UNIVERSIDAD DE BELGRANO

Las tesinas de Belgrano

Facultad de Ingeniería
Carrera de Ingeniería Electrónica

Desarrollo de Firmware aplicado a Sistemas de Domótica

N° 513

Martín G. Cervantes

Tutor: Eduardo Martínez

Departamento de Investigaciones
2012

Universidad de Belgrano
Zabala 1837 (C1426DQ6)
Ciudad Autónoma de Buenos Aires - Argentina
Tel.: 011-4788-5400 int. 2533
e-mail: invest@ub.edu.ar
url: http://www.ub.edu.ar/investigaciones

Introducción

Motivado por la ausencia de una solución completa y de calidad de domótica de Código Abierto (Open-Source) decidí llevar a cabo este proyecto conformado tanto por una parte de firmware/software y otra de hardware. Ésta tesina se enfocará en la primera parte.

El término domótica proviene de la unión de las palabras domus (que significa casa en latín) y robótica (robota, esclavo en checo). Se entiende por domótica al conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas. Se podría definir como la integración de la tecnología en el diseño inteligente de un recinto. A grandes rasgos, un "hogar digital" consistiría en una vivienda donde se han introducido automatizaciones para mejorar el confort, la seguridad, el ocio, las comunicaciones, reducir el consumo energético, en definitiva mejorar la calidad de vida.

Los tres componentes fundamentales de una vivienda domotizada o inteligente son: sensores, controladores y actuadores. La secuencia básica de funcionamiento sería la siguiente: los sensores captan la información, datos de las magnitudes del hogar que son enviadas a los controladores, los cuales en función de los valores de esas magnitudes deciden si es necesaria o no una actuación; de ser necesario se envía la orden a los actuadores que activan el funcionamiento del dispositivo correspondiente de la vivienda.

Planteamiento y estado del Arte

Si bien la domótica y la idea de proveer a un hogar/vivienda de inteligencia no es algo nuevo o innovador, la mayoría de los sistemas y empresas que existen en la actualidad intentan imponer sus soluciones y/o servicios propietarios. Muchas veces esto está ligado a elevados precios lo que ha llevado a que la domótica no se haya implementado o sea tan cotidiana como uno hubiese esperado que sea hace algunos años atrás. Dada esta situación mi intención es llevar a cabo el proyecto para brindar una solución de calidad que se adapte a las tecnologías ya existentes, que sea de implementación sencilla, económica y de código libre.

Algunos ejemplos de sistemas de código abierto que existen en la actualidad:

- OpenRemote[1] es una plataforma de automatización y domótica basada en un conjunto de software/hardware multiplataforma.
- OpenDomo[2] es una plataforma de domótica que corre sobre un servidor central basado en una distribución de Linux modificada a tal fin.
- The Virtual Crib[3] es un software de automatización libre que corre sobre plataforma Windows.
- OpenSourceAutomation[4] es otra plataforma de domótica que corre sobre Windows.

Objetivo

Todos estos proyectos comparten muchos de nuestros objetivos:

- Domótica a bajo costo.
- Código abierto.
- Multiplataforma

A mi entender el sistema no debería estar atado a un hardware y en tal caso menos a un Sistema Operativo. OpenRemote es un buen ejemplo de sistema multiplataforma pero sólo es capaz de brindar una interface para controlar el sistema de domótica dejando la implementación del hardware y comunicaciones a cargo del usuario.

Se evidencia en todos ellos la falta de diseño de firmware que podrían agregar al sistema los siguientes beneficios:

- Firmware embebido en hardware standalone[5].
- Sistema descentralizado, cada nodo es auto-suficiente.
- Comunicaciones alámbricas como inalámbricas.

Otro de los objetivos de la tesina es sentar las bases para que el proyecto pueda crecer y adaptarse a nuevas necesidades.

Delimitaciones

Esta tesina se delimitará a llevar acabo la implementación del sistema en dos plataformas muy dife-

1. OpenRemote detalles en http://www.openremote.org/
2. OpenDomo detalles en http://opendomo.org
3. The Virtual Crib detalles en http://www.vcrib.com/
4. OSA detalles en http://www.opensourceautomation.com/
5. Hardware + firmware que pueden funcionar por si solos.

rentes para demostrar su escalabilidad.

1. Implementación en plataforma Unix con comunicaciones TCP/IP (IPv4).
2. Implementación en CC2530 con comunicaciones 802.15.4

Ambas implementaciones son funcionales y cuentan con los componentes de hardware + firmware mínimos para poder cumplir con las funcionalidades básicas planteadas en el capítulo de diseño del sistema. Así como también se mencionó en los objetivos el proyecto busca ser multiplataforma a tal fin se desarrolló el mismo en distintas capas capaces de abstraerse del hardware sobre el cual estén corriendo.

Organización de la tesina

En las siguientes secciones se presentarán los detalles del diseño del sistema, su implementación y los resultados obtenidos con el prototipo. En muchas ocasiones se hará referencia a distintos métodos o funciones de programación utilizadas en la implementación de ser necesario pueden obtenerse más detalles de los mismos utilizando el Anexo A que hace referencia a la documentación técnica de la tesina. Muchos de ellos pueden ser reconocidos porque han sido resaltados en letra cursiva.

En otras ocasiones se utilizarán palabras de nivel técnico o inglés asociadas al tema aquí expuesto para facilitar la lectura, y de ser necesario, las aclaraciones han sido agregadas en notas de pie con sus respectivos índices.

Se podrá observar que los diagramas y flujos están confeccionados en su mayoría en inglés, esto se debe a que el código ha sido comentado también en inglés para su fácil difusión y mayor adaptación/recepción en caso de ser necesario compartirlo con la comunidad open-source.

Metodología del trabajo

Para llevar a cabo el proyecto que en sí mismo es bastante amplio fue necesario establecer ciertas pautas que debieron cumplirse por etapas para luego llegar al resultado final.

Etapas del proyecto y metodología utilizada:

1. Relevamiento de las necesidades y funcionalidades básicas del sistema, sentar bases para que el sistema pueda crecer en futuras versiones. Investigación sobre el hardware adecuado en el cual debería correr el sistema.
2. Diseño multicapa del firmware, diseño del hardware que dará soporte.
3. Implementación de cada uno de los componentes del firmware y su puesta a prueba individualmente.
4. Implementación de cada uno de los componentes del hardware y su puesta a prueba individualmente.
5. Primer prototipo funcional.

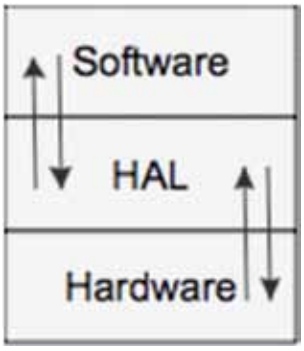
Para más detalles ver la sección Pruebas y resultados.

Diseño del Sistema

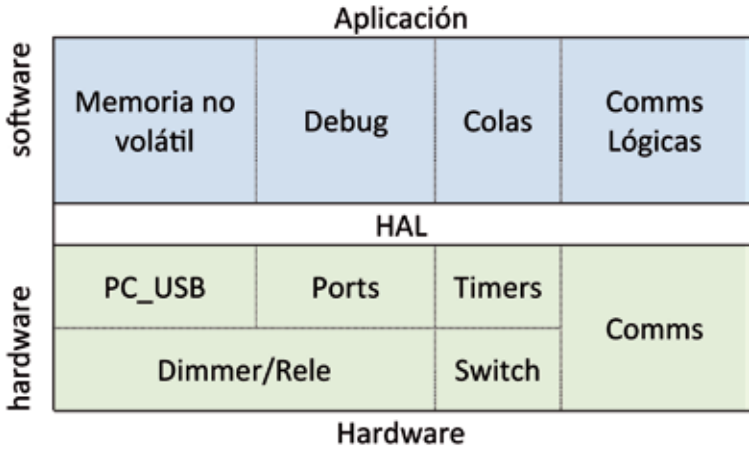
Teniendo en cuenta los objetivos del proyecto es que decidí enfocar el diseño los siguientes puntos:

1. Desarrollo del firmware basado en capas.
2. Comunicaciones alámbricas como inalámbricas.
3. Desarrollo en lenguaje C.

El desarrollo del firmware en capas provee un sistema independiente del hardware en cual se ejecuta a través de la implementación de una capa de abstracción o HAL por sus siglas en inglés Hardware Abstraction Layer.



Diseño en capas



La capa de software contiene la lógica y servicios necesarios para el funcionamiento del sistema, también será la responsable de la inicialización y el ciclo principal de programa. La capa de abstracción de hardware (HAL) provee una interfaz por la cual se puede acceder al hardware aunque este cambie. Esto se logra a través de definir un conjunto de prototipo de funciones que luego serán implementadas para cada uno de los distintos dispositivos de hardware en la capa inferior.

Las comunicaciones jugaron un papel muy importante en el diseño del sistema ya que las mismas definen en gran medida que componentes se utilizaron para la implementación del mismo. Uno de los objetivos del proyecto fue que las comunicaciones tengan la posibilidad de ser tanto inalámbricas como alámbricas teniendo en cuenta que el mercado actual tiende a la primera opción. Esto se debe a que la implementación de una sistema inteligente en una vivienda ya construida es mucho más sencilla si se realiza de manera inalámbrica, ya que no es necesario cablear los distintos dispositivos entre si.

Fue necesario establecer las siguientes decisiones en cuanto al diseño de las comunicaciones inalámbricas:

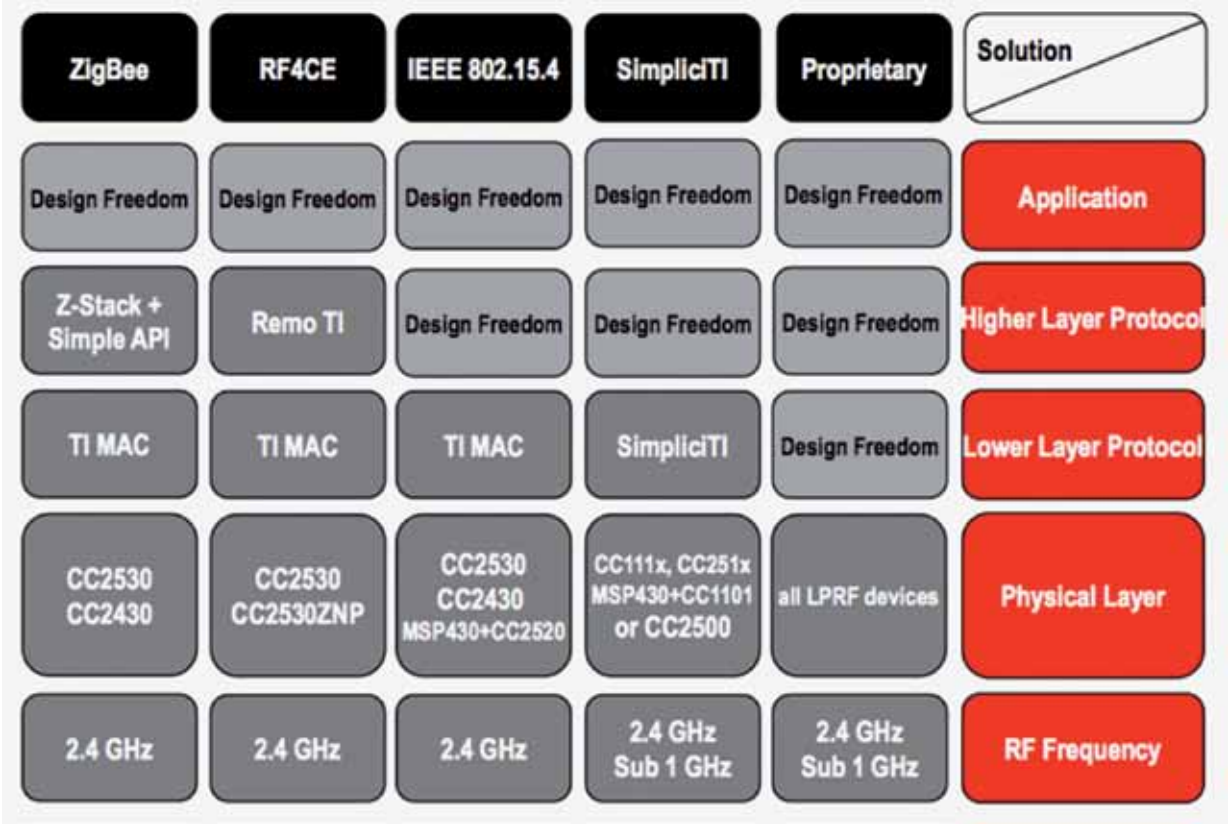
A. Elección de la tecnología a utilizar.
B. Elección del protocolo a utilizar.
A. ¿Qué tipo de tecnologías inalámbricas existen y qué beneficios ofrecen? Son las preguntas que surgieron inicialmente, de forma sencilla y/o general se pueden resumir en la siguiente tabla.

Tecnología	Rendimiento (Mbps)	Consumo	Alcance (m)	Frecuencia (Ghz)
Wi-FI	300	elevado	50-100	2,4 / 5
Bluetooth	3	medio	10-100	2,4
Zigbee	1	bajo	10-100	2,4
RFID	0,05	ninguno	0-1	varios estándares

Zigbee presentó la mejor opción ya que los dispositivos son de bajo consumo, la señalización de sensores y actuadores no requiere de grandes velocidades de comunicación y al trabajar en la banda de ISM (Industrial, Scientific, Medical) 2,4Ghz su uso está permitido sin necesidad de licencias siempre respetando las regulaciones que limitan los niveles de potencia transmitida.

B.- Si bien Zigbee engloba un tipo de tecnología su implementación varia según el protocolo que se utilice. El protocolo define en gran medida otras ventajas y desventajas que debieron ser analizadas, la siguiente tabla[6] delimita en base al tipo de estándar elegido (en este caso stacks provistos por Texas Instrument) que capas quedan libres para el diseño y desarrollo del sistema.

6. Información extraída de http://www.ti.com/lit/sg/slya020a/slya020a.pdf



A su vez cada estándar influye en el tipo de topología, tamaño de código (footprint como se conoce en inglés) y la complejidad e implementación como se define a continuación[7].

	Any Radio HW + Proprietary SW	SimpliciTI	802.15.4 TIMAC	RF4CE	ZigBee
Topology	**Any Topology**	**Point to Point** **Star Network**	**Star Network**	**Star Network**	**Mesh**
Code Size	**variable**	**< 8 KByte**	**<32 KByte**	**<64 KByte**	**>64 KByte**
Complexity	**variable**	**Low**	**Low**	**Low**	**Medium**

Al analizar beneficios/desventajas de cada uno surgió que las mejores opciones para el proyecto eran Z-Stack[8] y TiMAC[9]. Si bien Z-Stack permite generar redes mesh lo hace a costa de requerir un footprint más grande, esto influye en que el microcontrolador debe tener más espacio para poder alojar el firmware y ello por lo general se ve reflejado en un mayor costo precio x unidad. La complejidad en la implementación de Z-Stack se ve afectada, en parte, porque las comunicaciones son orientadas a la conexión y

7. Información extraída de http://www.ti.com/lit/sg/slya020a/slya020a.pdf
8. Más información en http://www.ti.com/tool/z-stack
9. Más detalles en http://www.ti.com/tool/timac

esto implica que antes de que cualquier comunicación entre dispositivos pueda ocurrir debe generarse un camino de comunicación entre ambos. Al tratarse de un sistema de comunicaciones en tiempo real decidí utilizar protocolos de comunicación no orientados a la conexión, esto significa que entre dos dispositivos de la red un mensaje puede ser enviado desde uno a otro sin previo acuerdo. El dispositivo en un extremo de la comunicación transmite los datos al otro, sin tener que asegurarse que el dispositivo receptor está disponible y listo para recibir los datos lo cual presenta ciertas ventajas y desventajas, debajo las más relevantes para el proyecto.

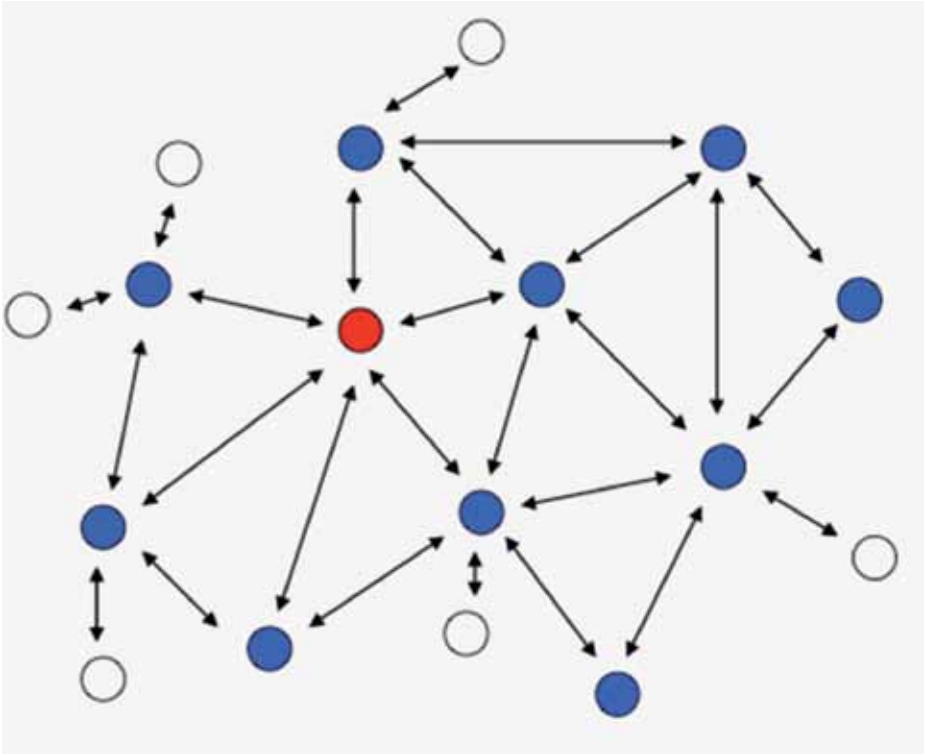
Ventajas:

- El tamaño de los datos adicionales para enviar un paquete es más pequeños.
- No es necesario establecer un camino de comunicación antes de transmitir los datos lo que acelera los tiempos de transmisión, importante para sistemas en tiempo real.

Desventajas:

- El protocolo no asegura que el envío o recepción del paquete de comunicación haya sido satisfactorio.

Con respecto a la topología una red mesh puede ser representada por el siguiente diagrama.



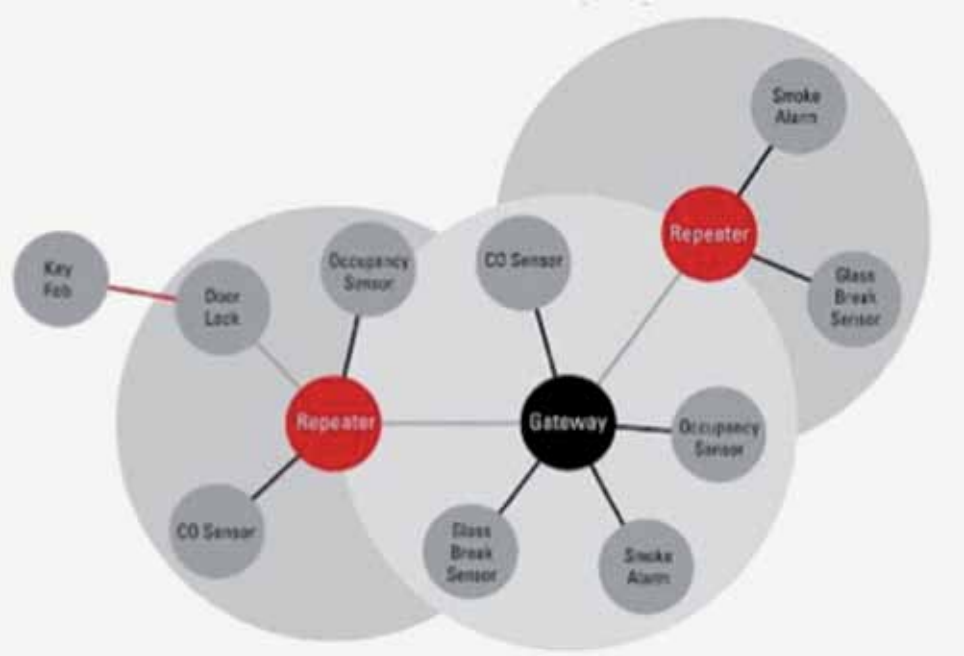
Dispositivos:

- Azul ⟶ direccionador Zigbee (Zigbee Router), direcciona paquetes y asocia a Zigbee Routers y End Devices.
- Blanco ⟶ Dispositivo Final (Zigbee End Device), no direcciona sólo envía y recibe paquetes.
- Rojo ⟶ Coordinador (Zigbee Coordinator), Inicia la red, direcciona paquetes, maneja la seguridad, asocia Routers y End Devices.

Este tipo de topología permite que dispositivos que no estén en el alcance del Coordinador de la red puedan unirse de todas maneras o que dispositivos que pretendan comunicarse y no estén en directo alcance puedan hacerlo gracias a los dispositivos de tipo Router que direccionarán las comunicaciones entre ellos. Pero como se mencionó esto impacta en la complejidad del sistema.

La topología disponible en TiMAC, estrella (star en ingés), puede ser representada por el siguiente diagrama[10].

10. Información extraída de http://www.ti.com/lit/sg/slya020a/slya020a.pdf



Dispositivos:

- Negro ⟶ Gateway/Coordinador, Inicia la red, asocia Coordinadores y End Devices.
- Rojo ⟶ Repeater (Repeditores), son capaces de reenviar y/o retransmitir paquetes.
- Gris ⟶ End Devices, Dispositivos Finales pueden comunicarse peer-to-peer.

Como se ve la desventaja de TiMAC es que si un Dispositivo Final no está al alcance del Coordinador requiere de un Repetidor para unirse a la red. Sin embargo la gran ventaja de TiMAC es que al tratarse de un stack de menor nivel que Z-Stack permite realizar comunicaciones peer-to-peer, es decir comunicaciones directas entre dispositivos finales, lo que permite implementar una comunicación no orientada a la conexión.

Además como puede observarse en las tablas anteriores Z-Stack está basado en TiMAC por lo que la libertad en el diseño y de las capas de Aplicación y Red permiten que uno pueda desarrollar e implementar su propia red mesh de ser necesario. De esta manera opté por utilizar TiMAC para la implementación del sistema.

La decisión de optar por TiMAC en 802.15.4 influyó también en la elección del hardware a utilizar, la opción elegida fue el integrado CC2530 de Texas Instruments que contiene un MCU (Micro Controller Unit) 8051 junto con un módulo transmisor-receptor 2.4Ghz IEEE 802.15.4 RF. Tratándose de hardware sus características no serán analizadas en ésta tesina.

En cuanto al lenguaje C este es considerado como nivel medio pero con muchas características y estructuras habituales de los lenguajes de alto nivel. A su vez dispone de construcciones del lenguaje que permiten un control a muy bajo nivel y se puede generar código con mucha eficiencia. Debido a estas características es que relativamente sencillo desarrollar compiladores de C, estos se encuentran disponibles para infinidad de plataformas lo que es un punto clave dentro de los objetivos del proyecto (multiplataforma).

Implementación

En el siguiente capitulo se desarrollará la implementación del proyecto siendo extensivo en el desarrollo del firmware y una descripción general del hardware diseñado y creado.

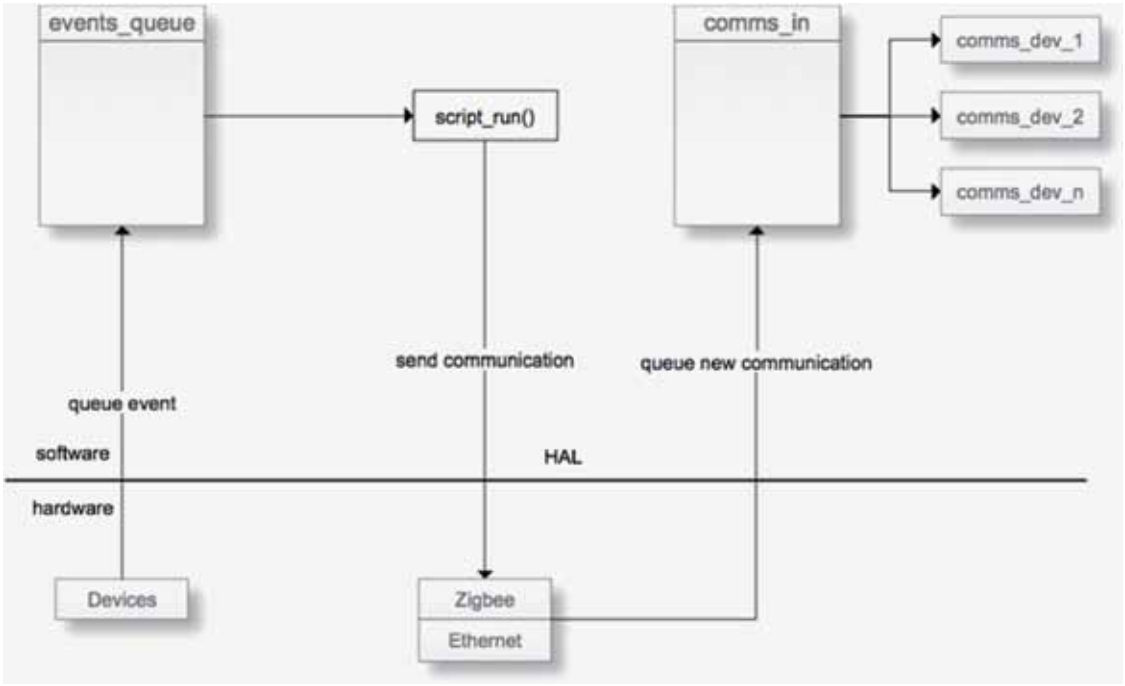
Firmware

Como se mencionó en anteriormente el proyecto está divido en capas, se presenta a continuación una visión general de la implementación .

HAL divide el software del hardware y provee una interfaz de comunicación entre ambos. El flujo de información sería el siguiente:

1. Los dispositivos o devices envían eventos al software a través de HAL.
2. Estos eventos son analizados y de requerir una acción se ejecuta un script (secuencia de pasos programada)

3. De ser necesario el script puede comunicarse con el módulo de comunicaciones localizada en el hardware a través de HAL para enviar mensajes a otros dispositivos tan sólo conociendo su número de identificación lógico. Más detalle debajo.
4. El módulo de comunicaciones puede también recibir comunicaciones y enviar dichos datos a la capa de software para que sean procesados.



Para la implementación del proyecto se definieron los siguientes eventos[11] que pueden ser recibidos por los dispositivos:

- DIMMER_UP, DIMMER_DOWN, DIMMER_CHANGE.
- SENSOR_CLOSE, SENSOR_OPEN, SENSOR_CHANGE.
- SWITCH_UP, SWITCH_DOWN, SWITCH_CHANGE.

El nombre de cada evento describe por si solo el tipo de evento recibido. Estos eventos no han sido seleccionados al azar sino que fueron parte de la investigación y diseño inicial de los dispositivos, pero al tratarse de elementos de Hardware estos detalles no serán expuestos en esta trabajo.

Una vez que estos eventos son procesados pueden generar un paquete de comunicación para ejecutar cierta acción, por el momento los códigos de acción[12] que se envían por la red son los siguientes:

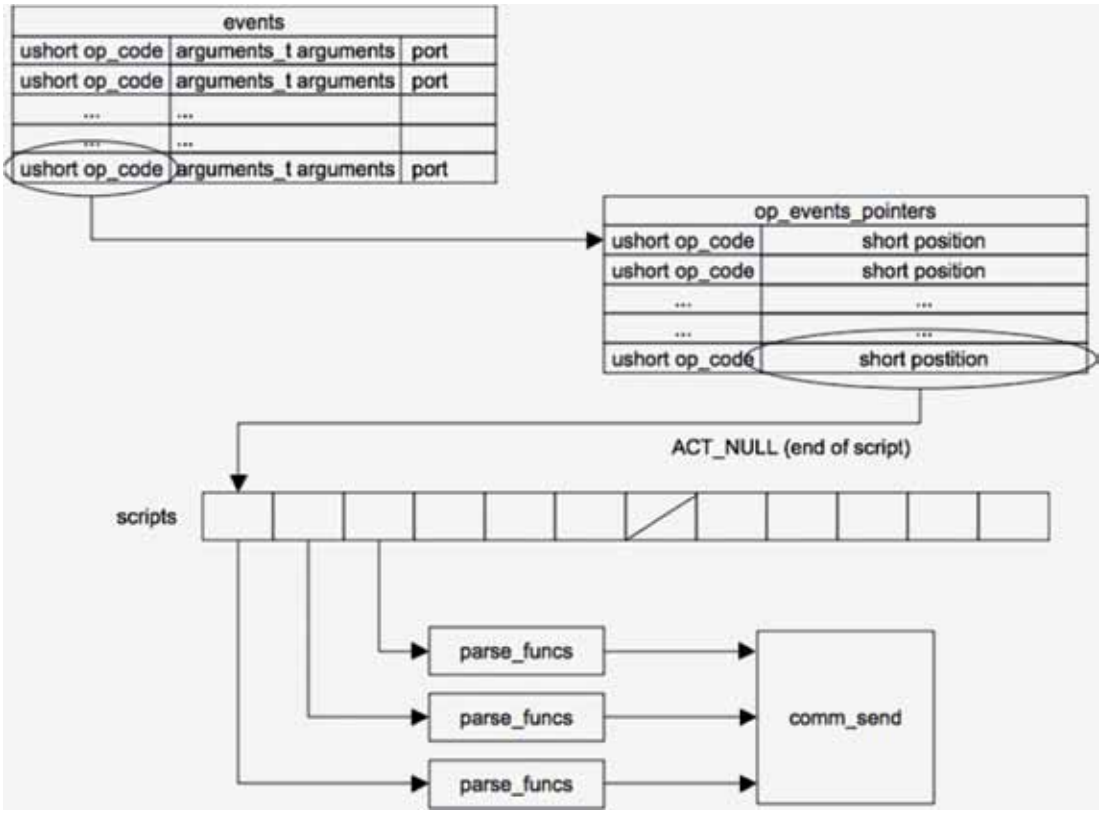
- REL_TURN_ON, REL_TURN_OFF, REL_SET_VAL.
- REL_SET_CHANGE, REL_SET_RELATIVE, REL_GET_VAL, REL_RET_VAL.
- TEMP_GET_VAL, IR_SEND, SWITCH_GET_VAL, SWITCH_RET_VAL.
- SENSOR_ON, SENSOR_OFF, SENSOR_GET_VAL, SENSOR_RET_VAL.
- DIMMER_GET_VAL, DIMMER_RET_VAL.

Si bien ambas listas son finitas el sistema a sido diseñado para su escalabilidad y permite agregar más eventos y/o códigos de acción.

En cuanto a los scripts que se ejecutarán es necesario explicar como se almacenan los mismos en memoria. El siguiente diagrama vemos:

- Cada evento tiene asociado un op_code que define el tipo de evento.
- Conociendo el op_code del evento recibido el sistema busca dentro de una tabla de punteros a eventos (ver IDE y Configuración del MBU) la dirección de donde comenzar a leer cada línea del script a ejecutar.
- Una vez determinado la posición inicial del script dentro del espacio de scripts (espacio reservado en memoria no volátil) el mismo se ejecuta línea a línea. Es necesario realizar una conversión de evento (event_t) a paquete de comunicación (comms_send) para ello se ejecuta parse_funcs. Una vez alcanzada una línea definida con código de acción ACT_NULL se detiene la ejecución del script.

11. Buscar event_op_code en la documentación web para mayor detalle en la implementación. Ver Anexo A.
12. Buscar action_op_code en la documentación web para mayor detalle en la implementación. Ver Anexo A.



Diseño del espacio de memoria para los scripts

Comunicaciones

En lo que respecta a las comunicaciones entre distintos MBUs[13] fue necesario tomar ciertas decisiones para cumplir uno de los objetivos del proyecto. El sistema debería funcionar tanto de forma alámbrica, es decir cableada, como de forma inalámbrica, sin cables.

Para la implementación cableada decidí utilizar el protocolo UDP/IP sobre Ethernet 802.3 que es de amplio conocimiento y en cuanto a la implementación inalámbrica como se mencionó en la etapa de diseño TiMAC fue el stack a implementar sobre 802.15.4

Como podemos ver en la siguiente tabla el stack TiMAC provee las librerías e implementaciones para el acceso a las capas de enlace y físicas dejando a mi criterio el diseño y desarrollo de las capas superiores del modelo OSI[14] (Open System Interconnection).

OSI	UDP/IP	TiMAC
Aplicación	Aplicación	Aplicación
Presentación		
Sesión		
Transporte	UDP	
Red	IP	
Enlace	Ethernet IEEE 802.3	IEEE 802.15.4
Física		2400MHz

Capas de Red y Transporte

El modelo OSI define a la capa de red como la encargada de realizar el direccionamiento lógico de la red y a la capa de transporte como la responsable del empaquetamiento de los datos y de proveer la fiabilidad de los mismos durante la comunicación. Para satisfacer estos requerimientos sobre ambos protocolos UDP/IP y TiMAC el firmware desarrollado para el proyecto cuenta con la siguiente lógica.

Transporte: Como se mencionó anteriormente para cada evento generado por el hardware existe un

13. Ver Implementación – Hardware.
14. http://es.wikipedia.org/wiki/Modelo_OSI

paquete de comunicación que será enviado a uno o más MBUs. Estos paquetes deben formarse teniendo en cuenta ciertos detalles. La estructura básica que contiene la información de un evento a enviarse es comms_t como podemos ver esta estructura de 7 bytes contiene,

- action_code ⟶ código de acción a ejecutar.
- device_num ⟶ número de dispositivo que originó el evento
- node_num ⟶ número de nodo en la red que aloja al dispositivo.
- exec_time ⟶ cuanto tiempo ha de esperarse antes de realizar la acción. Siendo que 2 bytes no signados representan 65535 y la resolución es en segundos esto representa una espera de cómo máximo 18hs.
- action_arg ⟶ argumentos relacionados con el código de acción a ejecutarse.
- fixed_arg ⟶ campo destinado para ciertos códigos de acción que siempre utilizan el mismo tipo de argumentos.

comms_t					
action_code	address_t		exec_time	action_arg	fixed_arg
	device_num	node_num			
1 byte	1 byte	1 byte	2 bytes	1 byte	1 byte

Ahora bien, es necesario formar un paquete que contenga a comms_t y lo prepare para ser enviado por la red. Aquí la capa de transporte agrega los datos necesarios para asegurar la fiabilidad de los datos. El paquete queda definido por la estructura packet_t de 15 bytes.

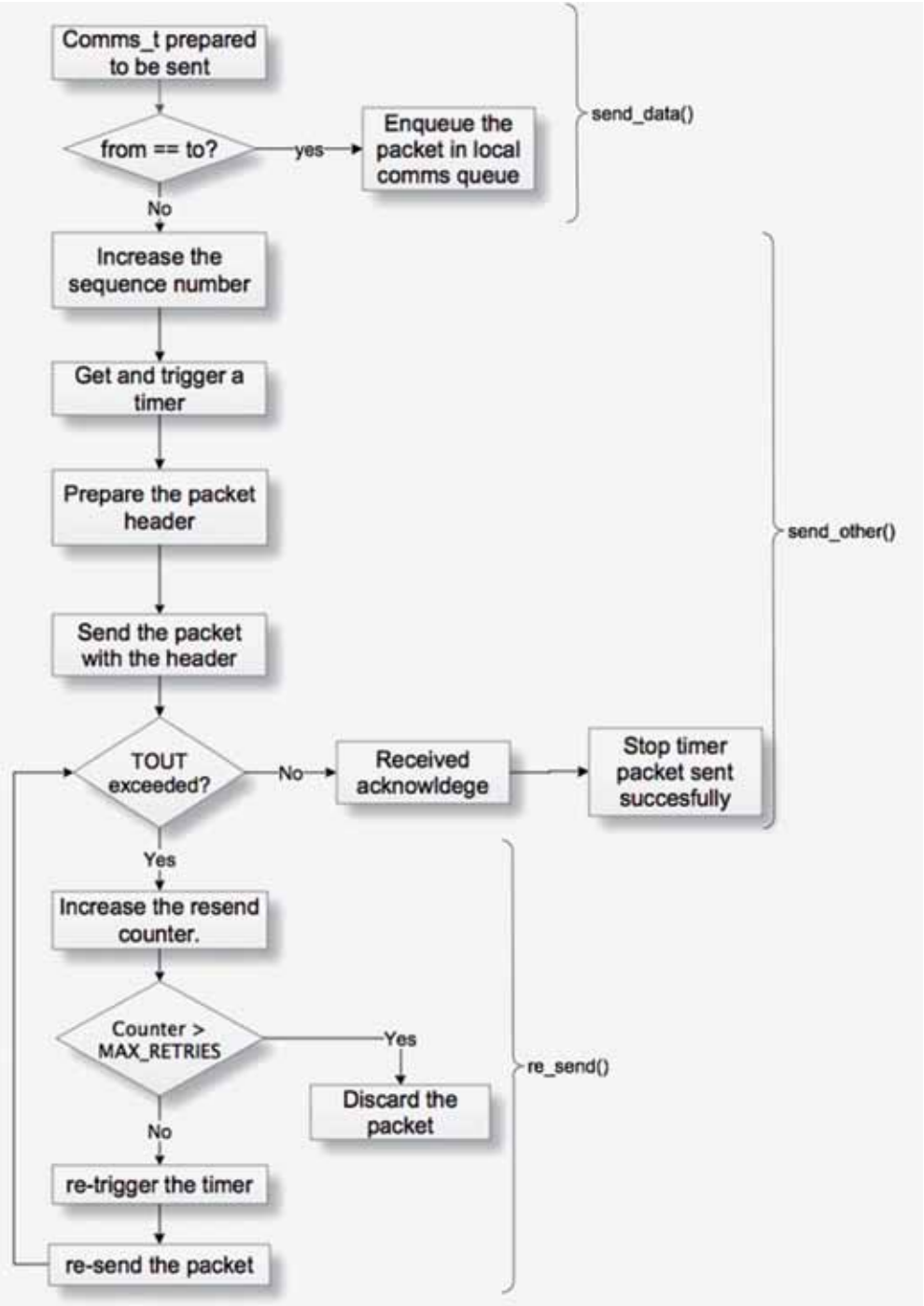
- from ⟶ dirección lógica del MBU que envía el paquete.
- to ⟶ dirección lógica del MBU a quien enviar el paquete.
- header_type ⟶ encabezado para definir el tipo de paquete.
- sequence ⟶ número de secuencia del paquete.
- time_handler ⟶ identificación del controlador de tiempo de re-envío.
- comms_t ⟶ estructura con la información del evento.
- local_seq ⟶ número de secuencia local, utilizado para sincronización entre MBUs remotos.

packet_t						
from	to	comm_header_t			data_t	
		header_type	sequence	time_handler	comms_t	local_seq
1 byte	1 byte	1 byte	2 bytes	1 byte	7 bytes	2 bytes

Los tipos de header o encabezados que la capa de transporte utiliza son,

- CTRL ⟶ Utilizado para cuando el paquete es de tipo acknowledge.
- APPL ⟶ Paquete de aplicación, utilizado para enviar acciones.
- STATUS ⟶ Utilizado para responder a métodos que piden el estado de un dispositivo.
- SYN_SEQ ⟶ Utilizado para sincronizar los números de secuencia entre distintos dispositivos.

Una vez definido el packet_t con la información requerida la capa de transporte envía el paquete a la capa de red para ser transmitido. Podemos ver entonces toda este proceso descripto en el siguiente flujo.



Cuando el paquete ha sido enviado a la capa de red en el bloque “Send the packet with header” la capa de transporte controla que durante el tiempo definido en TOUT se reciba un paquete de tipo acknowledge (acuse de recibo) en caso negativo el paquete se vuelve a enviar hasta MAX_RETRIES veces. Con este mecanismo se pretende asegurar la fiabilidad de la transmisión.

Direccionamiento lógico: En el momento de configuración a cada MBU se le asigna una dirección lógica mayor a 0 y menor a MAX_NODES esto hace que la configuración e identificación de MBUs se sencilla de recordar para el usuario final, pero a nivel a red cuando trabajamos con Ethernet y/o TiMAC las direcciones físicas tienen 32 bits y 16/64 bits respectivamente. Es por esto que la capa de red tiene la lógica necesaria para realizar esta conversión y realizar el envío como corresponde.

Una vez recibido el paquete de la capa de transporte es necesario agregar un encabezado adicional para formar la estructura low_level_pkg_t de 16 bytes.

low_level_pkg_t	
header	packet_t
1 byte	15 bytes

Estos encabezados de bajo nivel son,

- DATA_PKG ⟶ Paquete normal, contiene datos de la capa de aplicación.
- ACK_ADDRESS_PKG ⟶ Paquete para enviar un acuse de recibo luego de haber recibido un REQ_ADDRESS_PKG.
- REQ_ADDRESS_PKG ⟶ Paquete para solicitar la dirección física de un MBU remoto.

¿Qué sucede entonces cuando un MBU intenta comunicarse con otro? Para hacer esto posible el firmware ejecuta la siguiente lógica, supongamos por un momento que uno de nuestros MBU designado con el localhost = 1 (MBU_1) intenta comunicarse por primera vez con el MBU configurado como localhost = 3 (MBU_3), siempre teniendo en cuenta que esto es transparente para las capas superiores a la de red.

0) Inicialmente el MBU_1 contiene en memoria RAM la lista de direcciones físicas de los demás dispositivos en blanco ya que no se ha actualizado aún.

localhost	802.15.4/802.3
1	
2	
3	
n	

1) Un nuevo evento de hardware llega al MBU_1 la capa de transporte crea el packet_t necesario y luego lo envía a la capa de red y se determina que es necesario una comunicación con el MBU_3. La capa de red del MBU_1 inicia un broadcast[15] con un paquete encabezado como “REQ_ADDRESS_PKG” solicitando la dirección física del dispositivo con localhost = 3.
2) Todos los dispositivos en la red reciben el mensaje pero sólo el MBU_3 responde con la información solicitada para el MBU_1 enviando un paquete encabezado como “ACK_ADDRESS_PKG”. A su vez el MBU_3 actualiza su propia tabla y si no contenía el detalle de la dirección física de MBU_1 guarda su valor ya que el mismo es transmitido dentro del paquete broadcast.
3) En futuras comunicaciones ambos MBUs ya conocen sus direcciones y no enviarán un mensaje de broadcast sino un unicast[16].
4) Aunque un MBU conozca la dirección física del MBU remoto con el cual intenta comunicarse como se mencionó anteriormente si no se recibe un acknowledge en la capa de transporte, la capa de red limpiará el registro en la tabla local y volverá a ser vacío.

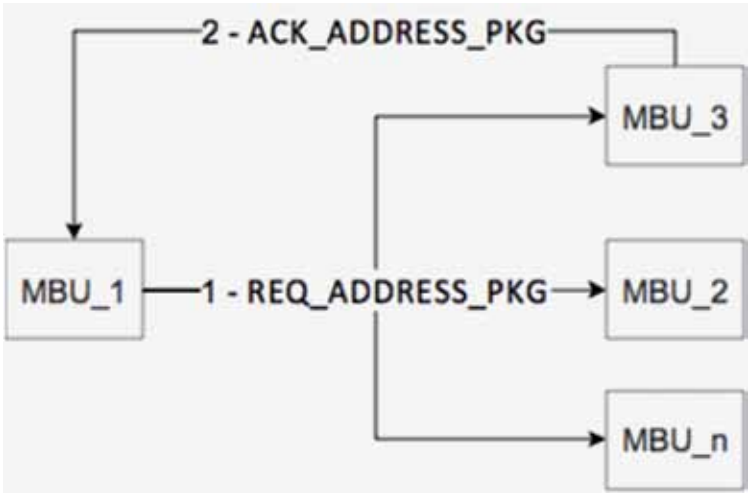


Diagrama de comunicación broadcast

15. Transmisión de información que será recibido por todos los dispositivos de la red.
16. Transmisión de información desde un único emisor a un único receptor.

Tabla en MBU_1

localhost	802.15.4/802.3
1	
2	
3	xxxx
n	

Tabla en MBU_3

localhost	802.15.4/802.3
1	xxxx
2	
3	
n	

Capa de Red

Como se mencionó anteriormente en la sección de diseño opte por utilizar el stack TiMAC provisto por Texas Instruments, esto implicó que debí configurar el código para que los distintos MBUs de la red coordinaran entre sí quién se convierte en coordinador de la red. Para ello el firmware ejecuta la siguiente lógica en el inicio del sistema.

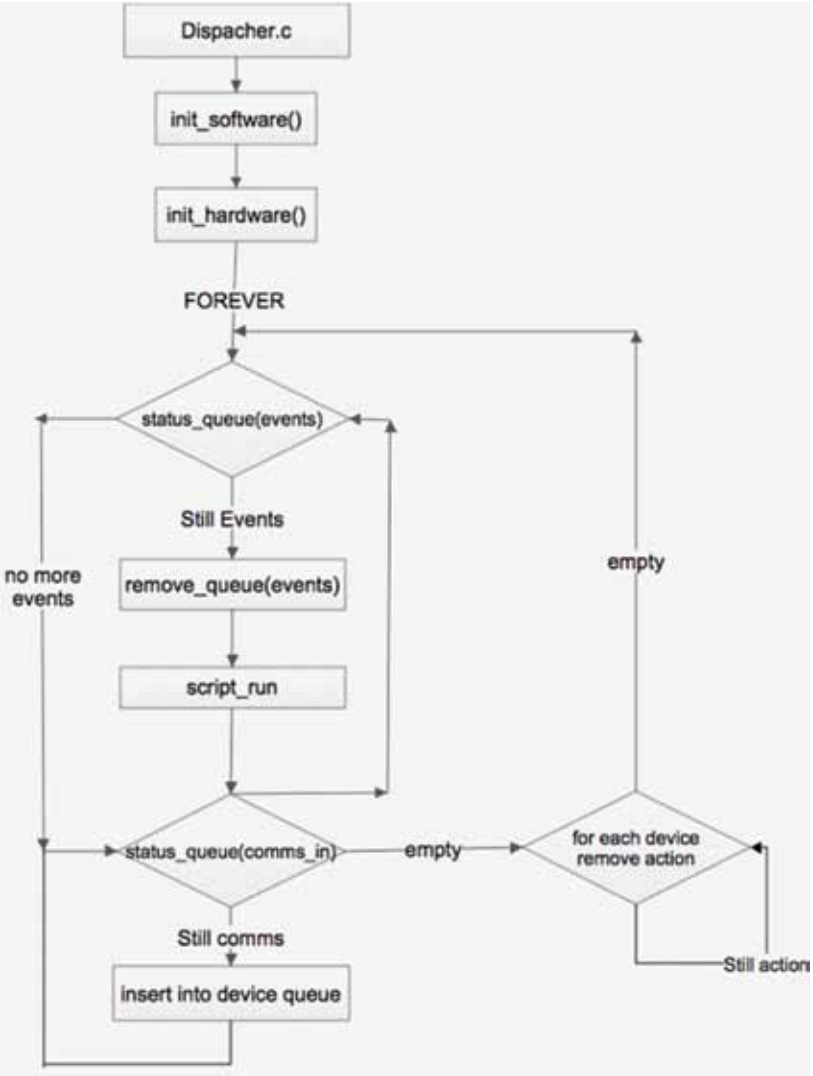
1. Se cargan los valores por defecto de la red como, número de PAN, dirección corta del coordinador y canal de comunicación entre otras.
2. Se envía un mensaje de escaneo a la red buscando por un coordinador activo que responda al mensaje, si luego de un tiempo de espera configurado no se recibe respuesta alguna el MBU inicia en modo Coordinador + Dispositivo Final.
3. Si otro MBU inicia dentro de la misma red al enviar el mensaje buscando por el coordinador el MBU iniciado como coordinador responde y el segundo MBU inicia el pedido de asociación a la red. De esta manera el coordinador de la red le asigna una dirección corta dentro de la red e inicia como Dispositivo Final.

Capa de Aplicación

Con respecto a la capa de aplicación se analizará el ciclo principal del firmware el cual ha sido nombrado dispatcher o despachador para su traducción en español. Para ello presentaré el flujo de dispatcher.c[17] para luego detallar cada una de sus partes.

17. Dispactcher.c contiene la implementación del ciclo principal programada en C.

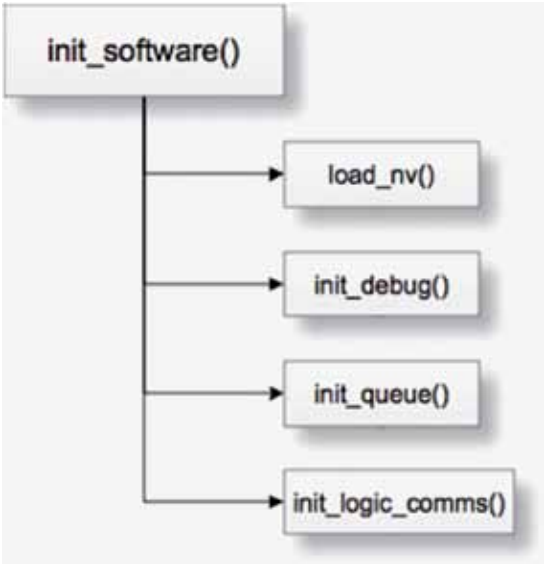
Dispatcher



Dispatcher contiene tanto el conjunto de funciones para inicializar el sistema como el ciclo principal que será encargado de procesar y mantener activo el sistema.

Init_Software()

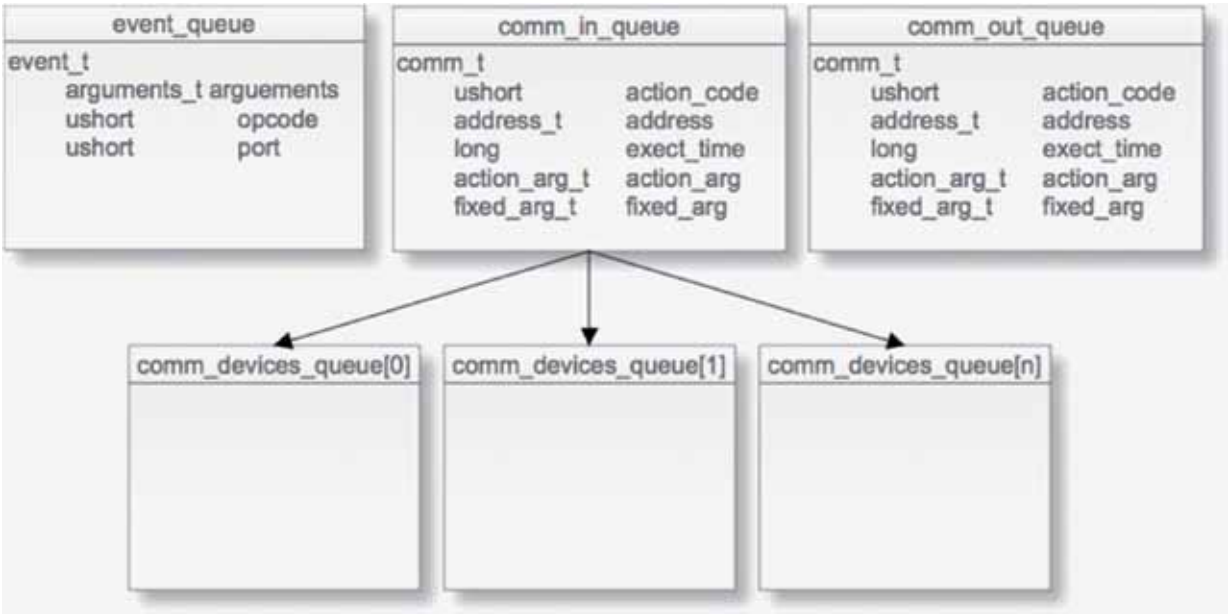
Se encarga de inicializar todas los recursos de software necesarios para el funcionamiento del sistema.



- load_nv() carga los valores no volátiles del sistema en memoria RAM. Los valores no volátiles son aquellos que permanecen configurados en el firmware incluso si la unidad pierde energía como ser número de unidad dentro de la red, scripts a ejecutar, dispositivos que contrala y otras implementaciones internas.
- init_debug() inicializa los servicios de debugging o de depuración. Dependiendo de la configuración el sistema puede enviar mensajes de debug por distintas interfaces para analizar el estado y funcionamiento del mismo. Para poder ver este pido de mensajes desarrollé una interfaz (IDE[18]) de comunicación con el dispositivo que se explicará más adelante.
- init_queue() como todo sistema de tiempo real más de una tarea o evento puede llegar al sistema por lo que es necesario contar con un servicio de colas o queues en inglés para poder administrar los eventos. Al inicializar el sistema de colas distintas funciones pueden ser utilizadas como insertar, remover y verificar el estado de cada una de las colas configuradas.
- init_logic_comms() se encarga de inicializar los recursos de comunicaciones a nivel lógico. A nivel lógico cada MBU es representada por un número entero no igual a 0 que se guarda en memoria no volátil.

La inicialización de las colas se describe con mayor detalle, el diseño del sistema requiere de tres tipos de colas:

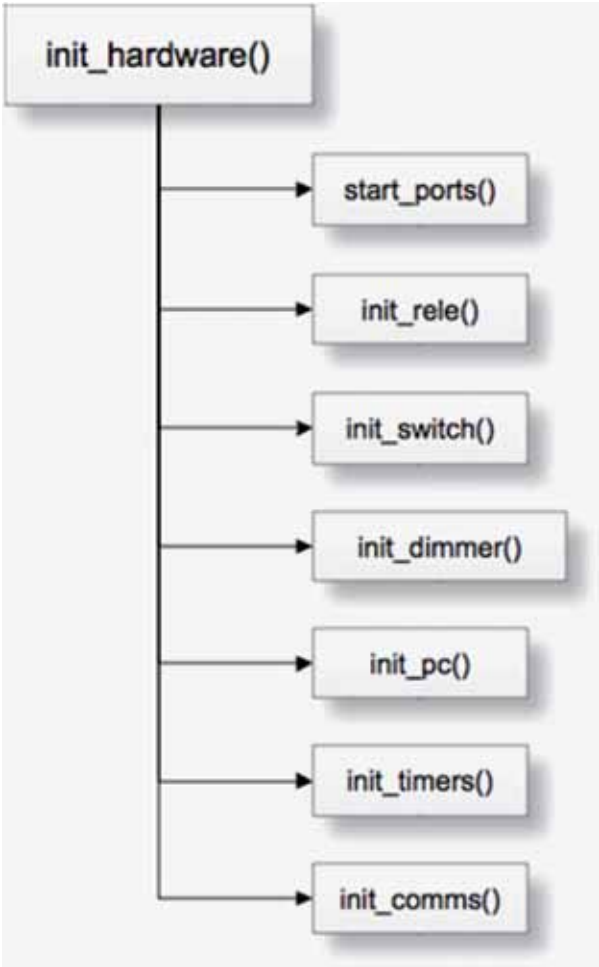
- event_queue utilizada para almacenar los eventos que lleguen al sistema a través del hardware.
- comm_in_queue contiene las comunicaciones que deben ejecutarse localmente. A su vez Dispatcher distribuye los elementos de comm_in_queue en colas individuales para cada uno de los módulos conectados al MBU.
- comm_out_queue contiene las comunicaciones que deben ejecutarse en una unidad remota.



18. IDE de las siglas en inglés para Entorno de Desarrollo Integrado.

Init_hardware()

Si bien esta tesina está enfocada al firmware/software del proyecto es necesario introducir algunos datos acerca del hardware y como se implementa en cada caso. Como init_hardware es llamado desde la capa de software cada una de estas llamadas pasan a través de HAL y luego cada implementación varia según la plataforma donde se ejecute el firmware.



- start_ports() rutina encargada de inicializar los puertos de comunicación del sistema a bajo nivel.
- init_rele(), init_switch(), init_dimmer() son las rutinas para inicializar los servicios de bajo nivel para los distintos módulos/actuadores. En la etapa inicial del proyecto sólo estos tres módulos fueron desarrollados pero el sistema esta diseñado para poder expandir su uso a otros módulos como sensores de temperatura, controladores de infrarrojo que cuya inicialización debería ser incluida en init_hardware() en el futuro.
- init_pc() inicializa los servicios para que el software IDE pueda comunicarse con el dispositivo para programarlo o recibir mensajes de depuración (debug) del mismo.
- init_timers() el servicio de timers o temporizadores en español es de gran necesidad en un sistema de tiempo real en el cual es necesario llevar cuenta del tiempo para dar sincronismo a los procesos y programar tareas o eventos a realizar en determinado tiempo. Su implementación esta ligada al hardware por lo cuál su código se incluye en la capa de hardware. Una vez inicializado el servicio de timers el sistema pueda iniciar, parar, re-iniciar y programar la ejecución de procesos y/o eventos a discreción.
- init_comms() esta rutina se encarga de la inicialización de los servicios de comunicaciones a bajo nivel, dependiendo de la plataforma distintos procesos serán ejecutados para inicializar pero en cualquier caso HAL proveerá servicios como, enviar, recibir, comprobar el estado de la red, re-envío de paquetes, descubrimiento de la red entre otras.

FOREVER

El ciclo FOREVER o “para siempre” en español es el ciclo principal donde se atenderán los eventos que llegan al sistema y las tomas de decisiones requeridas. Siguiendo el flujo de Dispatcher el ciclo realiza lo siguiente,

1. Verifica el estado de las colas.
2. Si hay eventos pendientes es la cola, se remueven de a uno y se ejecuta el script asociado al evento, bloques remove_queue() y script_run() respectivamente hasta que la cola event_queue quede vacía.
3. Si no hay eventos se revisa la cola de comunicaciones comms_queue y se distribuyen las comunicaciones a cada uno de los devices del sistema.
4. Se revisa cada cola de los devices y si hay comunicaciones pendientes se ejecutan todas antes de volver a iniciar el ciclo FOREVER.

Hardware

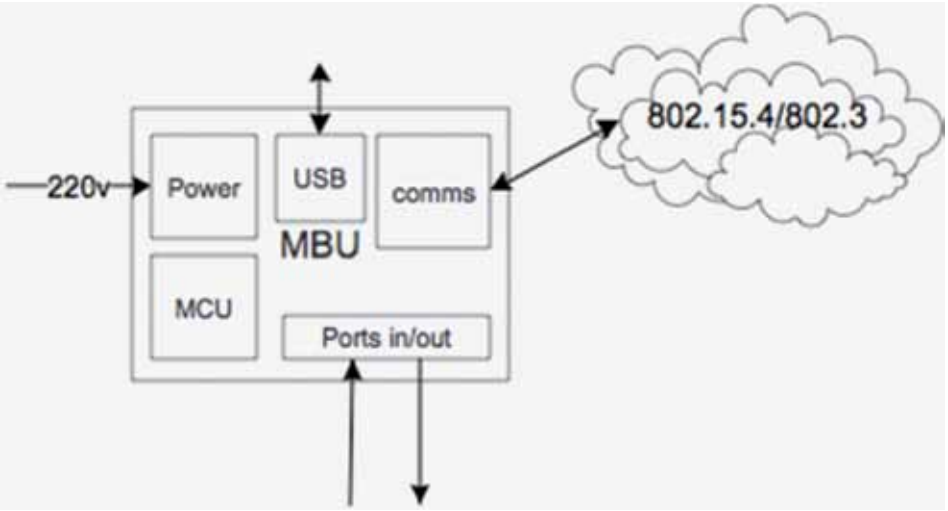
Dado que ésta tesina está enfocada en la parte de firmware aplicada al proyecto la información acerca del desarrollo e implementación del hardware se verán limitadas a mostrar el hardware en general con sus funciones a nivel alto para que sea fácil de entendimiento.

Como se mencionó anteriormente el firmware fue diseñado para funcionar en múltiples plataformas una de ellas un hardware independiente o standalone como se conoce en inglés.

El hardware consiste en de los siguientes componentes:

1. Main Board Unit (MBU) o Unidad Primaria
2. Módulos de expansión.
3. MBU Dev. Un kit de desarrollado diseñado para las pruebas y como prototipo funcional.

La siguiente ilustración representa de forma sencilla al MBU el cual consiste en una unidad funcional independiente alimentada por la energía eléctrica del hogar donde se instale. La unidad de proceso central o MCU contiene el firmware que es capaz de realizar comunicaciones 802.15.4 TiMAC o 802.3 Ethernet si es que el firmware se ejecuta en una PC. La MBU contiene 3 puertos de comunicación bidireccionales a los cuales se pueden conectar los distintos módulos de expansión, un módulo USB permite la conexión a la PC tanto para configuración y debugging.

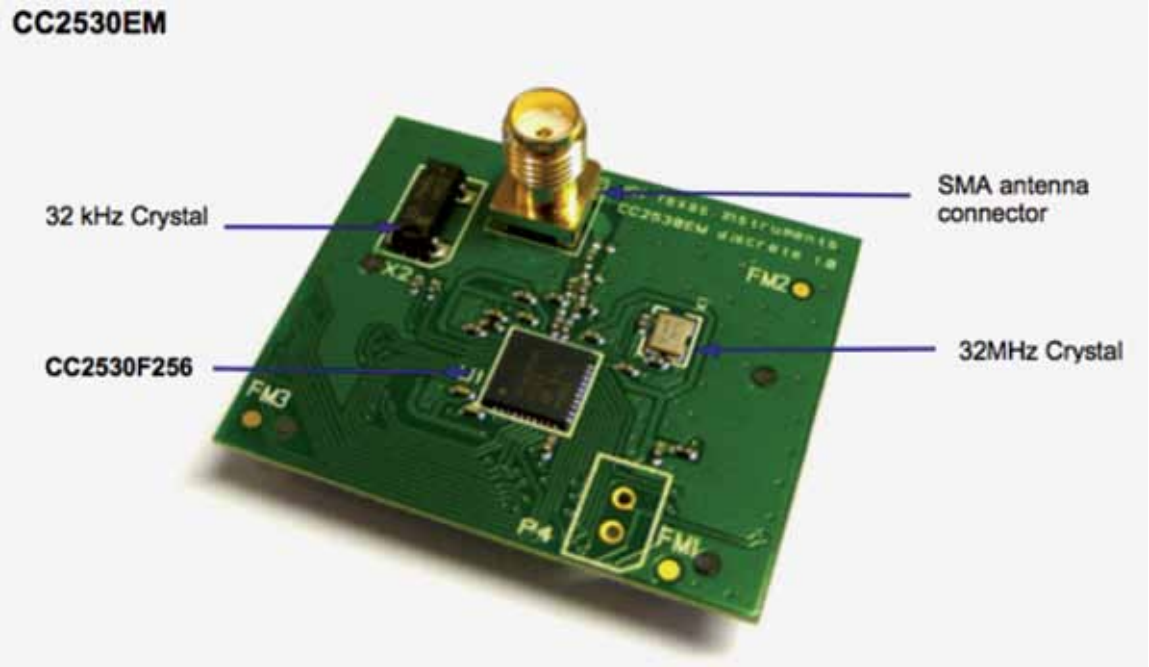


MBU – Main Board Unit

Esquemáticamente el MCU y la unidad de comunicaciones comms se encuentran desglosados pero en la implementación ambos forman parte del microcontrolador de Texas Instruments CC2530[19] que cuenta con un microcontrolador basado en 8050 junto con un módulo comunicaciones 802.15.4.

19. TI CC2530 http://www.ti.com/CC2530

A continuación una imagen del microcontrolador utilizado en el desarrollo del proyecto.



TI CC2530EM [20]

Los módulos de expansión desarrollados para ésta etapa del proyecto son:
Entrada:

- Encoder rotativo. Con la señal que el dispositivo genera el firmware es capaz de realizar acciones tales como atenuar la intensidad lumínica de lámparas, subir y bajar el volumen de un dispositivo entre otras.
- Llave o switch en inglés. Permite obtener una señal on/off que luego puede ser utilizada para distintos fines desde el firmware.

Salida:

- Triac. Éste dispositivo electrónico permite al MBU controlar dispositivos eléctricos de mayor potencia ya sea para prender, apagar u atenuarlos.

Otros dispositivos de entrada/salida pueden conectarse a los MBU tales como sensores/detectores de humo, movimiento, humedad entre otros. Su implementación no ha sido llevada a cabo pero el sistema está preparado para aceptarlos.

El MBU Dev representa un kit de desarrollo que se utilizó a lo largo del desarrollo del firmware en el cual se pueden conectar los módulos existentes y realizar pruebas de los diferentes escenarios y scripts a probar.

Como el microcontrolador CC2530 viene en encapsulados que no son aptos para ser soldados a mano, al realizar el proyecto adquirí un kit de desarrollo de Texas Instruments llamado CC2530DK[21] el cuál cuenta con el microcontrolador CC2530EM mostrado anteriormente. EM quiere decir Evaluation Module es decir un modulo de evaluación en el cual el microcontrolador ya se encuentra soldado con los componentes básicos para su funcionamiento y los pines de conexión están expuestos a través de dos zócalos de 20 pines cada uno. El kit contiene:

- 2 x CC2530EM
- 2 x SmartRF05EB, Placas de desarrollo que permiten grabar el firmware en el microcontrolador.
- 1 x CC2531, microcontrolador con capacidad para capturer pateques 802.15.4. Ver Pruebas y resultados para más detalles.

20. Imagen extraída de focus.ti.com/lit/ug/swru208b/swru208b.pdf
21. Más información acerca del kit en http://www.ti.com/tool/cc2530dk

IDE y configuración del MBU

Una de las características del sistema desarrollado es la capacidad que posee el usuario de crear scripts que se ejecutarán al momento de atender un nuevo evento. Para ello es necesario contar con una estructura de configuración y una forma de poder grabar/guardar esta información de forma no volátil en la memoria del MBU.

Frente a esta necesidad desarrollé un entorno sencillo pero eficiente para utilizar junto con el MBU_Dev en la etapa de desarrollo y puesta en marcha. El Integrated Development Environment (IDE) desarrollado es una interfaz que provee dos funcionalidades básicas:

1. Configuración del MBU.
2. Interfaz de debugging (depuración).

Cada MBU dentro del sistema tiene su propia configuración y se definen los siguientes datos para cada uno:

- Localhost, este valor identifica al MBU dentro de la red y es único para cada uno. Es un valor entero mayor a 0. No es necesario conocer el la dirección MAC del CC2530 ni la dirección IP si se corre el firmware en un entorno 802.3 ya que el sistema fue diseñado para poder ubicar a cada dispositivo tan sólo sabiendo su número de identificación lógico.
- Ports, define que dispositivos o módulos de expansión están directamente conectados al MBU.
- Script Lines, las líneas de script son las creadas por el usuario para dar funcionalidad al sistema y serán ejecutadas a la hora de atender a un evento. Si bien en esta etapa del proyecto la configuración de las mismas es poco amigable al usuario es evidente que esta característica debe ser mejorada en futuros desarrollos.
- Script Pts, es la información para definir el puntero a donde comienza cada script (ver Implementación – Firmware).
- Debug Level, con este parámetro definimos que tipos de información el MBU enviará al IDE a través de la interface USB.

Cada uno de estos parámetros es enviado al MBU a través de USB y se almacenan en memoria ROM, por lo tanto si el MBU pierde la alimentación eléctrica los datos no se pierden y permanecen guardados. Para verificar la integridad de la transferencia se uso comprobación CRC-16.

El IDE levanta la configuración programada para el MBU conectado usando archivos de configuración, para ello se utilizó una librería de código abierto llamada confuse[22] que provee métodos para trabajar con este tipo de archivos.

Como puede verse en el siguiente ejemplo de configuración el MBU se grabaría con:

- Localhost = 1
- Port_0 = SWITCH (entrada), RELE (salida)

22. Más detalles de confuse en http://www.nongnu.org/confuse/

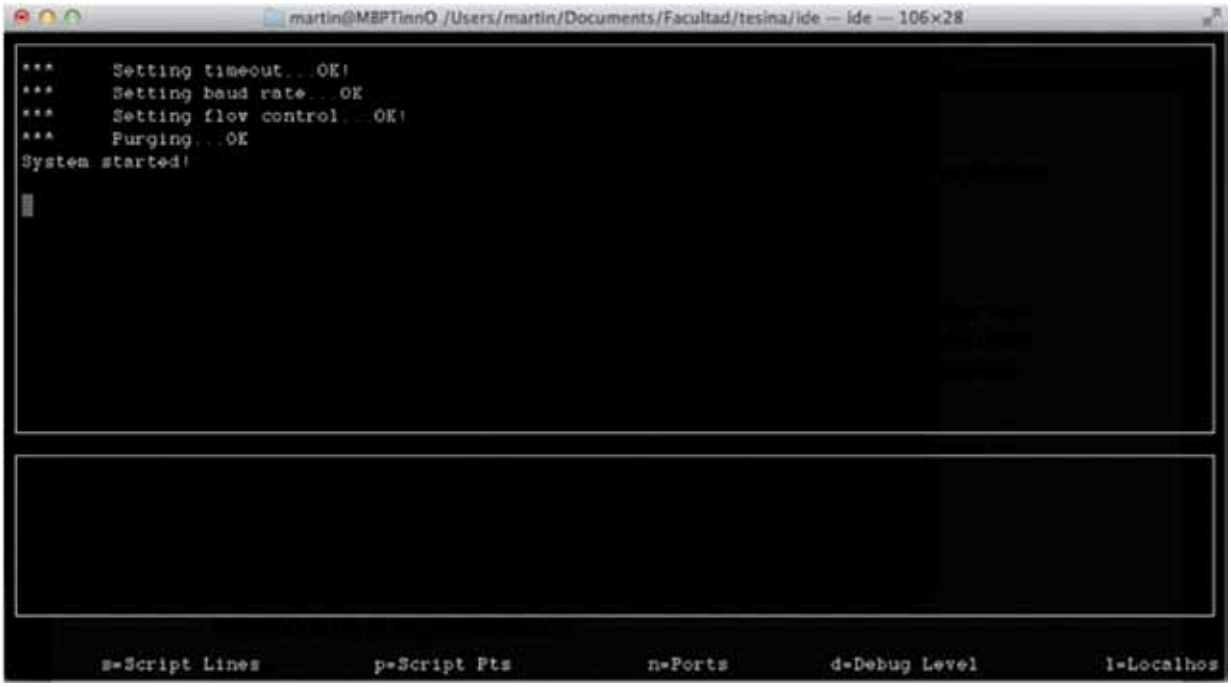
Port_1 = RELE (salida)
Port_2 = NONE (nada conectado)

- Script Pts indica que si el sistema recibe un evento SWITCH_DOWN desde el módulo conectado al puerto 0 se deberá ejecutar el script que está en la posición 1 del espacio de scripts.
- Script line contiene información acerca de dos líneas. La primera con el código de acción que debe ejecutarse en éste caso REL_TURN_ON (encender el relé de salida) en el MBU (nodo) número 1 y módulo 1 sin tiempo de espera. La segunda línea de script contiene código de acción igual a ACT_NULL por lo cual indica el fin del script.

En resumen lo que este MBU hará es encender el RELE conectado a él en el módulo 2 cada vez que el switch conectado en el módulo 1 pase a estado down o cerrado.

Ejemplo de archivo de configuración utilizado por IDE

```
#Sample configuration for Domotic Device
localhost = 1
script_line
{
line_number = 1
action_code = "REL_TURN_ON"
device_node = 1
node_num = 1
time = 0
fixed_arg = 0
}

script_line
{
line_number = 2
action_code = "ACT_NULL"
device_node = 1
node_num = 1
time = 0
fixed_arg = 0
}

script_pts
{
pos = "SWITCH_DOWN"
addr = 1
device_node = 0
}

port
{
port_number = 0
device_number = {"DEV_SWITCH", "DEV_RELE"}
}

port
{
port_number = 1
device_number = "DEV_RELE"
}

port
{
port_number = 2
device_number = "DEV_NONE"
}
```



IDE – corriendo en OS X

Además de las configuraciones expuestas anteriormente los MBU comparten configuraciones en común que son definidas antes de la compilación las mismas están definidas en gralconf.h[23], la siguiente es una lista de las mismas y su descripción:

- DEVICE_MOD ⟶ Modelo del MBU.
- DEVICE_VER ⟶ Versión del MBU.
- INTERVAL ⟶ Resolución mínima del sistema de timers o temporizadores.
- MAX_NODES ⟶ Máximo número de MBUs en la red.
- MAX_RELE ⟶ Máximo valor lógico en la escala del relé.
- MAX_TIMERS ⟶ Máximo número de timers que el sistema manejará simultáneamente.
- MAX_CTIMERS ⟶ Máximo número de timers reservados para comunicaciones. MAX_TIMERS – MAX_CTIMERS es la cantidad de timers libres para el resto del sistema.
- NUM_QUEUES ⟶ Número de colas que el sistema puede manejar simultáneamente.
- POOL_SIZE ⟶ Máximo número de elementos en el sistema de colas en general.
- SCRIPT_SPACE ⟶ Espacio reservado en ROM para almacenar la cantidad de script_lines que se deseen.
- MAX_RETRIES ⟶ Máximo número de reintentos al momento de enviar una comunicación si es que no se recibe un acknowledge.
- TOUT ⟶ Valor representado en segundos que determina cuanto tiempo ha de esperar el sistema a la espera de un acknowledge una vez enviado un mensaje unicast.
- TOUT_BROADCAST ⟶ Valor representado en segundos que determina cuanto tiempo ha de esperar el sistema a la espera de un acknowledge una vez enviado un mensaje broadcast.

Beneficios de la implementación

Los beneficios de la implementación están directamente ligados a los objetivos del proyecto. Es decir que se obtuvo un sistema de domótica más orientado al firmware y menos atado a una plataforma como las propuestas actuales del mercado.

Uno de los grandes beneficios de la implementación son los script personalizables. Con esta herramienta el usuario puede realizar infinidad de combinaciones y así lograr automatizar su vivienda como mejor le parezca. Si bien en esta etapa las posibilidades son más limitadas debidas a los pocos recursos (entrada y salida) disponibles como se plantea en la sección Líneas Futuras de Desarrollo esto puede cambiar gracias al potencial del proyecto.

Otro potencial beneficio es la liberación del código fuente a la comunidad open-source esto haría que el sistema fuese probado por muchos usuarios que podrían agregar funcionalidades o corregir potenciales errores.

23. Se puede obtener más detalles de gralconf.h en Anexo A.

Desarrollo Documental

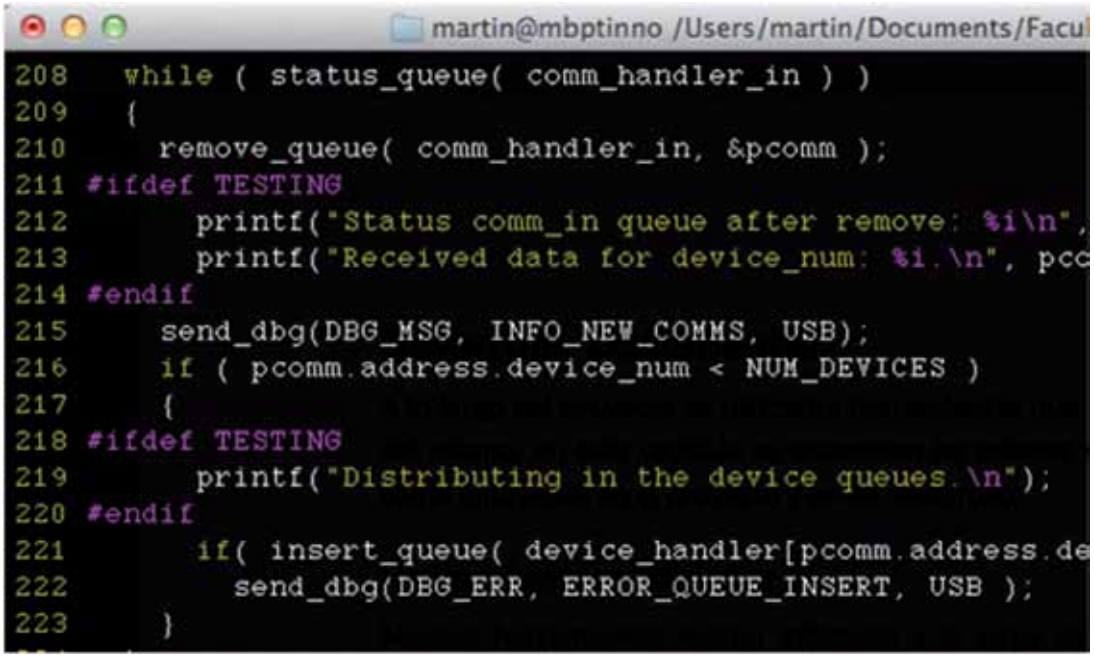
A lo largo del proyecto se utilizaron herramientas que ayudaron a la documentación y desarrollo del mismo, en éste capitulo se enumeran las mismas y se da una breve descripción de su uso y como influyeron en el proyecto y en mi desarrollo.

Herramientas utilizadas

Ya sea desde recurrir a un recurso en Internet o herramientas propias de los sistemas operativos las herramientas más utilizadas y que considero fundamentales fueron:

VIM[24]

Es una versión mejorada del editor de texto VI incluido en la mayoría de los sistemas operativos UNIX que si bien tiene un aspecto simple oculta una gran herramienta a la hora de trabajar con texto como es el caso de la programación. Toda la programación del firmware del proyecto lo realicé enteramente en VIM a excepción del código para el integrado CC2530 que fue necesario hacerlo en IAR Workbench.

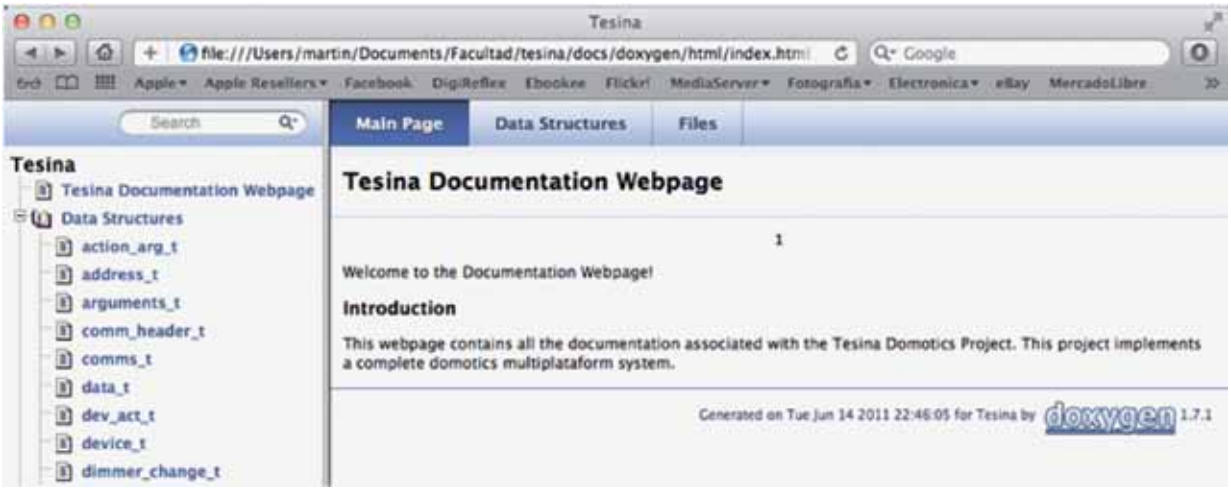


DOXYGEN[25]

Desde el comienzo del proyecto mi tutor hizo hincapié en la necesidad de contar con un código bien documentando. Ante esta necesidad DOXYGEN un sistema de documentación para código fuente era la herramienta ideal.

Doxygen es un software libre y gratuito que permite generar la documentación del código a medida que se escribe el mismo, la documentación puede ser luego exportada en formato web, PDF entre otras. Una de las grandes ventajas es que a la hora de exportar en formato Web pude contar con mi propio sistema para realizar búsquedas y consultar sobre cada tipo de dato o función que definí dentro de mi sistema.

24. http://www.vim.org/
25. http://www.doxygen.org/



SVN[26]

Al tratarse de un proyecto de firmware era necesario contar con un sistema de versiones para el código. Para ello recurrí a SVN que es de código libre y los existen infinidad de clientes multiplataforma.

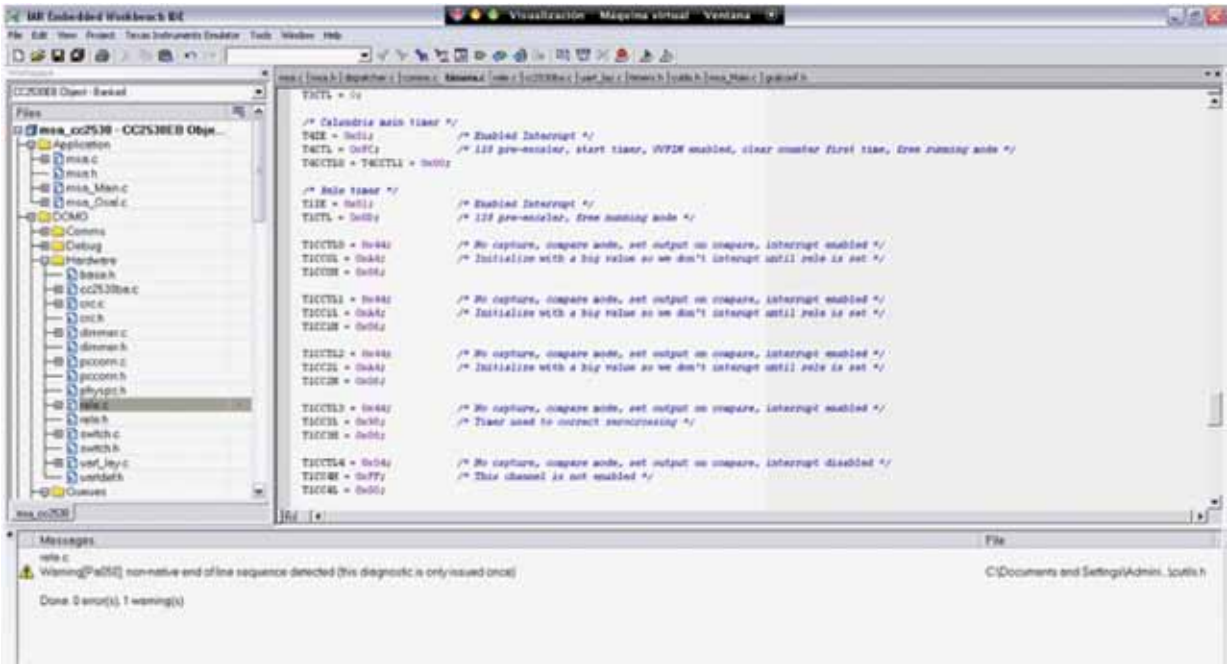
Si bien había utilizado otros sistemas de versiones en el pasado en este proyecto tuve que,

- Descargar, instalar y configurar un servidor SVN desde cero basándome en la documentación existente.
- Configurar el servidor para que pueda ser accedido desde cualquier conexión a internet.
- Configurar los distintos clientes que utilice en los distintos sistemas operativos:
  - TortoiseSVN en Windows.
  - CornerStone en Mac OS X.
  - Cliente nativo SVN en Linux.

IAR Workbench[27]

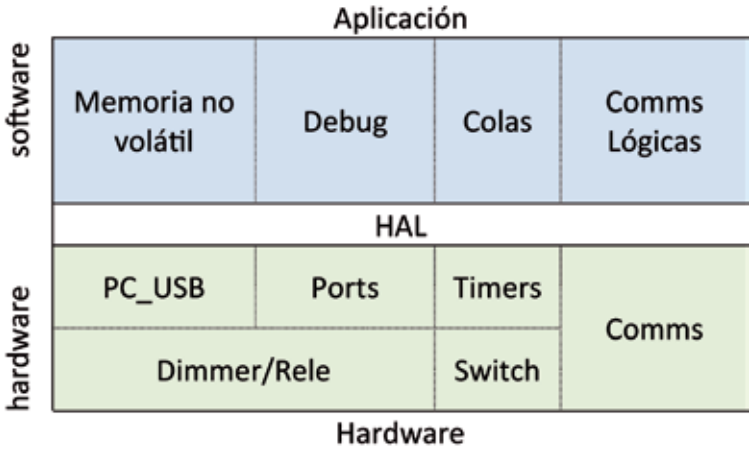
Para compilar el proyecto y grabarlo en el microcontrolador fue necesario utilizar este IDE de la empresa IAR. Más allá que como IDE deja mucho que desear no pude dejar de utilizarlo porque es la herramienta recomendada por Texas Instruments para trabajar con el microcontrolador elegido.

26. http://subversion.tigris.org/
27. http://www.iar.com/website1/1.0.1.0/50/1/



Pruebas y resultados

Luego de concluida la etapa de Diseño y durante el desarrollo de la Implementación fue necesario realizar las pruebas de cada componente del proyecto individualmente. Esto fue de gran importancia y un paso más que necesario debido a no contar con el hardware dedicado que sería la base para correr el sistema desde el comienzo del proyecto, el sistema ya estaba siendo probado en otras plataformas, inicialmente bajo un sistema operativo Unix.



Cada uno de los componentes diagramados lógicamente en el cuadro anterior fueron testeados. Si se inspecciona el código pueden notarse en alguno de sus archivos segmentos de código entre ordenes de pre-procesador para realizar las pruebas de cada componente por separado. Es decir que cuando me interesaba probar una parte en especial del código se agregaban dichas ordenes al makefile del proyecto y así aislaba secciones de código. En el ejemplo siguiente se testeaba el código relacionado al dimmer.

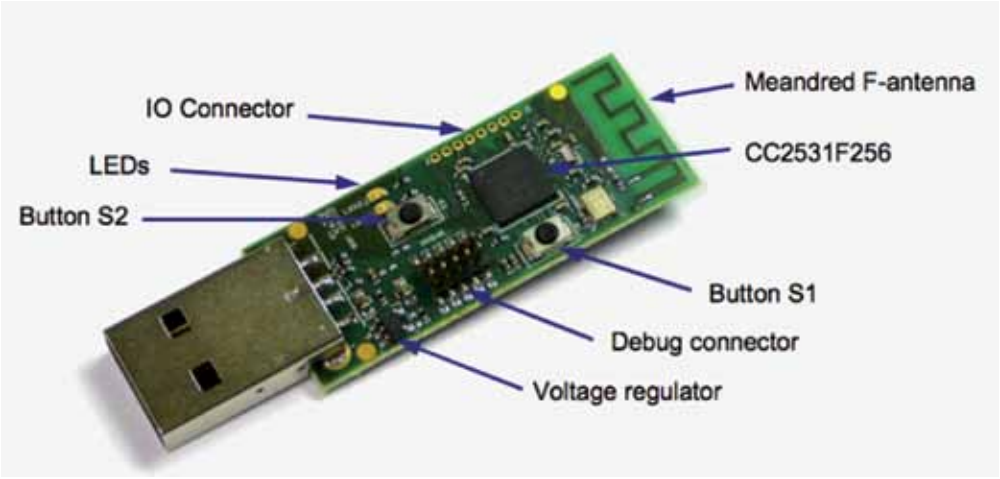
```
280 #ifdef TESTING_DIMMER
281 int
282 main()
283 {
284   connected_dev[0] = DEV_SWITCH;
285   connected_dev[1] = DEV_SWITCH;
286   connected_dev[2] = DEV_SWITCH;
287
288   start_ports();
289   init_timers();
290
291   HAL_ENABLE_INTERRUPTS();
292
293   for(;;);
294     do_domotics();
295 }
296 #endif
```

Con respecto al hardware hay que hacer una aclaración ya que al no contar con el mismo, en especial el microcontrolador CC2530 utilizado, desde el comienzo del proyecto fue necesario realizar las pruebas en PC mediante una herramienta que fuera de interfaz entre el firmware y el hardware (dimmer, switch y rele) . Para tal propósito desarrollé una placa USB llamada USB_INTERFACE que fue de gran ayuda para esa tarea pero al tratarse de un desarrollo más que nada relacionado con el Hardware no será tratado en esta tesina. Ver Anexo B para imágenes de referencia.

En cuanto a las comunicaciones mientras desarrollaba esta capa debí utilizar herramientas que me permitieran realizar capturas de los paquetes enviados por la red para corroborar que la información viajara de manera correcta para lograrlo utilicé,

- En el desarrollo de UDP utilicé WireShark[28] una herramienta de código libre de amplio conocimiento. Con la misma fui capaz de seguir la comunicación a través de la red tanto cableada como inalámbrica (Wi-Fi) .
- Para TiMAC utilicé una herramienta gratuita provista por Texas Instruments llamada SmartRF Protocol Packet Sniffer[29] capaz de capturar paquetes 802.15.4 y mostrar su contenido mediante la utilización de un microcontrolador CC2531.

Una imagen[30] de microcontrolador CC2531 diseñado por TI para utilizar en conjunto con el software de escañero de redes 802.15.4 a continuación

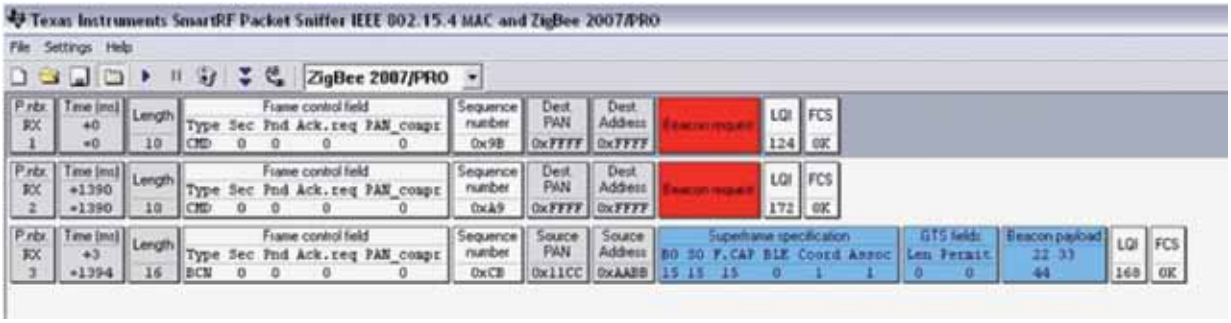


28. http://www.wireshark.org/
29. http://www.ti.com/tool/packet-sniffer
30. Imagen extraída de http://www.ti.com/lit/ug/swru208b/swru208b.pdf

Dongle CC2531

Gracias al Packet Sniffer de TI puede ser capaz de verificar la comunicación 802.15.4 entre los distintos dispositivos (MBUs), la captura debajo muestra un simple ejemplo de la comunicación a nivel de enlace.

1. Primer paquete, un MBU_A inicializa y busca en todas las redes si existe algún coordinador. Como no recibe respuesta el mismo inicia como coordinador.
2. Un segundo MBU_B inicia y envía un paquete a todas las redes.
3. MBU_A responde indicándole de su existencia y responde con la información de la red.



Todas estas herramientas y métodos de desarrollo me permitieron realizar las pruebas incluso antes de completar todo el proyecto. En muchas ocasiones dichas pruebas hicieron que tenga que volver a la hoja de diseño para solucionar problemas encontrados que necesitaban ser solucionados pero que fueron de una gran experiencia a nivel académico y profesional.

Concluida la etapa de Implementación tanto de firmware/software como la de hardware fue necesario realizar pruebas de todo el conjunto. De la misma manera las herramientas utilizadas para pruebas individuales fueron de gran ayuda para detectar errores en la implementación pero finalmente fui capaz de cumplir con los objetivos y limitaciones de la tesina que resulta en un prototipo funcional (MBU_Dev).

Las resultados demuestran que una vez configurado cada MBU_Dev con los distintos dispositivos conectados y cargados los scripts de usuario las comunicaciones y acciones funcionan como se espera.

Conclusiones

Poder haber concluido con el proyecto y cumplir con los objetivos previstos representa una satisfacción personal muy grande. A lo largo del diseño, desarrollo y puesta en marcha me enfrente a diferentes problemas que fueron resolviéndose ya se con la ayuda de mi tutor o diferentes ingenieros, de todos ellos he ganado una experiencia enriquecedora enorme tanto a nivel académico como ingenieril.

Las limitaciones del proyecto son muy pocas, el mismo puede crecer mucho más pero a fines prácticos y para el desarrollo de esta tesina era necesario acotarlo a las bases mínimas que se han expuesto a lo largo de este trabajo.

La tesina aquí expuesta sólo representa un leve porcentaje de todo el trabajo realizado, muchos detalles más pueden evidenciarse al utilizar el prototipo funcional MBU_Dev y descubrir su potencial.

A continuación se presentan las limitaciones actuales del proyecto y sus posibles soluciones y/o mejoras a futuro.

Limitaciones de la solución planteada

Como se mencionó en la sección de objetivos, uno de ellos era sentar las bases para que el sistema pudiera crecer y/o adaptarse a nuevos requerimientos, la solución planteada dista de ser perfecta pero tiene el potencial de crecer para cumplirlos.

Las principales limitaciones actuales son:

- La topología de red no es mesh, con las consiguientes ventajas y desventajas enunciadas en la sección de Diseño.
- Utilizando los scripts disponibles no es posible utilizar condicionales para tomar acciones al registrarse un evento.
- Reducida diversidad de dispositivos entrada/salida.

Todas ellas pueden ser solucionadas a futuro avanzando en el desarrollo del proyecto, a tal fin en la siguiente sección se enuncia una lista posible de posible mejoras.

Líneas futuras de desarrollo

Muchas de las líneas de desarrollo a futuro tienden a complementar las limitaciones actuales del proyecto, otras buscarán ampliar la gamas de usuarios del mismo y dotar de nuevas características al mismo.

Lo primero que habría que tener en cuenta es proveer al sistema de nuevos dispositivos de entrada/salida para enriquecer las acciones que el usuario tendría para controlar/usar. Algunas opciones serían:

Entrada

Sensor de temperatura: Con este sensor se podrían tomar acciones como controlar un sistema de refrigeración y/o calefacción según

Lector de RFID[31]/NFC[32]: Estos tipos de lectores podrían ser utilizados para identificar a distintos usuarios del mismo recinto para tomar distintas acciones desde abrir puertas a bloquear ciertos sectores del sistema.

Sensores on/off: Todo tipo de sensores que tengan dos estados, sensores de humo, gas, movimiento, luz, etc.

Salida

Infrarrojo: Mediante una salida por infrarrojo podría lograrse una especie de control universal para controlar otros dispositivos ya se prender, apagar, cambiar de canal u otras opciones.

Interfaz de audio: Una salida de audio permitiría enviar alertas de sonido o enviar streaming (flujo de datos) de música.

Un área muy importante para el desarrollo del proyecto sería dotar al sistema de distintas interfaces por la cuales puedan tomarse acciones y/o comprobar estados de los distintos dispositivos ya se por medio de dispositivos móviles como teléfonos inteligentes, PDA, Tablets y PCs. Se deberían desarrollar interfaces para:

PC ⟶ POSIX, Windows.

Dispositivos Móviles ⟶ Android, iOS, Blackberry, Windows Mobile, Symbian.

Para lograr dichas interfaces y favorecer al desarrollo del sistema sería conveniente crear una API (Application Programmer Interface) con métodos de acceso y comunicación abiertos para que desarrolladores de más alto nivel pueden generar dichas interfaces. Esta API sería una forma de interface entre programadores de alto nivel que quisieran acceder al sistema pero no necesariamente tienen que tener los conocimientos de bajo nivel implementados por el firmware.

En la actualidad existe una fuerte tendencia a utilizar Arduino[33], muchos entusiastas de la electrónica y/o informática se verían beneficiados si el proyecto se portara para funcionar en dicha plataforma. A su vez también podrían diseñarse interfaces con otros sistemas de domótica/control como X10[34] para abarcar más usuarios.

Durante el desarrollo de la sección IDE y configuración del MBU no se consideró que sucedería si uno usuario quisiera modificar los scripts una vez ya configurado el MBU. Esto implicaría que el usuario debería volver a conectar cada MBU al IDE para su reconfiguración, si bien esto no parece una complicación, si lo sería una vez que cada MBU haya sido empotrado en su ubicación final. Esto podría solucionarse implementando un nuevo tipo de encabezado (header) para los paquetes (packet_t) que identifique al paquete como de configuración y que el espacio de scripts sea actualizado con la nueva información.

Teniendo en cuenta que la reconfiguración de un MBU podría aprovecharse a fines maliciosos, otro aspecto importante a desarrollar sería implementar métodos de comunicación segura entre los distintos componentes del sistema durante las comunicaciones inalámbricas, ya que estas son más propensas a ataques por sus propias características. La seguridad podría ser implementada gracias a que el mismo microcontrolador CC2530 cuenta con un coprocesador de encriptación/des-encriptación Advance Encription Standard (AES) para facilitar dicha tarea.

31. http://www.rfid.org/ Definición Wikipedia: RFID (siglas de Radio Frequency IDentification, en español identificación por radiofrecuencia) es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas tags RFID. El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio.

32. http://www.nfc-forum.org/home/ Definición Wikipedia: NFC son las siglas en inglés de Near Field Communication (NFC), una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos a menos de 10cm. Es una simple extensión del estándar ISO 14443 (RFID).

33. http://www.arduino.cc/ Arduino es una plataforma de código abierto basado en prototipos de electrónica flexible y fácil de utilizar del hardware y software. Está pensado para artistas, diseñadores, aficionados y cualquier persona interesada en la creación de objetos interactivos ambientes.

34. http://www.x10.com/ X10 es un estándar internacional y abierto para la comunicación entre los dispositivos electrónicos utilizados para la domótica. Se utiliza sobre todo el cableado de alimentación de línea de señalización y control, donde las señales incluyen breves ráfagas de frecuencia de radio que representa la información digital.

Si bien dentro de los scripts personalizables se puede especificar cuanto tiempo ha de esperar el sistema para realizar una acción luego de recibir un evento sería conveniente incluir un servidor de tareas o planificador al cual se le podrían definir distintas acciones a realizar basadas en fechas u horarios para cumplir con acciones repetitivas o que quieran programarse a futuro. Este planificador debería tener una interfaz, por ejemplo web, por la cual el usuario fuera capaz de definir las acciones a realizar y/o cancelar las pendientes.

Bibliografía

Mucha de la documentación aquí expuesta fue utilizada para el diseño e implementación del proyecto. Bibliografía adicional como White Papers o Notas de Aplicación de distintos fabricantes no ha sido incluidas en esta sección pero se ha hecho referencia a ellos como otros.

- Zigbee Wireless Networking – Drew Gislason.
- Zigbee vision for the Home – Zigbee Alliance.
- Hands-on Zigbee, Implementing 802.15.4 with Microcontrollers – Zigbee Alliance.
- Teoría de circuitos – Boylestad y Nashelsky.
- Otros – Más detalles en Anexo A.

Anexo A – Firmware

Detalles técnicos del código

Ambas capas software y hardware presentan distintos servicios que el sistema usa para su funcionamiento, para conocer más detalles de los mismos el código se documentó utilizando Doxygen y toda la información está disponible en versión web[35].

En el sitio web se puede buscar por los siguientes headers que contienen la descripción de cada método.

Servicios	
Software	Hardware
osal.h	base.h
debug.h	rele.h
queue.h	switch.h
comms.h	dimmer.h
script.h	timers.h
exefuncs.h	physcomms.h

Además de los métodos también pueden buscarse definiciones de variables, enumeraciones y estructuras utilizadas en el proyecto.

Documentación técnica adicional

Muchos documentos técnicos fueron utilizados para la confección del proyecto entre ellos puede listar los más importantes:

- Documentación del microcontrolador, necesario para la implementación del firmware. Fue necesario conocer los espacios de memoria disponibles así como los detalles necesarios para la inicialización del ambiente y asignación de puertos en la capas inferiores de hardware. http://www.ti.com/lit/ug/swru191b/swru191b.pdf
- Documentación de herramientas utilizadas. En este caso hago referencia a herramientas como:
  - o Doxygen http://www.stack.nl/~dimitri/doxygen/manual.html
  - o confuse http://www.nongnu.org/confuse/manual/index.html
  - o ncurses http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/
  - o libFTDI http://www.intra2net.com/en/developer/libftdi/documentation/
- Documentación provista por Texas Instruments
  - o Soluciones wireless http://www.ti.com/lit/sg/slab056/slab056.pdf
  - o TiMAC http://www.ti.com/tool/timac

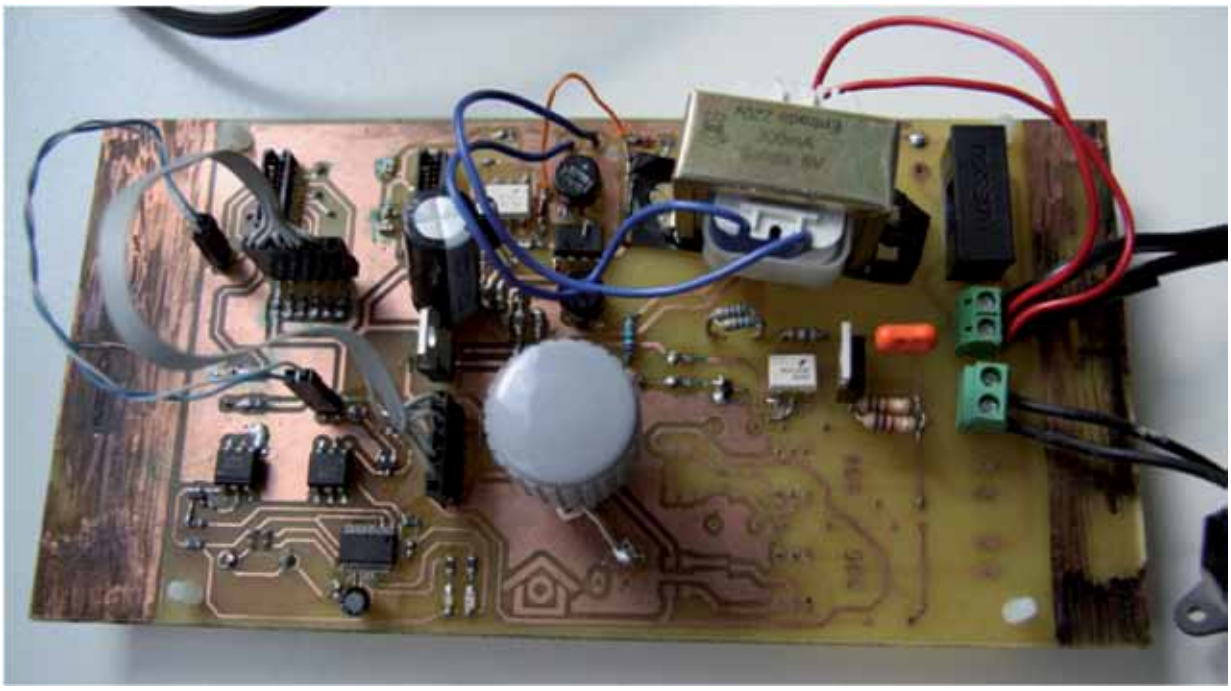
35. Durante la defensa de la tesina se presentará el sitio web de manera local donde se podrá ver los contenidos del mismo.

Anexo B – Hardware

En esta sección sólo a modo de referencia se agregan imágenes de algunos de los dispositivos de hardware creados para la tesina.

MBU_Dev

Prototipo de MBU_Dev versión 3.1 funcional pero no con todos sus componentes.



USB_INTERFACE

A continuación la primera imagen es de la placa base de USB_INTERFACE mientras que la segunda es la misma placa una vez armada junto a sus conexiones.