor;
	components new AMSenderC(AM_SSMMSG);
	components ActiveMessageC;
	//components PrintfC;
	//components SerialStartC;
```

```
        SeismicStationM.Boot->MainC.Boot;
        SeismicStationM.Leds->LedsC;
        SeismicStationM.Sampling->Timer1;
        SeismicStationM.PacketSend->Timer2;
        SeismicStationM.ADC->Sensor;
        SeismicStationM.AMSend->AMSenderC;
        SeismicStationM.SensorControl->Sensor.SplitControl;
        SeismicStationM.SplitControl->ActiveMessageC.SplitControl;
        SeismicStationM.HalMAX136xAdvanced->Sensor;
        SeismicStationM.AMPacket->ActiveMessageC;
        SeismicStationM.Packet->ActiveMessageC;
}


#include "Messages.h"
#include "MAX136x.h"

module SeismicStationM {
        uses {
                interface Boot;
                interface Leds;
                interface Timer<TMilli> as Sampling;
                interface Timer<TMilli> as PacketSend;
                interface AMSend;
                interface AMPacket;
                interface Packet;
                interface Read<max136x_data_t> as ADC;
                interface HalMAX136xAdvanced;
                interface SplitControl as SensorControl;
                interface SplitControl;
        }
}

implementation {
        message_t message;
        uint8_t ID_SRIS = 3;
        uint8_t HIGH_ALERT = 2;
        uint8_t LOW_ALERT = 1;
        uint16_t SAMPLE_PERIOD = 2; //20 = 50 Hz, 10 = 100Hz, 5 = 200Hz, 2 = 500Hz, 1 = 1000Hz
        uint8_t BUFFER_LENGTH = 21;
        uint16_t CONVERSION_FACTOR = 10000;
        uint16_t NORMAL_DELAY = 10000; //10s
        uint16_t ALERT_DELAY = 1000; //1s
        uint16_t THRESHOLD = 200;
        uint8_t COEFFICIENTS_NUMBER = 21;
        int16_t buffer[21];
        uint8_t index;
        uint8_t counter;
        uint8_t number;
        uint16_t OFFSET = 2047;
        int16_t value = 0;
        int32_t low = 0, high = 0, full = 0;
        uint32_t low_average = 0, high_average = 0, full_average = 0;
        int8_t i = 0;
        int8_t j = 0;
```

```
	int8_t k = 0;
	uint16_t samples = 0;
	bool buffer_full;
	bool flag;
	int16_t sn;
	double ax = 0.0f;
	int16_t count = 0;

	//	int16_t COEFFICIENTS_LOW[51] = {3, -4, -14, -19, -13, -1, -1, -32, -83, -109,
	//			-75, -13, -11, -127, -296, -355, -204, 35, 76, -260, -777, -948, -321, 1028,
	//			2418, 3008, 2418, 1028, -321, -948, -777, -260, 76, 35, -204, -355, -296,
	//			-127, -11, -13, -75, -109, -83, -32, -1, -1, -13, -19, -14, -4, 3};
	//	int16_t COEFFICIENTS_HIGH[51] = {-6, 0, 8, 10, 1, -17, -24, -1, 39, 51, 2, -77,
	//			-99, -3, 143, 179, 5, -258, -321, -6, 492, 646, 6, -1352, -2748, 6660, -
2748,
	//			-1352, 6, 646, 492, -6, -321, -258, 5, 179, 143, -3, -99, -77, 2, 51, 39, -1,
	//			-24, -17, 1, 10, 8, 0, -6};
	//	int16_t COEFFICIENTS_FULL[51] = {-3, -4, -6, -9, -13, -18, -25, -34, -45, -58,
	//			-73, -90, -110, -131, -153, -176, -200, -223, -245, -266, -285, -301, -314,
	//			-324, -330, 9668, -330, -324, -314, -301, -285, -266, -245, -223, -200, -
176,
	//			-153, -131, -110, -90, -73, -58, -45, -34, -25, -18, -13, -9, -6, -4, -3};

	int16_t COEFFICIENTS_HIGH[21] = {21, 1, -55, -104, -3, 291, 468, 5, -1251,
			-2696, 6660, -2696, -1251, 5, 468, 291, -3, -104, -55, 1, 21};

	int16_t COEFFICIENTS_LOW[21] = {-42, -28, 7, 25, -118, -460, -685, -268, 951,
			2372, 3008, 2372, 951, -268, -685, -460, -118, 25, 7, -28, -42};

	int16_t COEFFICIENTS_FULL[21] = {-21, -28, -48, -79, -121, -169, -218, -263,
			-300, -324, 9668, -324, -300, -263, -218, -169, -121, -79, -48, -28, -21};

	//	int16_t COEFFICIENTS_HIGH[15] = {-31, -1, 137, 300, 4, -1134, -2633, 6660,
	//			-2633, -1134, 4, 300, 137, -1, -31};
	//	int16_t COEFFICIENTS_LOW[15] = {7, -37, -216, -440, -213, 862, 2316, 3008,
	//			2316, 862, -213, -440, -216, -37, 7};
	//	int16_t COEFFICIENTS_FULL[15] = {-24, -38, -79, -140, -209, -272, -316, 9668,
	//			-316, -272, -209, -140, -79, -38, -24};
	//	int16_t COEFFICIENTS_HIGH[11] = {43, 115, 3, -936, -2516, 6660, -2516, -936, 3,
	//			115, 43};
	//	int16_t COEFFICIENTS_LOW[11] = {-68, -169, -132, 711, 2214, 3008, 2214, 711,
	//			-132, -169, -68};
	//	int16_t COEFFICIENTS_FULL[11] = {-25, -54, -129, -224, -302, 9668, -302, -224,
	//			-129, -54, -25};

	event void Boot.booted() {
		call SensorControl.start();
		call SplitControl.start();
		call Sampling.startPeriodic(SAMPLE_PERIOD);
		call PacketSend.startPeriodic(NORMAL_DELAY);
	}

	event void Sampling.fired() {
		call ADC.read();
		call Leds.led0Toggle();
	}
```

```
	event void AMSend.sendDone(message_t * msg, error_t error) {
		call Leds.led2Toggle();
	}

	event void ADC.readDone(error_t result, max136x_data_t val) {
		value = (val & 0xFFF) - OFFSET;
		//value = 1000 * (sin(6.28*50*ax) + sin(6.28*150*ax)); //50 + 150 Hz
		if(index >= BUFFER_LENGTH) {
			buffer_full = TRUE;
			index = 0;
		}

		buffer[index] = value;

		if(buffer_full) {
			low = 0;
			high = 0;
			full = 0;
			k = 0;

			for(i = index; i >= 0; i--) {
				if(k >= COEFFICIENTS_NUMBER)
					break;
				full = full + buffer[i]*COEFFICIENTS_FULL[k];
				high = high + buffer[i]*COEFFICIENTS_HIGH[k];
				low = low + buffer[i]*COEFFICIENTS_LOW[k];
				//printf("k: %i - i: %i --- buffer: %i - coeff: %i - full: %i ---- number:
%i\n", k, i, buffer[i], COEFFICIENTS_FULL[k], full, COEFFICIENTS_NUMBER);
				//printf("%i ",i);
				k++;
			}

			for(j = BUFFER_LENGTH - 1; j > index; j--) {
				if(k >= COEFFICIENTS_NUMBER)
					break;
				full = full + buffer[j]*COEFFICIENTS_FULL[k];
				high = high + buffer[j]*COEFFICIENTS_HIGH[k];
				low = low + buffer[j]*COEFFICIENTS_LOW[k];
				//printf("k: %i - j-1: %i --- buffer: %i - coeff: %i - full: %i ---- number:
%i\n", k, j - 1, buffer[j - 1], COEFFICIENTS_FULL[k], full, number);
				//printf("%i ",j);
				k++;
			}

			full_average = full_average + abs(full) / CONVERSION_FACTOR;
			high_average = high_average + abs(high) / CONVERSION_FACTOR;
			low_average = low_average + abs(low) / CONVERSION_FACTOR;

			samples = samples + 1;
		}
		index = index + 1;
		ax = ax + 0.002;
		//call Leds.led1Toggle();
	}
```

```
event void PacketSend.fired() {
	SsmMsg * ssmMsg;

	ssmMsg = (SsmMsg * ) call Packet.getPayload(&message, sizeof(SsmMsg));

	ssmMsg->full = full_average / samples;
	ssmMsg->fthg = high_average / samples;
	ssmMsg->ftlw = low_average / samples;

	ssmMsg->snid = ID_SRIS;

	if( ! flag) {
		if(ssmMsg->ftlw >= THRESHOLD) {
			ssmMsg->alrt = HIGH_ALERT;
			call PacketSend.startPeriodic(ALERT_DELAY);
			flag = TRUE;
		}
		else
			ssmMsg->alrt = LOW_ALERT;
	}
	else {
		if(counter >= NORMAL_DELAY / ALERT_DELAY) {
			counter = 0;
			flag = FALSE;
			call PacketSend.startPeriodic(NORMAL_DELAY);
		}
		else {
			if(ssmMsg->ftlw >= THRESHOLD)
				counter = 0;
			else
				counter++;
		}
		ssmMsg->alrt = HIGH_ALERT;
	}

	samples = 0;
	low_average = 0;
	high_average = 0;
	full_average = 0;

	call AMSend.send(AM_BROADCAST_ADDR, &message, sizeof(SsmMsg));
}

event void HalMAX136xAdvanced.alertThreshold() {
	// TODO Auto-generated method stub
}

event void HalMAX136xAdvanced.enableAlertDone(error_t error) {
	// TODO Auto-generated method stub
}

event void HalMAX136xAdvanced.setClockDone(error_t error) {
	// TODO Auto-generated method stub
}

event void HalMAX136xAdvanced.setRefDone(error_t error) {
```

```
		// TODO Auto-generated method stub
	}

	event void HalMAX136xAdvanced.setMonitorModeDone(error_t error) {
		// TODO Auto-generated method stub
	}

	event void HalMAX136xAdvanced.setConversionModeDone(error_t error) {
		// TODO Auto-generated method stub
	}

	event void HalMAX136xAdvanced.setScanModeDone(error_t error) {
		// TODO Auto-generated method stub
	}

	event void SensorControl.stopDone(error_t error) {
		// TODO Auto-generated method stub
	}

	event void SensorControl.startDone(error_t error) {
		call HalMAX136xAdvanced.getStatus();
	}

	event void SplitControl.startDone(error_t error) {
		// TODO Auto-generated method stub
	}

	event void SplitControl.stopDone(error_t error) {
		// TODO Auto-generated method stub
	}

	event void HalMAX136xAdvanced.getStatusDone(error_t error, uint8_t status,
			max136x_data_t data) {
		// TODO Auto-generated method stub
	}
}


/*
 *
 * Sensor Station

 * Copyright(C) 2014 Pablo Marcillo Lara

 * This program is free software; you can redistribute it and/or modify it under the
 * terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later version.

 * This program is distributed in the hope that it will be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public License for more details.

 * You should have received a copy of the GNU General Public License along with
 * this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave,
 * Cambridge, MA 02139, USA.
```

```
 * Contact me at email:  pablomarcillolara@gmail.com
 *
 */

//#include "printf.h"

configuration SensorStation {

}
implementation {
	components MainC, SensorStationM, LedsC, new TimerMilliC() as Timer1,
				new TimerMilliC() as Timer2, new TimerMilliC() as Timer3;
	components new MAX136xC() as Sensor;
	components new AMSenderC(AM_TMPHMDMSG) as AMSenderC1;
	components new AMSenderC(AM_HGHMSG) as AMSenderC2;
	components ActiveMessageC;
	components new SensirionSht11C() as Sensirion;
	//components SerialStartC;
	//components PrintfC;

	SensorStationM.Boot->MainC.Boot;
	SensorStationM.Leds->LedsC;
	SensorStationM.TmpHmdSampling->Timer1;
	SensorStationM.HghSampling->Timer2;
	SensorStationM.Mismatch->Timer3;
	SensorStationM.ADC->Sensor;
	SensorStationM.SensorControl->Sensor.SplitControl;
	SensorStationM.SplitControl->ActiveMessageC.SplitControl;
	SensorStationM.HalMAX136xAdvanced->Sensor;
	SensorStationM.AMSendTmpHmd->AMSenderC1;
	SensorStationM.AMSendHgh->AMSenderC2;
	SensorStationM.AMPacket->ActiveMessageC;
	SensorStationM.Packet->ActiveMessageC;
	SensorStationM.Temperature->Sensirion.Temperature;
	SensorStationM.Humidity->Sensirion.Humidity;
	SensorStationM.HalSht11Advanced->Sensirion;
	SensorStationM.SensorControl->Sensirion.SplitControl;
}

#include "Messages.h"

module SensorStationM {
	uses {
		interface Boot;
		interface Leds;
		interface Timer<TMilli> as TmpHmdSampling;
		interface Timer<TMilli> as HghSampling;
		interface Timer<TMilli> as Mismatch;
		interface HalMAX136xAdvanced;
		interface SplitControl as SensorControl;
		interface SplitControl;
		interface Read<max136x_data_t> as ADC;
		interface AMSend as AMSendTmpHmd;
		interface AMSend as AMSendHgh;
		interface AMPacket;
		interface Packet;
```

```
		interface Read<uint16_t> as Temperature;
		interface Read<uint16_t> as Humidity;
		interface HalSht11Advanced;
	}
}

implementation {
	message_t message;
	uint8_t ID_SRIS = 3;
	uint8_t HIGH_ALERT = 2;
	uint8_t LOW_ALERT = 1;
	uint16_t THRESHOLD_TEMP = 7960; //40 celsius grades
	uint16_t THRESHOLD_HUM = 1510; //50%
	uint16_t THRESHOLD_HEIGHT = 2572; //10 meters
	uint16_t NORMAL_DELAY = 20000; //20s
	uint16_t GAP = 10000;
	uint16_t ALERT_DELAY = 5000; //5s
	bool ready_tmp = FALSE;
	bool ready_hmd = FALSE;
	bool flag_tmp_hmd;
	bool flag_hgh;
	uint16_t aux_tmpr;
	uint16_t aux_hmdt;
	uint16_t aux_height;
	uint8_t counter_tmp_hmd;
	uint8_t counter_hgh;

	event void Boot.booted() {
		call SensorControl.start();
		call SplitControl.start();
	}

	event void ADC.readDone(error_t result, max136x_data_t val) {
		HghMsg * hghMsg = (HghMsg * ) call Packet.getPayload(&message, sizeof(HghMsg));

		aux_height = val & 0xFFF;

		hghMsg->height = aux_height;
		hghMsg->snid = ID_SRIS;

		if( ! flag_hgh) {
			if(aux_height >= THRESHOLD_HEIGHT) {
				hghMsg->alrt = HIGH_ALERT;
				call HghSampling.startPeriodic(ALERT_DELAY);
				flag_hgh = TRUE;
			}
			else
				hghMsg->alrt = LOW_ALERT;
		}
		else {
			if(counter_hgh >= NORMAL_DELAY / ALERT_DELAY) {
				counter_hgh = 0;
				call HghSampling.startPeriodic(NORMAL_DELAY);
				flag_hgh = FALSE;
			}
			else {
```

```
                        if(aux_height >= THRESHOLD_HEIGHT)
                                counter_hgh = 0;
                        else
                                counter_hgh++;
                }
                hghMsg->alrt = HIGH_ALERT;
        }

        call AMSendHgh.send(AM_BROADCAST_ADDR, &message, sizeof(HghMsg));
        call Leds.led1Toggle();
}

event void Temperature.readDone(error_t result, uint16_t val) {
        aux_tmpr = val;
        //printf("tmpr:%i\n",aux_tmpr);
        call Humidity.read();
        call Leds.led1Toggle();
}

event void Humidity.readDone(error_t result, uint16_t val) {
        TmpHmdMsg * tmpHmdMsg = (TmpHmdMsg * ) call Packet.getPayload(&message,
                        sizeof(TmpHmdMsg));

        aux_hmdt = val;
        //printf("hmdt:%i\n",aux_hmdt);
        tmpHmdMsg->tmpr = aux_tmpr;
        tmpHmdMsg->hmdt = aux_hmdt;
        tmpHmdMsg->snid = ID_SRIS;

        if( ! flag_tmp_hmd) {
                if(aux_tmpr >= THRESHOLD_TEMP || aux_hmdt >= THRESHOLD_HUM) {
                        tmpHmdMsg->alrt = HIGH_ALERT;
                        call TmpHmdSampling.startPeriodic(ALERT_DELAY);
                        flag_tmp_hmd = TRUE;
                }
                else
                        tmpHmdMsg->alrt = LOW_ALERT;
        }
        else {
                if(counter_tmp_hmd >= NORMAL_DELAY / ALERT_DELAY) {
                        counter_tmp_hmd = 0;
                        call TmpHmdSampling.startPeriodic(NORMAL_DELAY);
                        flag_tmp_hmd = FALSE;
                }
                else {
                        if(aux_tmpr >= THRESHOLD_TEMP || aux_hmdt >=
THRESHOLD_HUM)
                                counter_tmp_hmd = 0;
                        else
                                counter_tmp_hmd++;
                }
                tmpHmdMsg->alrt = HIGH_ALERT;
        }

        call AMSendTmpHmd.send(AM_BROADCAST_ADDR, &message,
sizeof(TmpHmdMsg));
```

```
        call Leds.led1Toggle();
}

event void SensorControl.startDone(error_t result) {
        call HalMAX136xAdvanced.getStatus();
        call TmpHmdSampling.startPeriodic(NORMAL_DELAY);
        call Mismatch.startPeriodic(GAP);
}

event void HalMAX136xAdvanced.alertThreshold() {
}

event void SplitControl.startDone(error_t result) {
}

event void SplitControl.stopDone(error_t result) {
}

event void SensorControl.stopDone(error_t error) {
}

event void HalMAX136xAdvanced.setScanModeDone(error_t error) {
}

event void HalMAX136xAdvanced.setMonitorModeDone(error_t error) {
}

event void HalMAX136xAdvanced.setConversionModeDone(error_t error) {
}

event void HalMAX136xAdvanced.setClockDone(error_t error) {
}

event void HalMAX136xAdvanced.setRefDone(error_t error) {
}

event void HalMAX136xAdvanced.getStatusDone(error_t error, uint8_t status,
                max136x_data_t data) {
}

event void HalMAX136xAdvanced.enableAlertDone(error_t error) {
}

event void HalSht11Advanced.getVoltageStatusDone(error_t error, bool isLow) {
}

event void HalSht11Advanced.setHeaterDone(error_t error) {
}

event void HalSht11Advanced.setResolutionDone(error_t error) {
}

event void AMSendTmpHmd.sendDone(message_t * msg, error_t error) {
        call Leds.led2Toggle();
}
```

```
	event void AMSendHgh.sendDone(message_t * msg, error_t error) {
		call Leds.led2Toggle();
	}

	event void TmpHmdSampling.fired() {
		call Temperature.read();
		call Leds.led0Toggle();
	}

	event void HghSampling.fired() {
		call ADC.read();
		call Leds.led0Toggle();
	}

	event void Mismatch.fired() {
		call HghSampling.startPeriodic(NORMAL_DELAY);
		call Mismatch.stop();
	}
}


/*
 *
 * Visualization Station

 * Copyright(C) 2014 Pablo Marcillo Lara

 * This program is free software; you can redistribute it and/or modify it under the
 * terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later version.

 * This program is distributed in the hope that it will be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public License for more details.cd

 * You should have received a copy of the GNU General Public License along with
 * this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave,
 * Cambridge, MA 02139, USA.

 * Contact me at email:  pablomarcillolara@gmail.com
 *
 */

#include "hardware.h"

configuration VisualizationStation {
}
implementation {
	components MainC, VisualizationStationM, new TimerMilliC();
	components ActiveMessageC, HplPXA27xGPIOC, LedsC;
	components new AMReceiverC(AM_CMRMSG) as CMRReceiverC;

	VisualizationStationM.Boot->MainC.Boot;
	VisualizationStationM.Leds->LedsC;
	VisualizationStationM.Timer->TimerMilliC.Timer;
	VisualizationStationM.SplitControl->ActiveMessageC.SplitControl;
```

```
	VisualizationStationM.CMRReceive->CMRReceiverC;
	VisualizationStationM.HplPXA27xGPIOPin-
>HplPXA27xGPIOC.HplPXA27xGPIOPin[STUART_TXD];
}

#include "Messages.h"

module VisualizationStationM {
	uses {
		interface Boot;
		interface Receive as CMRReceive;
		interface SplitControl;
		interface HplPXA27xGPIOPin;
		interface Timer<TMilli>;
		interface Leds;
	}
}
implementation {
	CmrMsg * cmrMsg;
	uint8_t HIGH_ALERT = 2;
	uint8_t LOW_ALERT = 1;
	uint16_t counter = 0;

	event void Boot.booted() {
		call SplitControl.start();
		call Timer.startPeriodic(10000);
		call HplPXA27xGPIOPin.setGPDRbit(TRUE);
	}

	event void SplitControl.startDone(error_t error) {
		// TODO Auto-generated method stub
	}

	event void SplitControl.stopDone(error_t error) {
		// TODO Auto-generated method stub
	}

	event message_t * CMRReceive.receive(message_t * msg, void * payload,
			uint8_t len) {
		call Leds.led1Toggle();
		cmrMsg = (CmrMsg * ) payload;
		if(cmrMsg->alrt == LOW_ALERT)
			call HplPXA27xGPIOPin.setGPCRbit();
		else
			call HplPXA27xGPIOPin.setGPSRbit();
		return msg;
	}

	async event void HplPXA27xGPIOPin.interruptGPIOPin(){
		// TODO Auto-generated method stub
	}

	event void Timer.fired(){
		if(counter%2 == 0)
			call HplPXA27xGPIOPin.setGPCRbit();
		else
```

```
                call HplPXA27xGPIOPin.setGPSRbit();
            counter++;
        }
    }
```

Código fuente de programas para cámara de video

```
/*
 *
 * activate_relay

 * Copyright(C) 2014 Pablo Marcillo Lara

 * This program is free software; you can redistribute it and/or modify it under the
 * terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later version.

 * This program is distributed in the hope that it will be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public License for more details.cd

 * You should have received a copy of the GNU General Public License along with
 * this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave,
 * Cambridge, MA 02139, USA.

 * Contact me at email:  pablomarcillolara@gmail.com
 *
 */

#!/bin/sh

flag_low=true
flag_high=true

mbus /dev/io w 4,0
mbus /dev/io w 1,0

while true; do
  if [ `mbus /dev/io w 2 r 1` = "01" ]; then
    if [ "$flag_low" = true ] ; then
      mbus /dev/io w 4,0
      #echo flag_low
    fi
    flag_low=false
    flag_high=true
  else
    if [ "$flag_high" = true ] ; then
      mbus /dev/io w 4,1
      #echo flag_high
      trigger-ftp-relay
    fi
    flag_high=false
    flag_low=true
  fin
  sleep 1
done
```

```
/*
 *
 * trigger_ftp_relay

 * Copyright(C) 2014 Pablo Marcillo Lara

 * This program is free software; you can redistribute it and/or modify it under the
 * terms of the GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option) any later version.

 * This program is distributed in the hope that it will be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public License for more details.cd

 * You should have received a copy of the GNU General Public License along with
 * this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave,
 * Cambridge, MA 02139, USA.

 * Contact me at email:  pablomarcillolara@gmail.com
 *
 */

#!/bin/sh

i=1
period_of_alert=30
delay=2

while [ $i -lt $period_of_alert ]
do
   /bin/ftpscript /etc/config/trigger-ftp.scr
   i=`expr $i + 2`
   sleep $delay
done
```

Normas de construcción de módulos

Módulo Base

		Pines	Color Cable
Alimentación		A -> VCC (+) B -> GND (-)	
GPS		A -> VCC (+) B -> GND (-) D -> Rx C -> Tx	
Datos		A -> Rx B -> GND (-) C -> Tx	
Depuración		A -> Rx B -> GND (-) C -> Tx	

Módulo de Sensores

		Pines	Color Cable
Alimentación		A -> VCC (+) B -> GND (-)	
Sensor		A -> VCC (24V) B -> GND (-)	

Módulo Sísmico

		Pines	Color Cable
Alimentación		A -> VCC (+) B -> GND (-)	
Sensor		A -> ADC B -> REF	

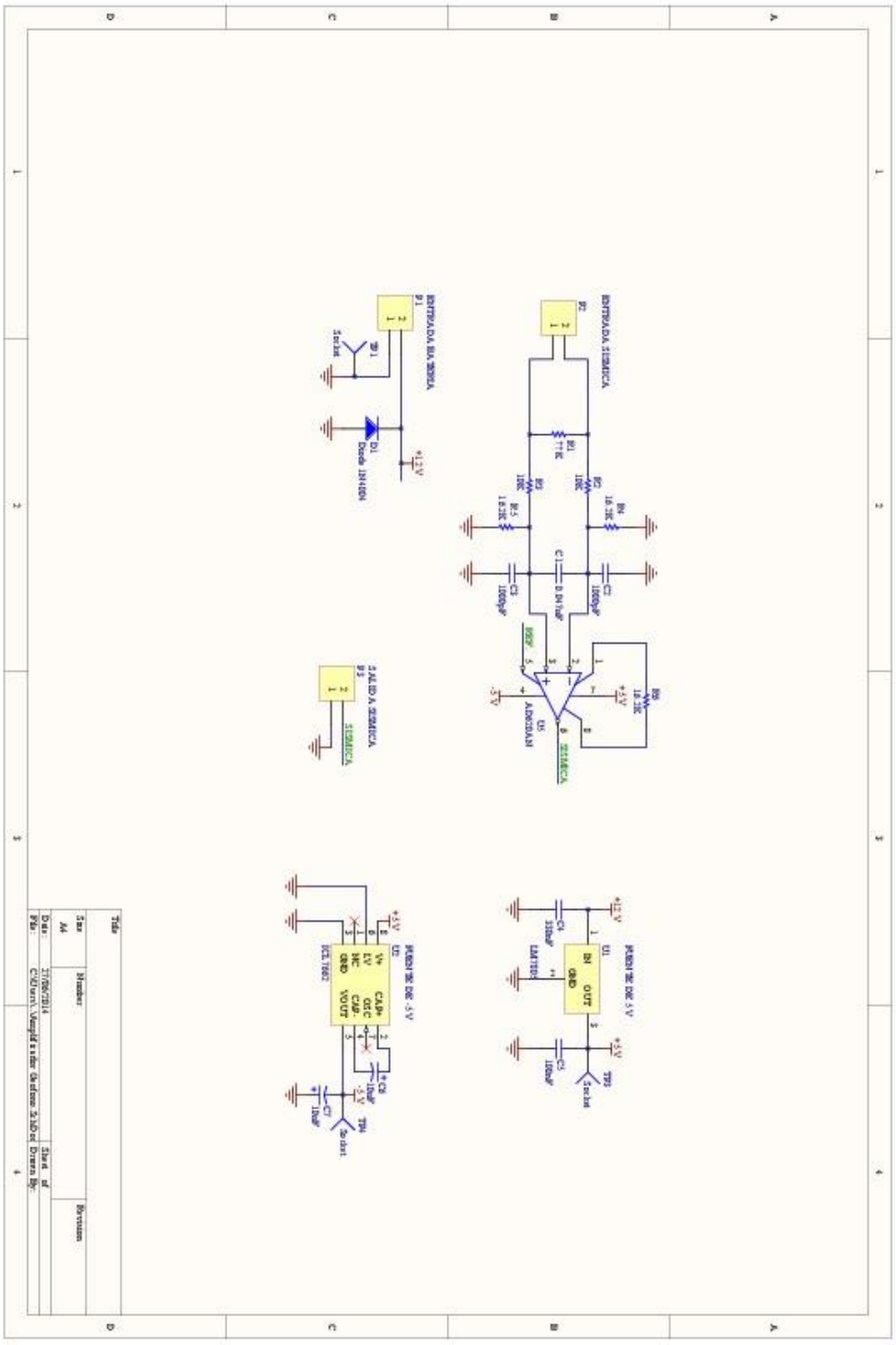
Módulo de Visualización

		Pines	Color Cable
Alimentación		A -> VCC (+) B -> GND (-)	
Cámara		B -> GND (-) C -> Tx	

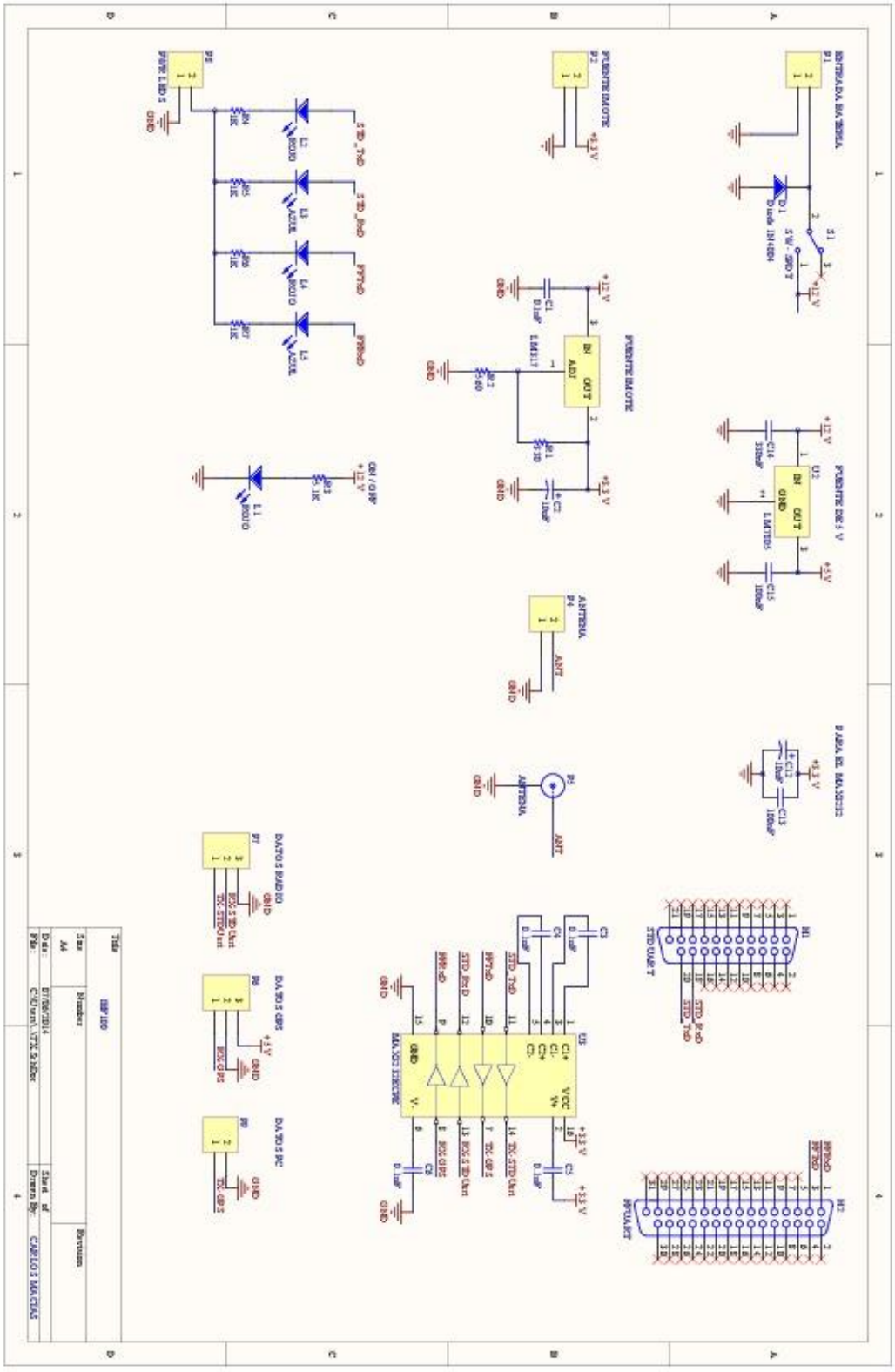
*** Los conectores especificados en este documento son tipo militar hembra**

Diagramas Esquemáticos

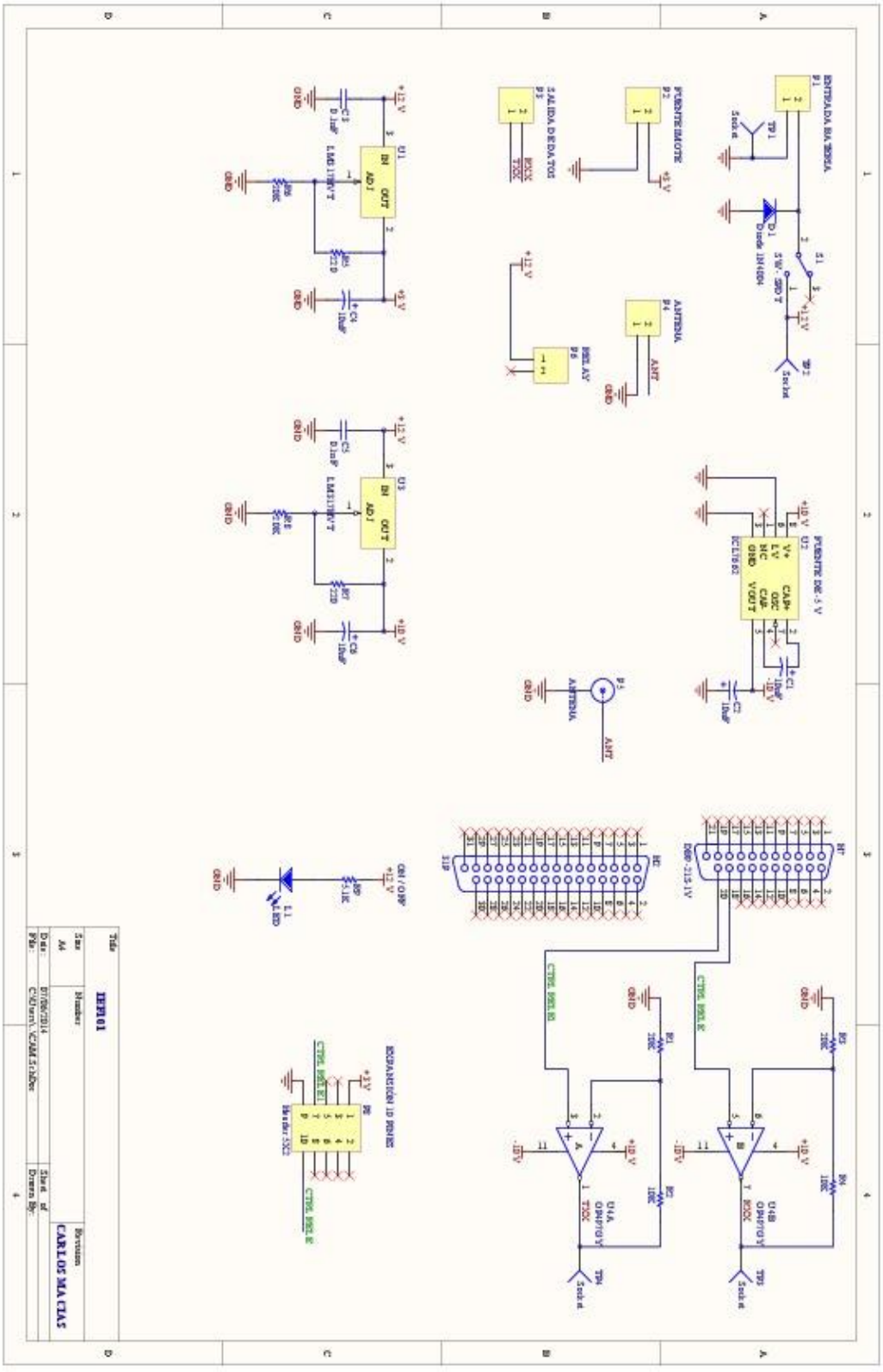
Amplificador



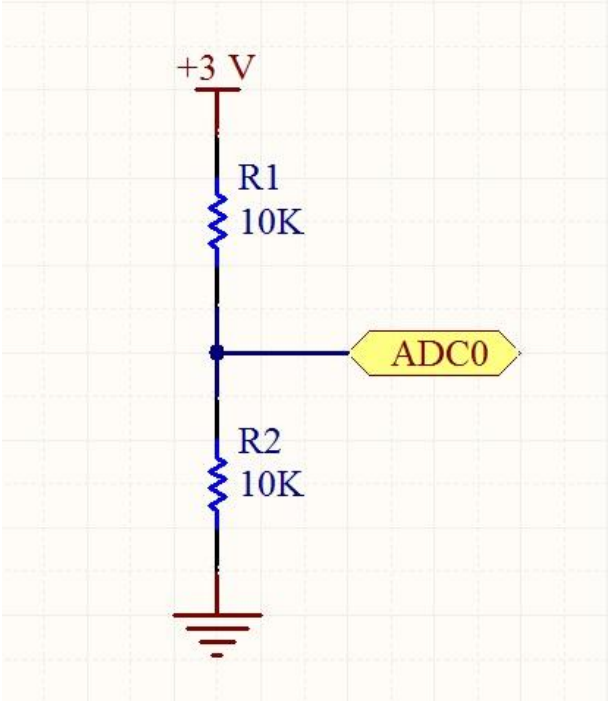
Tarjeta de interfaces IEF100



Tarjeta IEF101



Configuración de resistencias para el Módulo Sísmico



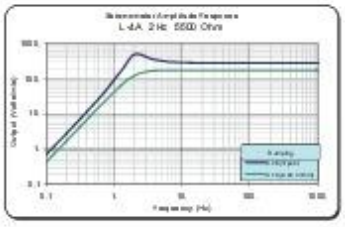
Fichas Técnicas

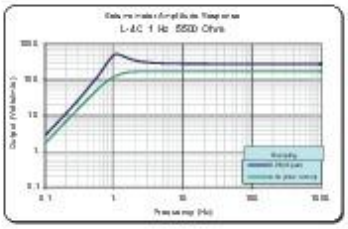
Sismómetros Sercel

SEISMOMETERS

	L-4A			L-4C		
Natural Frequency	2 Hz (± 0.25 Hz)			1 Hz (± 0.05 Hz)		
Tilt Tolerance	< 0.1 Hz Change to 10°			<0.05 Hz Change to 5°		
Open Circuit Damping	0,28					
Damping with Resistor	0,7					
Pk-Pk Coil Travel	6,25 mm					
Moving mass	1000 g			500 g		
Diameter	76 mm					
Length	130 mm					
Weight	2150 g			1700 g		
Operating Temperature	-29° to 60°C					
Coil Resistance	500 Ω	2000 Ω	5500 Ω	500 Ω	2000 Ω	5500 Ω
Sensitivity (V/m/s)	83,5	166,9	276,8	83,5	166,9	276,8
Shunt Resistor Value	820	3279	9016	820	3279	9016
Damping Constant ($R_s B_g f_n$) (ΩHz)	554	2217	6097	1109	4434	12194

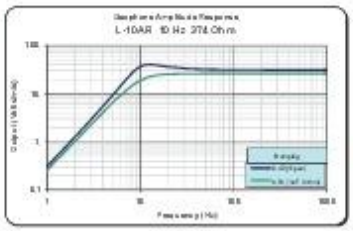
Parameters are specified at 20°C and no tilt unless otherwise noted.
All dimensions are for basic unit only (no land or marsh case)

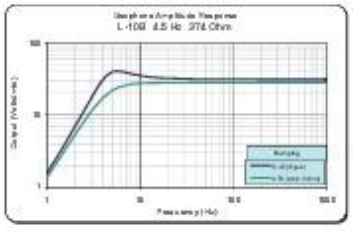




	L-10AR		L-10B	
Natural Frequency (± 0,5Hz)	10 Hz	14 Hz	4,5 Hz	
Coil Resistance (± 6,5%)	374 Ω		374 Ω	3600 Ω
Sensitivity (± 10%)	31,2 V/m/s		31,2 V/m/s	96,9 V/m/s
Harmonic Distortion (1)	0,20%			
Open Circuit Damping (± 10%)	0,42	0,31	0,42	
Damping Constant (RTBGfn)	6356,2 ΩHz		4561,5 ΩHz	43907,8 ΩHz
Moving Mass	12,2 g		17 g	
Pk-Pk Coil Travel	2 mm		2 mm	
Spurious Resonance	> 250 Hz		na	
Diameter	31,75 mm		31,75 mm	
Length	35,56 mm		35,56 mm	
Weight	141,75 g		141,75 g	

Parameters are specified at 20°C and no tilt unless otherwise noted.
All dimensions are for basic unit only (no land or marsh case)
(1) measured at 17.8 mm/sec pk-pk velocity and the higher of either 12 Hz or the nominal natural frequency





Plataforma imote2

MEMSIC

Imote2

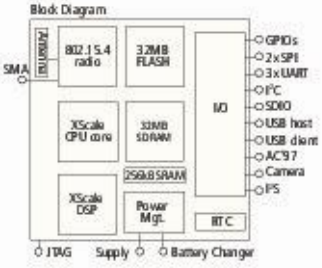
HIGH-PERFORMANCE WIRELESS SENSOR NETWORK NODE

- Marvell PXA271 XScale® Processor at 13 – 416MHz
- Marvell Wireless MMX DSP Coprocessor
- 256kB SRAM, 32MB FLASH, 32MB SDRAM
- Integrated 802.15.4 Radio
- Integrated 2.4GHz Antenna, Optional External SMA Connector
- Multi-color Status Indicator LED
- USB Client With On-board mini-B Connector and Host Adapters
- Rich Set of Standard I/O: 3xUART, 2xSPI, I²C, SDIO, GPIOs
- Application Specific I/O: I²S, AC97, Camera Chip Interface, JTAG
- Compact Size: 36mm x 48mm x 9mm

Applications

- Digital Image Processing
- Condition Based Maintenance
- Industrial Monitoring and Analysis
- Seismic and Vibration Monitoring

Block Diagram



Imote2

The Imote2 is an advanced wireless sensor node platform. It is built around the low-power PXA271 XScale CPU and also integrates an 802.15.4 compliant radio. The design is modular and stackable with interface connectors for expansion boards on both the top and bottom sides. The top connectors provide a standard set of I/O signals for basic expansion boards. The bottom connectors provide additional high-speed interfaces for application specific I/O. A battery board supplying system power can be connected to either side.

Processor

The Imote2 contains the Marvell PXA271 CPU. This processor can operate in a low voltage (0.85V), low frequency (13MHz) mode, hence enabling very low power operation. The frequency can be scaled from 13MHz to 416MHz with Dynamic Voltage Scaling. The processor has a number of different low power modes such as sleep and deep sleep. The PXA271 is a multi-chip module that includes three chips in a single package, the CPU with 256kB SRAM, 32MB SDRAM and 32MB of FLASH memory. It integrates many I/O options making it extremely flexible in supporting different sensors, A/Ds, radios, etc. These I/O features include I²C, 2 Synchronous Serial Ports (SPI) one of which is dedicated to the radio, 3 high speed UARTs, GPIOs, SDIO, USB client and host, AC97 and I²S audio codec interfaces, a fast infrared port, PWM, a Camera Interface and a high speed bus (Mobile Scaleable Link).

The processor also supports numerous timers as well as a real time clock. The PXA271 includes a wireless MMX coprocessor to accelerate multimedia operations. It adds 30 new media processor (DSP) instructions, support for alignment and video operations and compatibility with Intel MMX and SSE integer instructions. For more information on the PXA271, please refer to the Marvell datasheet.

Radio & Antenna

The Imote2 uses the CC2420 IEEE 802.15.4 radio transceiver from Texas Instruments. The CC2420 supports a 250kb/s data rate with 16 channels in the 2.4GHz band.

The Imote2 platform integrates a 2.4GHz surface mount antenna which provides a nominal range of about 30 meters. For longer range a SMA connector can be soldered directly to the board to connect to an external antenna.

Power Supply

The Imote2 can be powered by various means:

Primary Battery: This is typically accomplished by attaching a MEMSIC Imote2 Battery Board to either the basic or advanced connectors.

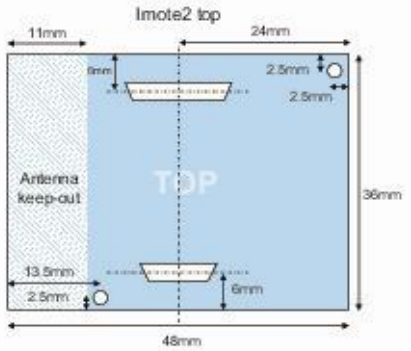
Phone: 408.964.9700 | Fax: 408.324.4841 | E-mail: infoca@memsic.com | www.memsic.com

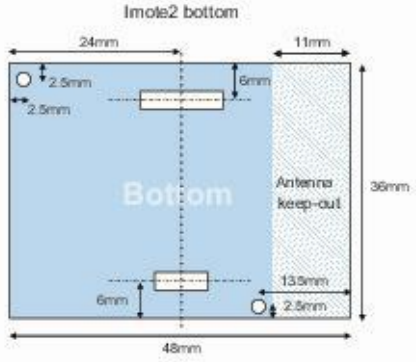
Rechargeable Battery: This requires a specially configured battery board attached to either the basic or advanced connectors. The Imote2 has a built-in charger for Li-Ion or Li-Poly batteries.

USB: The Imote2 can be powered via the on-board mini-B USB connector. This mode can also be used to charge an attached battery.

Battery Pads: A suitable primary battery or other power source can be connected via a dedicated set of solder pads on the Imote2 board.

Processor/Radio Board	IPR2400	Remarks
CPU		
Processor	Marvell PXA271	
SRAM Memory	256 kB	
SDRAM Memory	32MB	
FLASH Memory	32MB	
POWER CONSUMPTION		
Current Draw In Deep Sleep Mode	390 µA	
Current Draw In Active Mode	31 mA	13MHz, radio off
Current Draw In Active Mode	44 mA	13MHz, radio Tx/Rx
Current Draw In Active Mode	66mA	104 MHz, radio Tx/Rx
Radio		
Transceiver	TI CC2420	
Frequency Band (ISM)	2400.0 – 2483.5 MHz	
Data Rate	250 kb/s	
Tx Power	-24 – 0 dBm	
Rx Sensitivity	-94 dBm	
Range (line of sight)	~30 m	With integrated antenna
I/O		
USB Client (mini-B), USB Host		
UART 3x, GPIOs, I^2C, SDIO, SPI 2x, I^2S, AC97, Camera		
Power		
Battery Board	3x AAA	
USB Voltage	5.0 V	
Battery Voltage	3.2 – 4.5 V	
Li-Ion Battery Charger		
Mechanical		
Dimensions Imote2 Board	36mm x 48mm x 9mm	
Weight	12g	





Imote2 design licensed from Intel® Corporation.

Imote2
Battery Board

The universal battery board is designed to power the Imote2 using 3 primary AAA cells. Alternatively, rechargeable cells such as AAA NiMH can be used if charged separately.

The battery board can accommodate a plugged-in Imote2 via either the basic connectors (top) or the advanced connectors (bottom).

A mechanical switch on the side provides manual power shut-off. The battery board is fused for 500mA maximum current.

Battery Board	IBB2400CA
Batteries	3x AAA
Maximum Current	500mA Fused
Size	52mm x 43mm x 18mm
Weight with 3 AAA Batteries	51g
Weight without Batteries	14g

Imote2 Third Party Software

Several operating systems are available for Imote2 including TinyOS, Linux and SOS. Additional system software is available from Open Source.

For the latest operating systems and additional third party accessories please visit www.memsic.com.

Ordering Information

Model	Description
IPR2400CA	Imote2 Board plus the Battery Board (included)
IBB2400CA	Imote2 Battery Board (stand-alone)
IIB2400CA	Imote2 Interface Board

Phone: 408.964.9700 | Fax: 408.324.4841 | E-mail: infoca@memsic.com | www.memsic.com

Reflex VG7

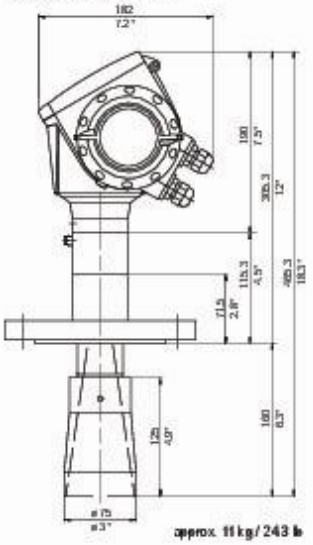
REFLEX VG SERIES HIGH FREQUENCY FMCWRADAR

Technical Data Sheet - Type VG7

Input		
Function		K-band FMCW radar
Parameter		Level, distance, volume and reflectivity
Min. tank height		0.5 m / 1.5 ft
Max. measuring range		40 m / 131 ft
Blocking Distance (dead zone)		Antenna extension length + antenna length + 0.1 m / 4"
Output		
Output signal	Output 1	4 ... 20 mA HART® or 3.8 ... 20.5 mA acc. to NAMUR NE 43
Accuracy		0.05% (ref. 20 mA; 20°C / 68°F)
Resolution		±2 µA
Temperature drift		Typically 50 ppm/K
Error signal		High: 22 mA; Low: 3.6 mA acc. to NAMUR NE 43
Max. Load		350 ohm
Measuring accuracy		
Reference conditions	Temperature	+20°C ± 5°C / +68°F ±9°F
acc. to IEC770	Pressure	1013 mbar abs. ±20 mbar / 14.69 psig ± 0.29 psig
	Relative air humidity	60% ± 15%
Resolution		1 mm / 0.04"
Accuracy		±3 mm / ±0.12"
Beam angle	DN 40 / ANSI 1 1/2"	20°
	DN 50 / ANSI 2"	15°
	DN 80 / ANSI 3"	10°
Application conditions		
Temperature	Ambient temperature	-40...+80°C / -40...+175°F; EEx i: -40...+60°C / -40...+140°F
	Storage temperature	-40...+85°C / -40...+185°F
	Flange temperature	-40...+150°C / -40...+300°F (Ex: refer to relevant device's approval and temperature class)
Thermal shock resistance		100°C/min
Process conditions	Operating pressure	-1...40 bar / -14.5...580 psig; subject to process connection used and flange temperature
	Dielectric constant	≥ 1.5
Vibration resistance		IEC 68-2-6 and prEN 50178 (10...57Hz: 0.075 mm / 57... 150 Hz: 1 g)
Protection category		IP 66/67 equiv. to NEMA 6-6X
Mechanical data		
Material	Housing	Aluminium
	Wetted parts	Stainless steel (1.4404 / 316L); Hastelloy C-22 (2.4602)
	Process fitting	Stainless steel (1.4404 / 316L); Hastelloy C-22 (2.4602)
	Gaskets	Viton (-40...+150°C / -40...+300°F); Kalrez 6375 (-20...+150°C / -5 ...+300°F)
Process connection	Thread	G 1 1/2"; NPT 1 1/2"
	Flange	DN 40... DN 150 (PN 40 / PN 16); 1 1/2"...8" (150 lb / 300 lb); 10 K (40...100A)
Electrical connection		
2-wire power supply	Terminals output 1	
	Non-Ex/ EEx i	24 V DC (14 ... 30 V DC)
	EEx d	24 V DC (20 ... 36 V DC)
Cable entry		M20x 1.5; NPT 1/2"; G 1/2"
Terminals		0.5...1.5 mm²
Human machine interface		
Display		9 lines, 160x160 pixels in 8-step greyscale with 4-button keypad
Operating languages		English (UK), German, French, Italian, Spanish, Portugese, Japanese, Chinese (Mandarin), Russian
Approvals	Overfill protection	WHG
	ATEX	ATEX II G/D 1, 1/2, 2 EEx ia IIC T6; ATEX II G/D 1/2, 2 EEx d ia IIC T6
	FM	IS class I Div. 1 Gr. A...G; XP class I Div. 1 Gr. A...G
	CSA	IS class I Div. 1 Gr. A...G; XP class I Div. 1 Gr. A...G

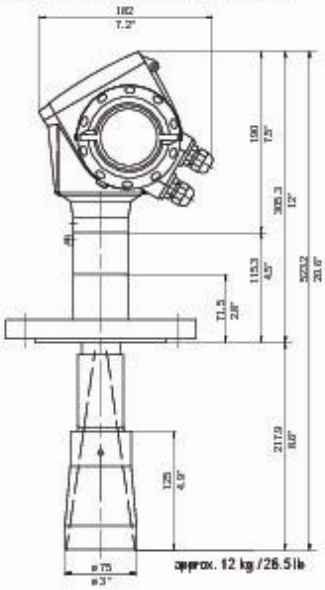
Dimensions and weights

Flange (Antenna DN80)

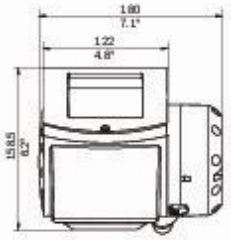


approx. 11 kg / 24.3 lb

Antenna DN80 with antenna extension



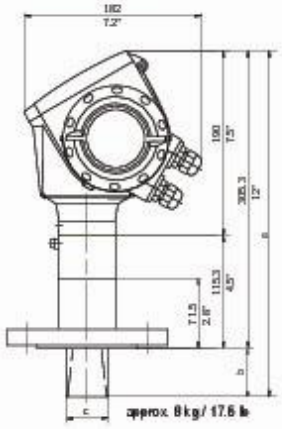
approx. 12 kg / 26.5 lb



Note:
Cable glands are not delivered with the device.

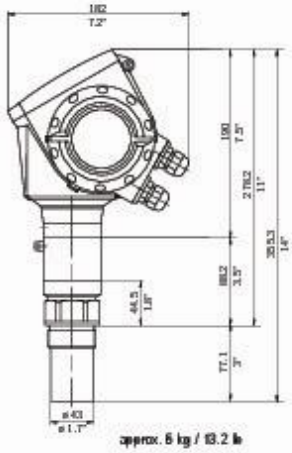
Note:
Additional antenna extensions of 105 mm / 4.1" length are available.

Flange (Antenna DN40/50)



approx. 8 kg / 17.6 lb

Thread

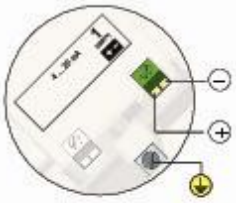


approx. 6 kg / 13.2 lb

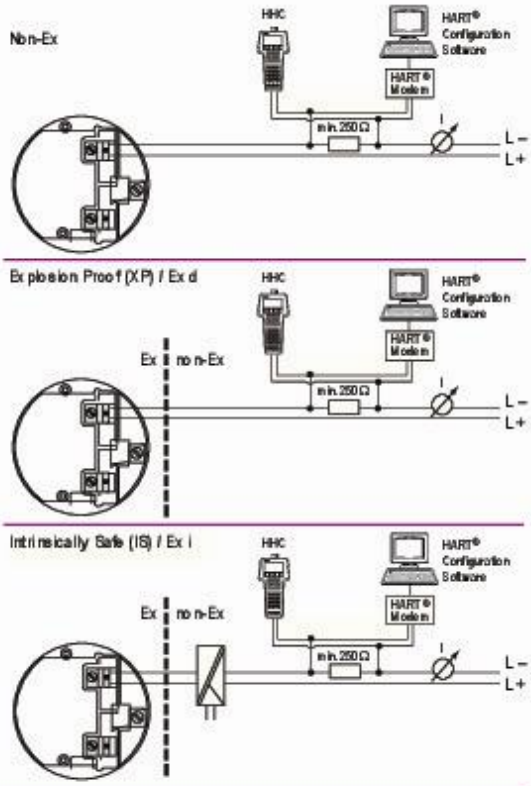
Antenna type	c mm / inch	b mm / inch	a mm / inch
Antenna DN 40	39 / 1.5	38.5 / 1.5	343.8 / 13.5
Antenna DN 50	43 / 1.7	50 / 2	355.3 / 14

Dimensions in mm and inch

Electrical connection and wiring



Output 1
4 ... 20 mA/HART
or
3.8 ... 20.5 mA/HART
acc. to NAMUR NE 43



Note: Other options how to connect the HHC (Hand Held Communicator) and modems to the HART® loop are available.

State-of-the-art with PACTware

The VG7 is PACTware-ready. Each device is supplied ex-factory with the appropriate DTM.

A DTM (Device Type Manager) is a device driver making available the device functionality independent from the FIELDBUS protocol and providing a graphical user interface optimized for device operation and configuration.

Simple on-screen and intuitive setup procedure for devices without a display, or for setup from the Central Control Room. Summarized setup provides perfect control of initial input, and guarantees perfect results.

All features of PACTware are fully supported:

- Online device setup
- Displays measured values
- Records measured information during operation
- Shows status of device
- Gives stepwise setup with on-screen progress check
- Displays summary of setup selection for final supervision