UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE COMUNICACIÓN INALÁMBRICO PARA AUTOMATIZACIÓN Y MONITOREO

TESIS

Que para obtener el título de
Ingeniero Eléctrico Y Electrónico

PRESENTAN:

Diana María Osorio Londoño
y
Armando Delgado del Río

DIRECTOR DE TESIS:

M.I. Antonio Salvá Calleja

Ciudad Universitaria, marzo de 2016

Dedicatoria

A Margarita, mi madre, por haber cuidado de mi hasta en los momentos más difíciles.

Armando

A mis padres, Gloria y Gabriel, mi ejemplo a seguir y mi apoyo incondicional, en cada momento de mi vida.

Diana

Agradecimientos

A mi madre, que me enseñó que trabajando con perseverancia y disciplina se pueden alcanzar todas las metas propuestas, así como superar todos los obstáculos que en la vida se presenten.

A mis hermanas, Diana y Denisse, que con su trabajo me permitieron continuar con mis estudios y desarrollarme como profesional.

Armando

A mis padres, que con su fortaleza, dedicación y constancia, me impulsan a seguir mis metas y son el apoyo incondicional que me permitió llegar a ser lo que soy.

A mi familia, por estar a mi lado y brindarme el cariño y el soporte que me ayudó a desarrollarme como persona y como profesional.

Diana

A nuestro tutor de tesis, el M. Antonio Salvá Calleja, que con su talento y sabiduría nos asesoró y nos proporcionó las herramientas para el desarrollo y culminación de este trabajo.

A la Facultad de Ingeniería de la UNAM, por brindarnos la educación de excelencia para constituirnos como profesionales en servicio de nosotros mismos, así como de la comunidad en la que vivimos.

Ambos.

RESUMEN

En este trabajo de tesis se diseñó un sistema de comunicaciones inalámbrico para automatización y monitoreo (CIAM), orientado a la automatización de procesos y monitoreo de variables de interés en diversas aplicaciones. Este sistema está basado en el protocolo de comunicaciones TCP, que garantiza la transmisión de datos sin errores y en el orden en el que fueron enviados. El sistema CIAM está diseñado para intercambiar datos entre microcontroladores, computadores personales y dispositivos móviles inteligentes. Se hace uso de un dispositivo de red para crear y administrar una red de área local, a la cual los dispositivos deben conectarse.

Se diseñó un programa para el sistema operativo Windows, programado en el lenguaje de alto nivel Object Pascal, que habilita un servidor de comunicaciones basado en el protocolo de transporte TCP. Este servidor es el proceso que maneja todas las conexiones y el intermediario que usan los demás dispositivos para comunicarse entre sí. Los dispositivos móviles inteligentes y las computadoras personales cuyo sistema operativo sea Windows o Android, se conectan al servidor TCP como clientes, mediante programas igualmente programados en Object Pascal. El servidor y los clientes hacen uso de los componentes del proyecto InDy, el cual es multiplataforma, multilenguaje y de código abierto.

Los microcontroladores se comunican directamente con un módulo que utiliza la tecnología Wifi, que a su vez se conecta de manera inalámbrica como cliente al servidor TCP. Para ello, se diseñó una tarjeta de desarrollo que incluye una fuente de poder integrada, un microcontrolador de propósito general, y el módulo Wifi. Esta tarjeta de desarrollo está basada en la serie de tarjetas MINICON_XX y hace uso del compilador cruzado de BASIC MINIBAS8A, ambos diseñados en el Departamento de Control y Robótica de esta facultad, por el M.I. Antonio Salvá Calleja.

En esta plataforma se diseñó una biblioteca de subrutinas para que el microcontrolador comande al módulo Wifi y habilite un modo de transmisión transparente, para que este último se comporte como un cable serial virtual. De esta manera se le puede agregar comunicación inalámbrica a cualquier proyecto basado en microcontrolador, que sea supervisado por el puerto serial, simplemente utilizando las rutinas de la biblioteca y sustituyendo la interfaz serial por el módulo Wifi. Finalmente, se realizaron tres ejemplos de aplicación, uno de electrónica de potencia, otro orientado a domótica y uno más de instrumentación virtual, que hacen uso del sistema CIAM.

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Incontables proyectos basados en microcontrolador hacen uso de la comunicación serial para intercambiar información. El uso de sistemas cableados es uno de los mayores inconvenientes al momento de implementar estos proyectos, pues el acceso físico es limitado. Además, ocasiona dificultad de mantenimiento y escasa posibilidad de expansión o crecimiento del proyecto. En este sentido, hemos intentado sufragar estas carencias creando un sistema que sustituya el cableado por un método de comunicación inalámbrico, compatible con cualquier proyecto existente que utilice la comunicación serial como método de interacción.

Muchos de estos proyectos están orientados al incremento de la eficiencia en los distintos procesos y tareas, tanto en la vida cotidiana de las personas como a nivel industrial o el campo de la investigación científica. Las comunicaciones inalámbricas han representado el aumento de la versatilidad y movilidad en todas estas áreas, ya que han permitido expandir las capacidades de los sistemas, hacerlos más rápidos y cubrir grandes distancias sin aumentar exageradamente el costo, como sucedería al intentar realizarlo con tecnología cableada.

La tecnología que ha permitido esta transición requiere de dispositivos cada vez más eficientes, tanto que hoy en día existen empresas y centros universitarios dedicadas al desarrollo de tecnología de punta orientada a satisfacer esta necesidad.

En México también existe una importante incursión en los campos de las telecomunicaciones y la automatización de procesos industriales, por lo que es indispensable continuar con esta labor de desarrollar tecnología que incremente la versatilidad, rapidez y eficacia de los sistemas industriales.

Dentro de este contexto, la idea de límite se hace cada vez menos relevante, pues actualmente ya se habla de conceptos como el Internet de las cosas. La capacidad de conectar cualquier objeto a la red de Internet y administrarlo desde cualquier lugar del mundo, ya sea un dispositivo, un sistema o un servicio, ha ampliado los ambientes en los que las personas nos desarrollamos, a un entorno global, mediante el uso de la red de redes.

Los sistemas inalámbricos han revolucionado la versatilidad con la que se controla un robot industrial, se monitorean procesos o se gestionan las tareas del hogar. Valiéndose de sistemas embebidos, o hardware especializado que le permite no solo la conectividad a la red de Internet, sino que además programa eventos específicos en función de las tareas que le sean dictadas remotamente.

Tomando lo anterior en consideración, en el presente trabajo se describe el diseño de un sistema de comunicación inalámbrico, el cual permite el intercambio de información entre microcontroladores y computadoras personales o dispositivos móviles inteligentes. Esto con la finalidad de facilitar la automatización de algunos procesos y el monitoreo de variables de interés en tiempo real y de manera inalámbrica.

Estado del arte de las tecnologías inalámbricas

A continuación, se enlistan algunas de las tecnologías inalámbricas más utilizadas actualmente, clasificadas según su alcance.

- Tecnologías de corto alcance:
  - *Bluetooth*. Es una especificación regulada por la norma IEEE 802.15.1, que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda de los 2.4 [GHz]. El alcance de esta tecnología es de aproximadamente 10 [m], debido a que ha diseñada ahorrar en el consumo de energía.
  - *DECT (digital enhanced cordless telecommunications).* Es un estándar para teléfonos inalámbricos digitales, comúnmente utilizado para propósitos domésticos o corporativos. Admite la transferencia de datos a 2[Mbps], y tiene un alcance desde los 25 hasta los 100 [m].
  - *IrDa (Infrared Data Association).* Es un estándar que utiliza el espectro de frecuencia de infrarrojo para transmitir información. Su uso se vio expandido gracias a su bajo costo de implementación y bajo consumo de energía. Requiere de la alineación del dispositivo emisor con el receptor, con una desviación permitida hasta los 30°.
  - *NFC (Near field communication).* Permite la transmisión de datos mediante un enlace de radiofrecuencia en la banda de 13.56 [MHz]. Requiere que los dispositivos estén muy próximos entre sí, a menos de 20 [cm].
  - *ZigBee.* Es una especificación de protocolos de alto nivel para su uso con radiodifusión digital de baja tasa de envío de datos y maximización del consumo energético. Está regulado por el grupo de trabajo IEEE 802.15.4.
- Tecnologías de medio alcance:
  - *Wifi*. Es un mecanismo de conexión de dispositivos, que garantiza el cumplimiento de las normas IEEE 802.11, relacionadas a redes inalámbricas de área local. Los diversos tipos de *Wifi* se basan en la familia de estándares que engloba las normas antes mencionadas, y son las siguientes:
    - IEEE 802.11b, IEEE 802.11g e IEEE 802.11n, que gozan de la disponibilidad de la banda de 2.4 [GHz], con una rapidez de transmisión de hasta 11 [Mbps], 54 [Mbps] y 300 [Mbps], respectivamente.
    - IEEE 802.11ac, conocido como *Wifi* 5, que opera en la banda de 5 [GHz], recientemente habilitada. Como no existen otras tecnologías utilizando dicha banda, existen muy pocas interferencias.
  - *WiMAX (Worldwide Interoperability for Microwave Access)*. Es una norma de transmisión de datos que utiliza las ondas de radio en las frecuencias de 2.5 a 5.8

[GHz] y puede tener una cobertura de hasta 50 [km]. Es similar a *Wifi*, pero establece puntos de acceso para cubrir un área mayor.

- Tecnologías de largo alcance:
  - *Radio enlace*. Utilizan diferentes bandas del espectro electromagnético, como puede ser la banda de infrarrojos, microondas o láser, para establecer comunicaciones punto a punto o de punto a multipunto.
  - *Satélite*. Cubren grandes superficies y tienen un amplio ancho de banda. Presentan inconvenientes de retardos de propagación de la señal.
  - *3G*. Dispone de banda ancha para telefonía móvil y transmisión de volumen de datos, permitiendo las videoconferencias, descargas de video y televisión en tiempo real.
  - *4G*. Es la integración de red con la tecnología de Internet, dando como resultado una red IP, que se combina con otras tecnologías como *Wifi* y *WiMAX*, para potencial el rendimiento, reduciendo los costos.

Objetivos del proyecto

Diseñar un sistema de comunicación inalámbrico para automatización de procesos y monitoreo en tiempo real de diversas variables de interés, basado en la familia de protocolos TCP/IP, el cual deberá ser fácil de utilizar y ser útil en múltiples campos de aplicación.

Objetivos particulares

1. Investigar las características técnicas de la familia de protocolos TCP/IP.
2. Elaborar un programa que provea servicios TCP y arbitre las comunicaciones entre los dispositivos del sistema.
3. Desarrollar programas multiplataforma que sirvan como interfaz de usuario y se conecten al servidor como clientes TCP.
4. Diseñar y construir una tarjeta de desarrollo que incluya un microcontrolador de propósito general y un módulo de comunicaciones Wifi, y sea capaz de procesar y ejecutar los comandos que le sean enviados desde cualquier cliente conectado al servidor, así como de dar respuesta a dicho cliente
5. Generar una biblioteca de subrutinas que le permita a la tarjeta de desarrollo antes mencionada conectarse al servidor como otro cliente TCP, de manera que esta biblioteca sirva de base para cualquier aplicación a la que el microcontrolador esté destinado.
6. Aplicar el sistema de comunicaciones a ejemplos de aplicación reales.

Motivación

Como parte del trabajo como ingenieros, buscamos aumentar la calidad de vida de los seres humanos, en un mundo que se mueve con gran rapidez y los usuarios requieren de sistemas cada vez más eficaces y versátiles. Asimismo, la tecnología inalámbrica, hoy en día constituye una de las formas más eficientes de desarrollo, además de ser accesible a la mayor parte del sector salud, industrial, doméstico, entre otros.

Por otro lado, mediante este trabajo se pretende mostrar que se puede generar tecnología propia en México, de bajo costo, gracias al diseño e implementación del prototipo de un sistema capaz de facilitar la realización de tareas comunes en espacios habitables y procesos industriales.

Metodología

El sistema de comunicación inalámbrico está integrado por varios bloques funcionales, algunos de los cuales, se diseñaron para fines de la aplicación, otros son dispositivos comerciales tales como teléfonos inteligentes, tabletas electrónicas, módulos de comunicación digital, entre otros.

Para los módulos que se diseñaron para fines del sistema de comunicación inalámbrico de automatización y monitoreo, se empleó software y hardware didáctico diseñado en el Departamento de Control y Robótica de la Facultad de Ingeniería. Además, se diseñaron con herramientas propias para ello, el software ejecutable asociado con la interfaz de usuario requerida por el sistema.

Partiendo del trabajo desarrollado entorno a las tarjetas MINICON_XX, desarrolladas en esta facultad, se buscó llevarlas a un nivel posterior, introduciendo en ellas el software y hardware asociado a la comunicación inalámbrica entre interfaces de usuario y módulos de actuación remota.

CAPÍTULO 1. DESCRIPCIÓN DEL PROYECTO

En el presente capítulo se explica el funcionamiento del proyecto presentado en esta tesis, el cual es un sistema de Comunicación Inalámbrico para Automatización y Monitoreo, que de ahora en adelante será denominado por el acrónimo CIAM. Se describen, de manera general, las partes que lo componen y el papel que desempaña cada una en el sistema. Finalmente, se incluye la descripción de las herramientas auxiliares que se utilizaron para la realización del proyecto.

1.1 Descripción del sistema CIAM

El sistema CIAM hace uso de una red inalámbrica de área local[1], que es creada y mantenida por un dispositivo de red que reúne las cualidades de un router, un switch y un punto de acceso, este tipo de dispositivos es comúnmente denominado como enrutador inalámbrico (*Wireless router*).

Las partes principales de este sistema son:

- Un servidor de comunicaciones basado en el protocolo de transporte de datos TCP, denominado Zero, diseñado para dispositivos con sistema operativo Windows y programado en el lenguaje de alto nivel *Object Pascal*.
- Los programas compatibles con los sistemas operativos Windows y Android, codificados en el lenguaje Object Pascal, que se conectan como clientes al servidor TCP y facilitan la intervención de un usuario humano con el sistema CIAM. Para fines de demostración de las capacidades del sistema CIAM, se implementaron tres de estos programas, los cuales, en conjunto, fueron denominados *clientes comandantes*.
- Las tarjetas de desarrollo, de la serie MINICONW_SH32, cada una incluye una fuente de poder integrada, un chip MC9S08SH32 el cual es un microcontrolador de propósito general, y un módulo wifi ESP8266 que permite, entre otras facilidades, la interfaz de un microcontrolador con una red inalámbrica de área local y su posterior conexión al servidor TCP. Para fines de demostración de las capacidades del sistema, se construyeron tres de estos dispositivos, los cuales, en conjunto, fueron denominados *clientes subordinados*.

Además, se diseñó una biblioteca de subrutinas para el entorno de desarrollo PUMMA_EST, la cual facilita la codificación de programas ejecutables por un chip MC9S08SH32 de manera que este configure a un módulo Wifi ESP8266 a través del su puerto serial asíncrono y que, a su vez, el ESP8266 se comunique, de manera inalámbrica, con un servidor de comunicaciones TCP.

[1] Una red de este estilo también es conocida por el acrónimo del inglés, WLAN (*Wireless Local Area Network*). Para más información sobre redes de dispositivos véase el apéndice A, al final de este documento.

El sistema CIAM está diseñado para dar soporte a múltiples *comandantes* y *subordinados*, que pueden conectarse y ser operados simultáneamente. El número de dispositivos en operación está solamente limitado por las capacidades del hardware en el cual se ejecuta el servidor TCP, pues este último crea un hilo de procesamiento para cada dispositivo conectado al mismo.

Este sistema es capaz de adaptarse a muchos ámbitos de la ingeniería electrónica. Cualquier proyecto basado en microcontrolador, que funcione mediante la comunicación serial, puede adaptarse para hacer uso del sistema CIAM, puesto que la comunicación entre el microcontrolador, incluido en las tarjetas de desarrollo, y el servidor TCP es transparente, como si de un cable serial virtual se tratara.

1.1.1 Funcionamiento.

Tanto los *clientes comandantes* como los *clientes subordinados* se conectan al Servidor Zero, este último maneja, tanto las conexiones individuales de cada cliente, como el intercambio de mensajes entre los mismos. Así pues, el intercambio de información es bastante ágil, ya que cada cliente solo se tiene que preocupar de una única conexión con Zero, el cual atiende cada conexión en un hilo de ejecución paralelo, aprovechando, por ejemplo, las capacidades de los procesadores multinúcleo.

El servidor Zero del sistema CIAM está diseñado para distinguir a los *clientes subordinados* de los *clientes comandantes.* Esta distinción se hace porque los *comandantes* sirven de interfaz para un usuario humano que desee ejecutar alguna acción o monitorear alguna variable, con ayuda del microcontrolador de los *subordinados*. Además, para el humano es importante conocer cuántos dispositivos están conectados a la red para que éste pueda intervenir y tomar decisiones. Zero comparte ésta información sólo con los *comandantes*, pues para los *subordinados* no es tan relevante, ya que estos últimos solo están programados para interpretar los mensajes que reciben como comandos a ejecutar.

Para distinguir entre *comandantes* y *subordinados*, se hace uso de un nombre identificador (o *Nickname*). Zero admite como *Nickname* de un cliente, cualquier combinación de letras del código ASCII de 7 bits, y en el caso de los *subordinados*, todos los identificadores deben llevar el prefijo "sub_". Cabe destacar que el *Nickname* no se puede repetir y tampoco puede ser "Zero" (o cualquier combinación entre mayúsculas y minúsculas de la cadena "Zero"). Es muy recomendable que todo *Nickname* sea corto y de fácil lectura.

Por conveniencia, los *Nickname* escogidos para demostrar el funcionamiento del proyecto de esta tesis, en el caso de los *comandantes* se tomaron del alfabeto radiofónico[2] de la *Western Union*, estos son: Adams, Boston, Chicago, etc. Para el caso de los subordinados, los identificadores se tomaron del alfabeto radiofónico de la OTAN, entre los que se encuentran:

[2] Un alfabeto radiofónico permite la desambiguación de fonemas en la transmisión por radio frecuencia de voz, tanto por los servicios civiles como militares.

Alpha, Bravo, Charlie, Delta, etc. Por lo tanto, algunos ejemplos de identificadores para *subordinados* son: sub_Alpha, sub_Bravo, sub_Charlie, etc.

Para explicar más fácilmente la operación del sistema CIAM, se hará uso de la Figura 1.1, en la que se muestra un escenario típico del funcionamiento del sistema CIAM. En este ejemplo, se considera que están conectados al servidor TCP tres comandantes y tres subordinados.

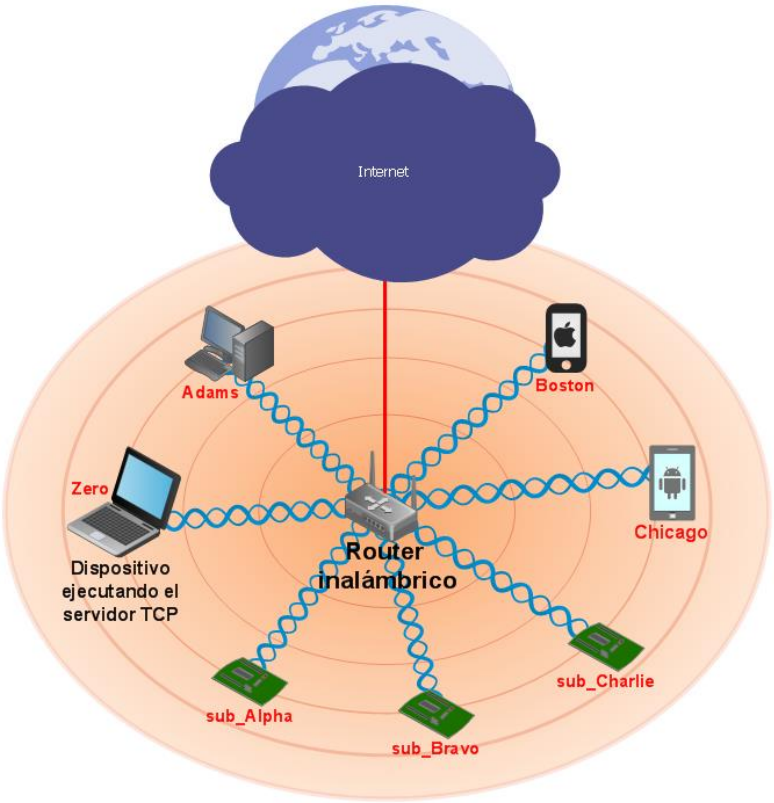


Figura 0.1 Concepto del sistema CIAM

En la Figura 1.1, Zero es un servidor TCP ejecutándose en una computadora personal con sistema operativo Windows. Adams, Boston y Chicago, son dispositivos conectados al servidor TCP como clientes *comandantes*, Adams se ejecuta en una computadora personal, Boston en un teléfono móvil y Chicago en una tableta electrónica. Desde estos dispositivos se envían los comandos a ejecutar por sub_Alpha, sub_Bravo y sub_Charlie, los cuales son dispositivos conectados al servidor TCP como clientes *subordinados*. El router inalámbrico[3] es el encargado de crear la red inalámbrica local, asignar la dirección IP de cada dispositivo, encaminar el tráfico de datos de un punto a otro y conectarse a Internet. Este aparato es indispensable para crear una red de más de dos dispositivos.

Nótese además que la red CIAM tiene salida a Internet, lo cual quiere decir que cualquier dispositivo en el mundo, conectado a Internet, es capaz de solicitar conexión a Zero y hacer uso del sistema CIAM, siempre y cuando, dicho dispositivo conozca tanto la dirección IP

[3] Para más información sobre dispositivos de red véase el apéndice A, al final de este documento.

pública como el puerto del servidor, le enuncie al servidor un *Nickname* válido y respete el *formato de mensajes del sistema CIAM.*

1.1.2 Formato de mensajes del sistema CIAM

Los mensajes que intercambian los dispositivos conectados al sistema son cadenas de caracteres, que deben tener la siguiente estructura:

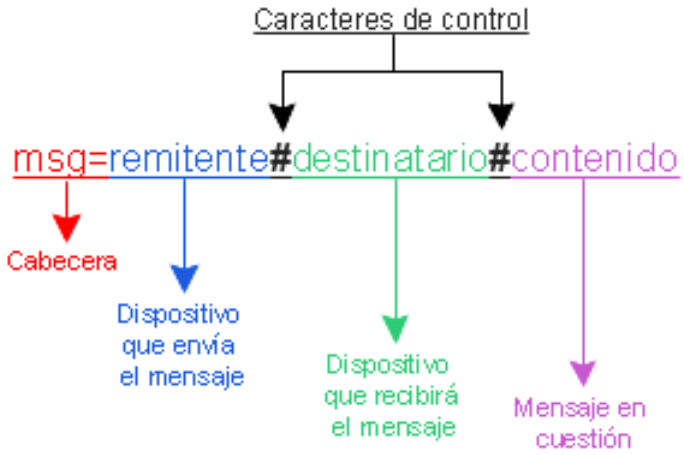


Figura 0.2 Estructura del Formato de mensajes del sistema CIAM

Este formato permite identificar fácilmente el dispositivo que emite un mensaje, el dispositivo al cual está dirigido y la información que desea transmitir. Por ejemplo: supóngase que el *comandante* con *Nickname* "Boston" envía el siguiente mensaje al servidor: "msg=Boston#sub_Alpha#CambiaEdo". Zero recibirá el mensaje, lo procesará e identificará al cliente que lo envió ("Boston") y al cliente al que va dirigido, ("sub_ Alpha") para reenviárselo a este último. Entonces "sub_ Alpha", el cual es un *subordinado*, recibirá el mensaje e identificará tanto el comando (CambiaEdo) como el comandante (Boston). Luego, si "sub_ Alpha" identifica dicho comando como válido, lo ejecutará y responderá con un mensaje al servidor con el siguiente formato: "msg=sub_Alpha#Boston#Cambiado". Finalmente, Zero reenviará la respuesta a Boston y este podrá mostrar algo en la interfaz de usuario para confirmar que el comando se ejecutó satisfactoriamente.

Cabe destacar que el servidor TCP, no soporta a la transmisión de caracteres del código ASCII extendido, por lo que los caracteres especiales, son sustituidos automáticamente por un signo de interrogación. Esto se debe a que la mayoría de las aplicaciones TCP/IP usan código binario [1].

La longitud de la cadena, incluyendo el encabezado, depende de los componentes que participen en el intercambio de información. Tanto el servidor Zero como los clientes comandantes tienen la capacidad de transmitir y procesar cadenas de la longitud de una variable del tipo "AnsiString" (esto es, hasta 2^{31} caracteres), mientras que el módulo Wifi ESP866 tiene la capacidad de transmitir cadenas de hasta 2048 bytes. Sin embargo, si bien el MCU que valida a los clientes subordinados puede enviar por poleo cuantos caracteres

sean necesarios, éste solo tiene un buffer de recepción de hasta 32 caracteres, por lo que esta es la longitud recomendada para los mensajes.

1.2 Descripción de las herramientas auxiliares

En este subtema, se describirán las herramientas de programación que se usaron para desarrollar el software del servidor TCP, así como para el software de los clientes *comandantes* y de los clientes *subordinados*. Asimismo, se describe el hardware utilizado para crear la red inalámbrica mediante la cual están interconectados los dispositivos que conforman el sistema CIAM.

1.2.1 Ambiente de desarrollo integrado PUMMA_EST.

El ambiente de desarrollo integrado PUMMA_EST es un software capaz de generar código máquina ejecutable en dispositivos CHIPBAS8, a partir de un código en el lenguaje BASIC o ensamblador, ya que cuenta con los ensambladores cruzados ENS08 y ENS_430, además de un compilador cruzado de BASIC, denominado MINIBAS8A. El concepto de CHIPBAS8 se describe con mayor profundidad en [2]. El PUMMA_EST es ejecutable en sistemas operativos Microsoft® Windows 98/M/XP/V/7/8/8.1/10 y fue diseñado en el departamento de Control y Robótica de la Facultad de Ingeniería de la UNAM. El software está integrado por varias ventanas, entre las que destacan:

- La ventana de edición, en la cual se escribe y almacena el código fuente de los programas ejecutables por el MCU. Esta ventana incluye los botones para iniciar la compilación y ejecución de los programas, realizar el borrado de la memoria FLASH y para el manejo de archivos (véase Figura 1.3).

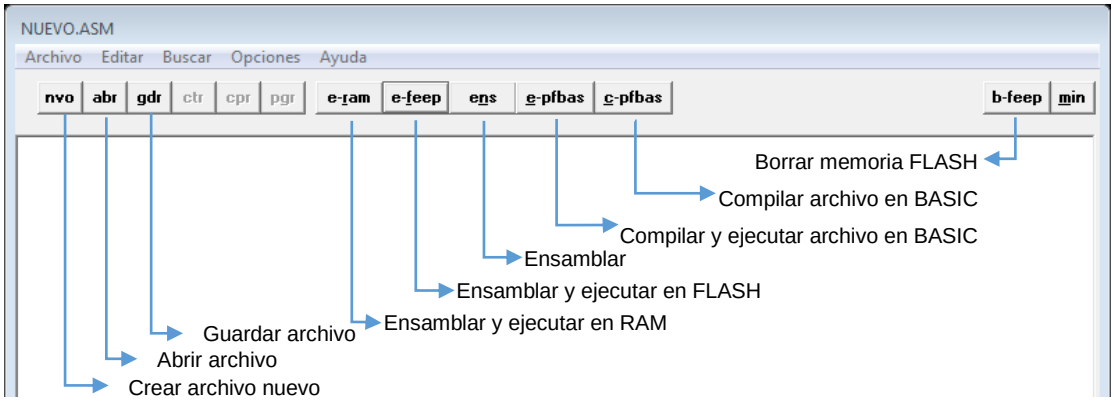


Figura 0.3 Ventana de edición del ambiente PUMMA_EST

- La ventana de manejo hexadecimal, en esta interfaz se pueden leer segmentos 256 bytes de la memoria del MCU, examinar una localidad de memoria, escribir uno o más bytes, e incluso, cargar y ejecutar en memoria FLASH archivos con el formato S19 generados en otro ambiente de programación (como por ejemplo el entorno CodeWarrior). En la figura 1.4 se muestra un posible aspecto de la ventana de manejo hexadecimal del software manejador PUMMA_EST.

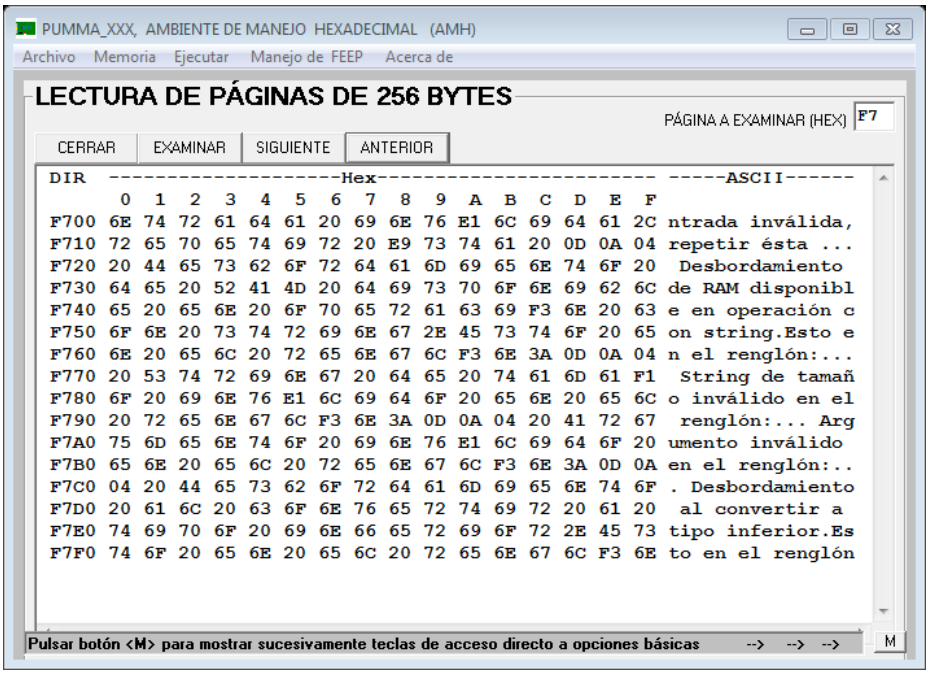


Figura 0.4 Ventana del manejador hexadecimal

- La ventana del emulador de terminal serial, que permite interactuar con el puerto serie de la computadora, se utiliza para validar la ejecución de programas en el MCU, enviar datos a éste, o bien leerlos. En la figura 1.5 se muestra la ventana del emulador de terminal del PUMMA_EST.

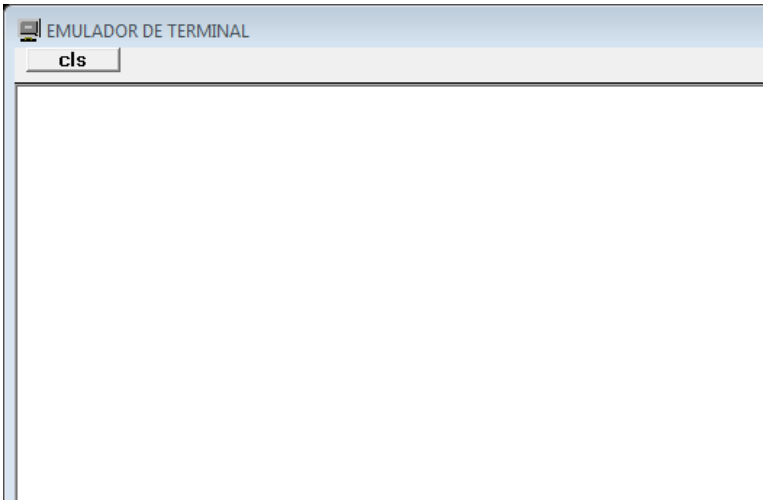


Figura 0.5 Ventana del emulador de terminal del PUMMA_EST

Cabe mencionar que, el compilador cruzado presente en el PUMMA_EST soporta la directiva *#include*, utilizada en este trabajo como una herramienta para la inclusión de archivos que contienen código predefinido, por ejemplo, subrutinas que pueden ser utilizadas dentro del código fuente de los programas escritos en este entorno, sin declararlas explícitamente.

El enlace entre el MCU y el manejador PUMMA_EST se establece a través del puerto serial, como las computadoras modernas no cuentan con este puerto, se utiliza un dispositivo que valida un puerto serial virtual en el puerto USB de la computadora. Este dispositivo es el circuito convertidor USB-SERIE III con llave para MINIBAS8A, el cual permite comunicar un dispositivo serie asíncrono con el puerto USB de una PC (véase Figura 1.6).

Figura 0.6 Fotografía del convertidor USB-SERIE III con llave para MINIBAS8A

El código fuente de la ventana de edición es compilado y posteriormente grabado en la memoria del MCU a través del enlace serial asíncrono, sin la necesidad de contar con costosos dispositivos de grabación de chips, gracias a un núcleo de comunicaciones integrado en el firmware de base almacenado en la memoria no volátil del MCU.

El ambiente de desarrollo PUMMA_EST, el compilador cruzado de BASIC MINIBAS8A, el software de base que incluyen los dispositivos CHIPBAS8 y el convertidor USB-SERIE III con llave para MINIBAS8A, fueron diseñados en el Departamento de Control y Robótica de la Facultad Ingeniería de la UNAM. El manual del PUMMA_EST encuentra en [3].

1.2.2 El entorno de desarrollo RAD Studio XE8

RAD Studio XE8 es la vigésimo segunda versión del entorno RAD Studio, el cual es un producto de Embarcadero Technologies y se trata de un ambiente de desarrollo integrado que contiene las herramientas: Delphi Builder, C++ Builder y HTML5 Builder. El acrónimo RAD proviene del inglés, *Rapid Application Development* (desarrollo de aplicaciones rápido) el cual hace énfasis en el enfoque de esta suite.

Según la página del desarrollador, "*RAD Studio® XE8 de Embarcadero® es la manera más rápida de crear y actualizar aplicaciones visuales atractivas, hiperconectadas y ricas en datos para Windows, Mac, Mobile, y más, usando Object Pascal y C++*" [4].

Para el desarrollo del servidor TCP, así como para los clientes comandantes, se ha utilizado la herramienta *Delphi Builder* del entorno RAD Studio XE8. Delphi es una herramienta de desarrollo y programación que utiliza el lenguaje Object Pascal, el cual es la versión orientada a objetos del antiguo Pascal. Es producido por la empresa estadounidense CodeGear,

adquirida en mayo de 2008 por Embarcadero Technologies de la mano de Borland Software Corporation. En sus diferentes variantes, permite producir archivos ejecutables para Windows, MacOS X, iOS, Android, GNU/Linux y la plataforma Microsoft .NET.

El entorno RAD Studio XE8 tiene la ventaja de incluir dos marcos de trabajo, el marco VCL, (del inglés, Visual Component Library) y el marco FMX (Firemonkey). Ambos marcos de trabajo facilitan la creación de interfaces de usuario y el desarrollo de programas bajo el concepto de componente: propiedades, métodos y eventos. Se diferencian en que VCL está estructuralmente y visualmente sincronizado con Windows, pues el aspecto de las ventanas, es similar a las de dicho sistema operativo, mientras que FMX está orientado al diseño de programas multidispositivo, con soporte para los sistemas Windows, MacOS X, iOS y Android. Para el desarrollo de programas exclusivos del sistema operativo Windows, el marco VCL es más eficiente que FMX. En la figura 1.7 se muestra el aspecto de la ventana, propia del entorno RAD Studio XE8, para el desarrollo de aplicaciones.

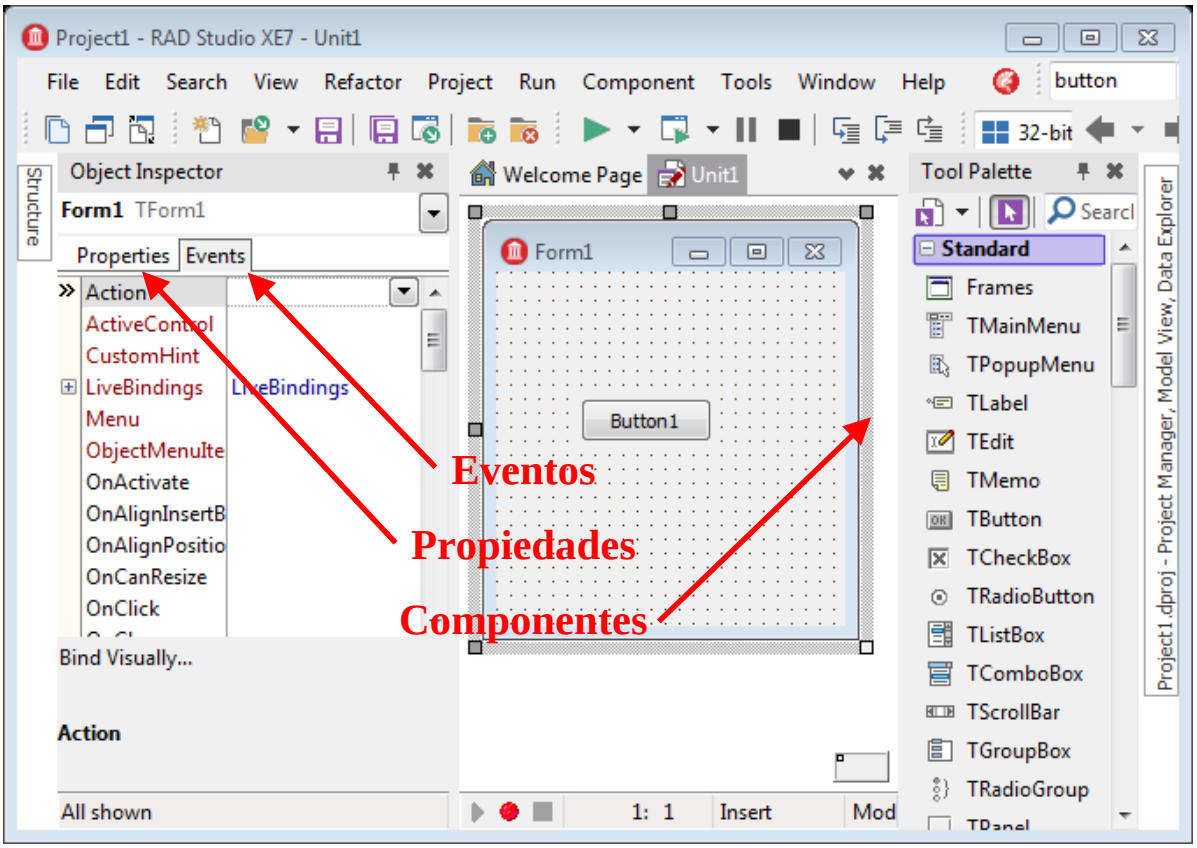


Figura 0.7 Ventana del entorno RAD Studio

El uso de componentes[4], permite acelerar la creación de aplicaciones, y hacen que el entorno sea sencillo, visual y cómodo. Se diferencian dos tipos de componentes:

[4] Un componte es una clase de objetos predeterminada con propiedades, métodos y eventos predefinidos.

1. Componentes visuales: Entre los visuales se distinguen los botones, las etiquetas y demás elementos visibles en la ventana de la interfaz de usuario.
2. Componentes no visuales: Entre los no visuales están los temporizadores, los *sockets de Internet*[5] y demás elementos sin interfaz gráfica. Los componentes no visuales no son visibles en la ventana del programa cuando éste se ejecuta.

Es importante mencionar que los componentes visuales son exclusivos del marco de trabajo para el que fueron diseñados, por ejemplo, un componente visual de VCL no puede ser usado en el marco FMX. No obstante, la mayoría de los componentes no visuales son compatibles para ambos marcos de trabajo.

Tanto el servidor TCP, así como los clientes TCP denominados Comandantes, están basados en los componentes no visuales del proyecto Indy y han sido programados en el entorno de desarrollo RAD Studio XE8.

1.2.3 El proyecto Internet Direct (Indy)

El proyecto Indy es un conjunto de componentes no visuales de código abierto, que comprehenden una enorme biblioteca para la programación de *sockets de Internet* y está disponible para múltiples plataformas. Indy está escrito en el lenguaje Object Pascal e incluye componentes para crear clientes y servidores TCP o UDP, así como implementaciones de más 100 protocolos de nivel superior como SMTP, FTP, y HTTP.

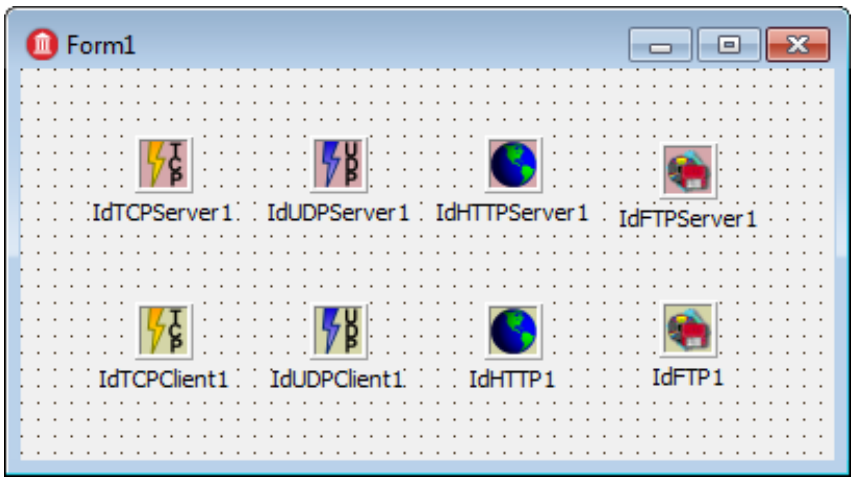


Figura 0.8 Algunos componentes Indy

Los sockets de Indy están disponibles para las siguientes plataformas de desarrollo [5]:

- *C++ Builder / Delphi / Kylix.*
- *Visual Basic.NET, C #* y otros lenguajes *.NET.*
- *FreePascal* y el *IDE Lazarus.*

[5] De manera general, un *socket de Internet* (se traduce del inglés como enchufe de Internet) es el proceso o conjunto de procesos que se encarga de crear el conducto virtual para la comunicación entre un programa Cliente y un programa Servidor en una Red TCP/IP. Para más información véase el apéndice A.

Los servidores Indy hacen uso extensivo de hilos de procesamiento, lo cual permite manejar un amplio número de conexiones simultáneas. La programación de un socket de Indy es relativamente simple, muy parecida al manejo de archivos. Además, gracias a que los componentes Indy son componentes no visuales, la mayoría de estos son compatibles con el marco multiplataforma FMX y los programas que usan estos componentes pueden ejecutarse en los sistemas operativos Windows, Android, iOS, OSX y Linux.

1.2.4 El router inalámbrico

El router inalámbrico es el dispositivo que se encarga de crear y mantener la *WLAN*. Es recomendable contar con un equipo que permita, de manera sencilla, administrar dicha red, esto es: asignar a voluntad dirección IP a los *hosts* de la red, configurar la redirección de puertos, crear listas de acceso y redes virtuales, control parental, cambiar las credenciales de identificación inalámbrica, entre otras configuraciones útiles. Para este proyecto se utilizó el router inalámbrico comercial LINKSYS EA2700 (véase la figura 1.9), el cual es de fácil instalación y permite las configuraciones antes mencionadas.

Figura 0.9 Fotografía de Router LINKSYS EA2700, vista frontal

En la parte posterior (véase la figura 1.10) tiene 4 puertos Ethernet, los cuales se comportan como un *switch* de cuatro interfaces y una interfaz de conexión a Internet. El router encamina la información de un dispositivo de la red local a otro, pero cuando un dispositivo desea comunicarse con otro que no pertenezca a la red local, el router envía el tráfico a su interfaz de Internet con la intención de encontrar al dispositivo desconocido conectado a Internet. Este modelo, además permite crear una red de hasta 32 dispositivos inalámbricos conectados simultáneamente.

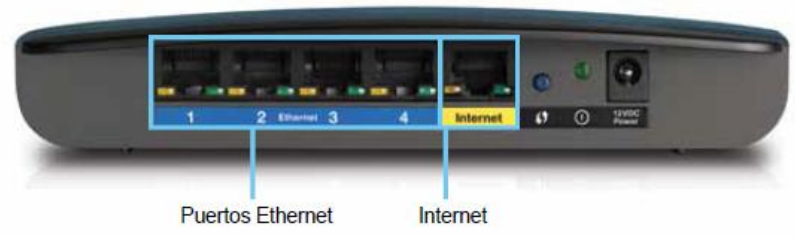


Figura 0.10 Fotografía de Router LINKSYS EA2700, vista trasera

En México, cuando se contrata un servicio de Internet, generalmente la compañía incluye en la renta un módem ADSL, un equipo de este tipo puede hacer las veces del router inalámbrico antes mencionado, aunque, generalmente estos dispositivos suelen tener menores prestaciones, (como el alcance de la señal inalámbrica), pues han sido seleccionados para que la compañía ahorre dinero. Los módems ADSL se diferencian de los routers inalámbricos en que los primeros, además de proveer conexión a Internet y crear una WLAN, modulan y demodulan señales digitales que viajan en el par trenzado de la red telefónica.

CAPÍTULO 2. DISEÑO DEL SOFTWARE EJECUTABLE EN DISPOSITIVOS INTELIGENTES Y COMPUTADORAS PERSONALES.

Tal y como se explicó anteriormente, el sistema CIAM utiliza un servidor TCP, para intercomunicar a *comandantes* y *subordinados*. En este capítulo, si bien no se explicará línea por línea código fuente del servidor, sí se explicarán sus componentes, métodos, propiedades y eventos más sobresalientes. Una vez hecho esto, se explicará lo propio del software de los clientes denominados *comandantes*. Tanto los *comandantes*, como el servidor TCP están basados en los componentes del proyecto Indy, y han sido programados en el ambiente de desarrollo integrado RAD Studio XE8.

2.1 El componente *TIdTCPSever*.

En el entorno de desarrollo RAD Studio XE8 está instalada, por defecto, la versión 10 de los componentes Indy, estos se pueden encontrar en la paleta de herramientas.

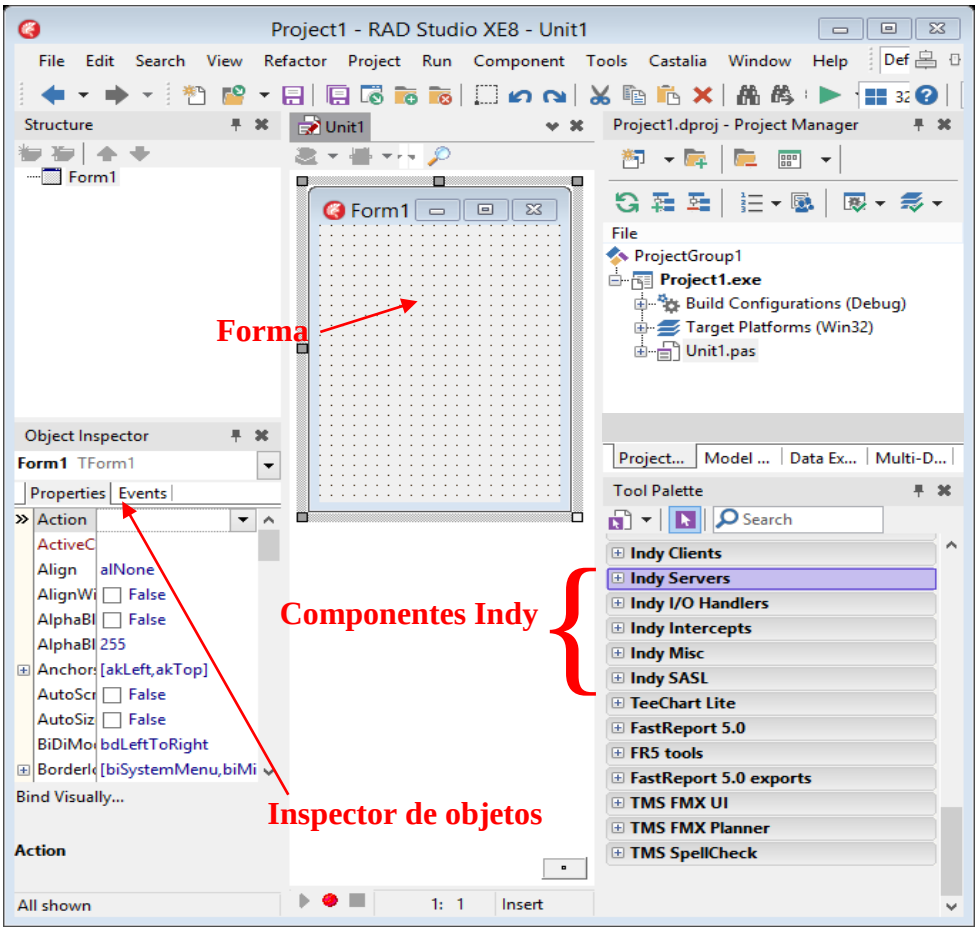


Figura 2.1 Ubicación de los componentes Indy en la ventana del entorno RAD Studio

El servidor Zero está basado en el componente *TIdTCPSever*, éste es una clase que facilita la programación de servidores TCP, que al igual que la mayoría de los componentes servidores de Indy, está diseñado para hacer uso de hilos de ejecución. *TIdTCPSever*, en su configuración por defecto, funciona como se describe a continuación.

TIdTCPSever crea un hilo de ejecución secundario, independiente del hilo principal, que está a la espera de solicitudes entrantes por parte de los clientes. Para cada cliente que lo solicite, *TIdTCPSever* crea un hilo nuevo para servir específicamente a la conexión con ese cliente, a partir de este punto se pueden disparar eventos en el contexto de ese hilo, por ejemplo, cuando el cliente "habla" con el servidor, se dispara el evento *OnExecute* en el cual se puede leer el mensaje del cliente con el comando *ReadLn*. Cada hilo tiene su propia pila, aislada del programador, el componente *TIdTCPSever* de Indy implementa internamente muchos detalles que son automáticos y transparentes. La Figura 2.2 muestra un esquema del funcionamiento del componente antes descrito.

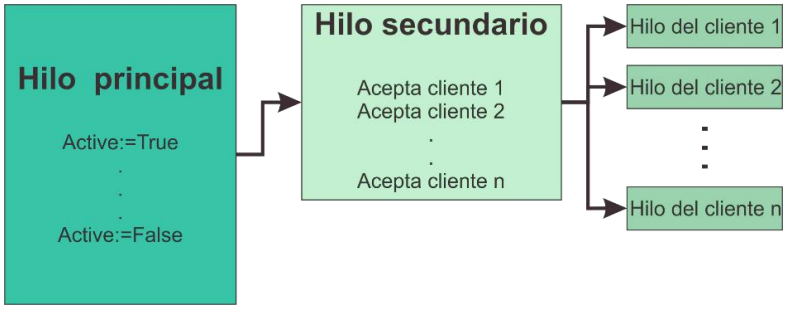


Figura 2.2 Esquema de un servidor Indy

Los eventos en Indy también están hilados, y aunque no pertenezcan a una clase de hilaturas, son disparados dentro del contexto de un hilo. De esta manera, los eventos se pueden diseñar al momento de escribir el código, sin necesidad de declarar clases personalizadas o sobrescribiendo métodos manualmente. Por ejemplo, si se están recibiendo mensajes al mismo tiempo, de diferentes clientes, el evento *OnExecute* se disparará individualmente en el contexto de cada uno de los clientes, de esta manera se pueden mantener múltiples "conversaciones" simultáneas. En los siguientes párrafos se mostrará un ejemplo con el código mínimo para crear un servidor TCP con Indy. El componente *TIdTCPSever* deberá estar presente en la forma, además se hará uso de un botón y una caja memoria, la cual servirá para testificar el funcionamiento del programa. En la Figura 2.3 se muestra la forma del programa en cuestión.

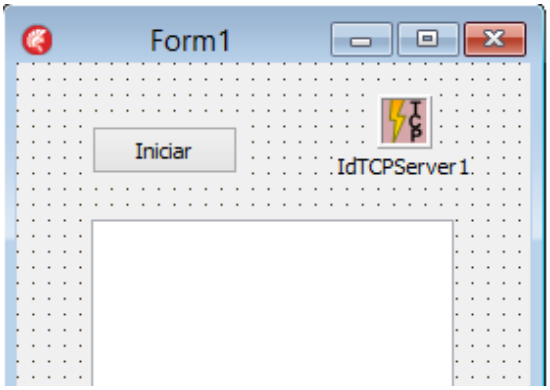


Figura 2.3 Interfaz de usuario del servidor TPC ejemplo

Antes de activar el servicio TCP se debe modificar la propiedad *Binding* del componente *TIdTCPSever* en el inspector de objetos (véase Figura 2.1), agregando la dirección IP en red y el puerto donde se recibirán las solicitudes entrantes. La dirección IP debe estar asignada al dispositivo en el que se ejecutará el servidor y el puerto debe estar libre, no debe estar en uso por otro programa.

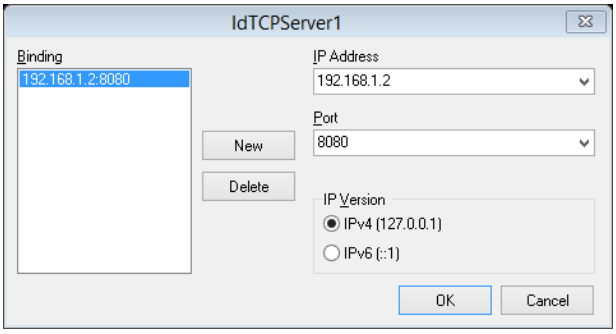


Figura 2.4 Cuadro de diálogo de la propiedad Binding del componente TIdTCPServer.

La IP y el puerto del servidor, también pueden modificarse en el código fuente del programa, por ejemplo, en el método *OnClick* del botón que está en la forma, en este mismo método se puede activar el servidor. El código se muestra a continuación.

```
//Procedimiento OnClick del botón "Iniciar"
 procedure TForm1.BotIniciarClick(Sender: TObject);
 begin
   IdTCPServer1.Bindings.Add.IP := '192.168.1.2'; //Asigna IP
   IdTCPServer1.Bindings.Add.Port := 8080;        //Asigna puerto
   IdTCPServer1.Active := true;                   //Inicia servicio TCP
   BotIniciar.Caption := 'Iniciado...';           //
   BotIniciar.Enabled := False;                   //deshabolita botón "iniciar"
 end;
```

El servidor TCP del sistema CIAM, ha sido programado usando este modelo de ejecución. Al implementar un servidor usando dicho modelo, se debe definir el procedimiento *OnExecute*, o bien, se debe sobrescribir el método *DoExecute* usando el comando O*verride*. Cuando un cliente se conecta al servidor, ocurre la siguiente secuencia:

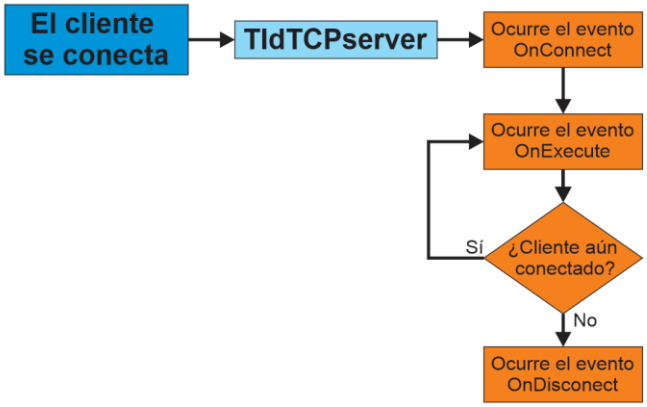


Figura 2.5 Diagrama de un cliente conectándose a un servidor Indy

Justo después de que el cliente se conecta, se dispara un evento que llama al procedimiento *OnConnect*, en este punto se pueden implementar protocolos de seguridad o de identificación del cliente, por ejemplo, solicitarle al cliente un nombre de usuario y contraseña. Inmediatamente después, se dispara el evento que llama a *OnExecute*, al finalizar éste, para revisar el estado del cliente, se verifican las siguientes condiciones automáticamente:

- El cliente no solicitó la desconexión.
- El método *OnDisconnect* no fue llamado durante el método *OnExecute.*
- No ocurrieron excepciones no soportadas durante el método *OnExecute.*
- No ocurrieron errores fatales.
- El servidor sigue Activo.

OnExecute seguirá llamándose repetidamente para cada cliente conectado, siempre y cuando estas condiciones se cumplan y el cliente pase su *contexto de conexión*[6] como argumento al método *OnExecute*. Si el método *OnExecute* no está definido, se dispara una excepción, es decir un error con la posibilidad de ser manejado (como cuando se intenta la división de un número entre cero). El siguiente código corresponde los métodos *OnConnect y OnExecute.*

```
// Método OnConnect
procedure TForm1.IdTCPServer1Connect(AContext: TIdContext);
var
  Anfitrión: String;
begin
  Anfitrión := AContext.Connection.Socket.Binding.PeerIP;
  Memo1.lines.Add('El cliente ' + Anfitrión + ' se ha conectado');
end;

// Método OnExceute
procedure TForm1.IdTCPServer1Execute(AContext: TIdContext);
Var
  CadenaEntrante: String;
begin
  try
    begin
      CadenaEntrante := Trim(AContext.Connection.iohandler.ReadLn);
      if CadenaEntrante <> '' then
        Memo1.lines.Add('El cliente dice: ' + CadenaEntrante)
      else AContext.Connection.Disconnect;
    end;
  except
    on e: Exception do
    begin
      AContext.Connection.iohandler.Writeln('Error=' + e.message);
    end;
  end;
end;
```

[6] El contexto de una conexión con un cliente está asociado con el hilo de ejecución creado para dicho cliente; tiene que ver, por ejemplo, con la pila asignada a éste para almacenar sus variables locales, con sus atributos, métodos, etc.

La mejor manera de desconectar a un cliente del servidor es llamando al método *OnDisconnect*, aunque el cliente también puede solicitar la desconexión. Siguiendo con el ejemplo, el siguiente código corresponde al método *OnDisconnect.*

```
// Método OnDisconnect
procedure TForm1.IdTCPServer1Disconnect(AContext: TIdContext);
var
  Anfitrión: String;
begin
  Anfitrión := AContext.Connection.Socket.Binding.PeerIP;
  Memo1.lines.Add('El cliente ' + Anfitrión + ' se ha desconectado');
end;
```

En el código antes mostrado, el método *OnConnect* testifica la conexión del cliente. En el método *OnExecute* se verifica que la trama enviada por el cliente no este vacía, si esto se cumple, el mensaje se agrega a la memoria de comandos, si no se cumple se llama al método *OnDisconnect*, en el cual se testifica la desconexión del cliente. Para verificar el funcionamiento de este programa, se utilizó un navegador de Internet, en este caso se usó el navegador Mozilla Firefox, y se introdujo la dirección IP y el puerto del Servidor en la barra de direcciones, esta información corresponde a la asignada a la propiedad *Binding* del componente *TIdTCPSever*, tal y como se muestra en la figura 2.4. El resultado se muestra en la siguiente captura de pantalla.

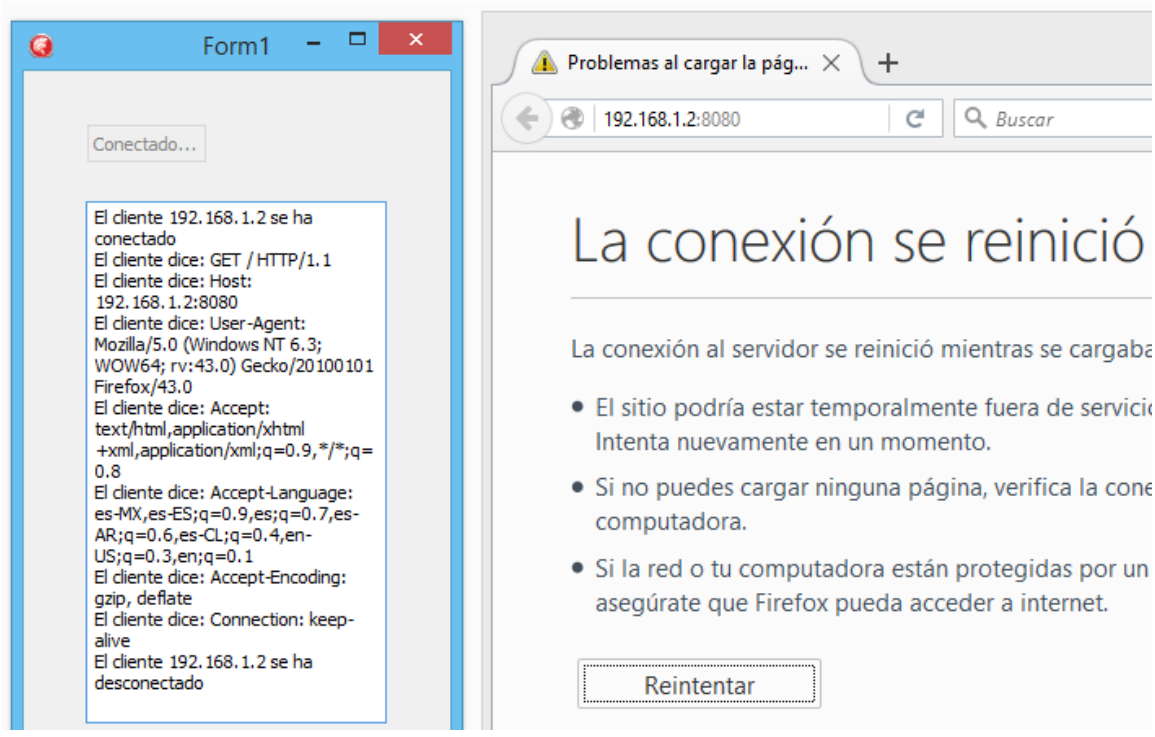


Figura 2.6. Ejecución del servidor TPC ejemplo[7]

[7] El código fuente del servidor TCP ejemplo se encuentra en el apéndice B.

Como se puede observar en la Figura 2.6, cuando el navegador se conecta al servidor, este último agrega, a la caja de memoria, la línea "El cliente 192.168.1.2 se ha conectado". Como el navegador se ejecuta en la misma computadora que el servidor, ambos tienen la misma dirección IP, en este momento se establece el conducto virtual de comunión entre el servidor y el navegador, es decir se establece un *socket* (enchufe). Luego, el navegador envía varios comandos al servidor, los cuales corresponden al protocolo HTTP, en efecto, el navegador intenta descargar una página web, pues para ello ha sido diseñado. En la Figura 2.6, estos comandos se agregan a la caja de memoria, y son precedidos por la frase "El cliente dice". El comando final que envía el navegador es un mensaje de fin de texto, el cual es interpretado por el servidor como mensaje vacío, provocándose la desconexión y testificándose con la adición de la línea "El cliente 192.168.1.2 se ha desconectado", a la caja memoria. Finalmente, el navegador interpreta esto como un reinicio en la conexión. Para más información sobre los servidores TCP de Indy revisar [6].

2.2 El servidor TCP "Zero" del Sistema CIAM.

Con base en el componente *TIdTCPSever*, se ha programado el servidor TCP denominado "Zero". A diferencia del servidor ejemplo recientemente explicado, Zero hace uso de un record de longitud dinámica, para almacenar el socket de cada cliente que se conecta. En programación, un record es una estructura de datos parecida a los arreglos, que, en lugar de almacenar datos de un mismo tipo, almacenan variables u objetos de diferentes clases. Zero, además de tener la capacidad de comunicarse individualmente con cada cliente, es capaz de redirigir el tráfico de un cliente a otro, usando el formato de mensajes del sistema CIAM, definido en el primer capítulo de este documento. El servidor Zero funciona como se explica a continuación:

Al ejecutarse, aparece una caja de diálogo que pregunta al usuario si desea cambiar la dirección IP y el puerto del servidor, si el usuario escoge *no*, se usará la configuración por defecto, si el usuario escoge *sí*, se le preguntará por la nueva configuración mediante cajas de entrada (véase Figura 2.7).

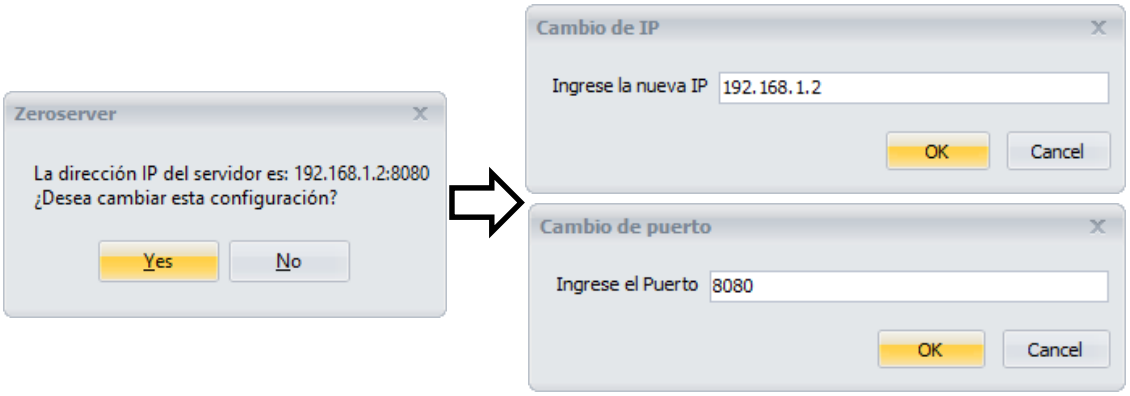


Figura 2.7 Cambio de la dirección IP y puerto del servidor Zero

Como se puede observar en la Figura 2.5, cuando un cliente se conecta a Zero, ocurre el evento que llama al método *OnConnect*, el cual se mantiene a la espera de un mensaje. Este mensaje debe ser el identificador[8] del cliente, si el identificador empieza por los caracteres "sub", Zero lo añade a la lista de clientes *subordinados*, en otro caso lo añadirá a la lista de clientes *comandantes*. Si un cliente proporciona el identificador que ya está en uso por algún otro dispositivo, Zero solicitará al cliente que cambie su nombre.

La diferencia entre *comandantes* y *subordinados* es que Zero no comparte la lista de dispositivos conectados con los clientes *subordinados*, en otras palabras, los *subordinados* no pueden saber que otros dispositivos están conectados, mientras que los *comandantes* sí. Esto es así porque los *subordinados* están diseñados solamente para ejecutar comandos y no para solicitar que éstos sean ejecutados por otro cliente. Una vez que se ha establecido la conexión con el cliente, Zero está preparado para responder las solicitudes de ese cliente.

Para entender cómo funciona el servidor, se pondrá como ejemplo el siguiente escenario: Supóngase que se conectan dos dispositivos a Zero; Adams, un *comandante* ejecutándose en un dispositivo móvil inteligente y sub_Bravo, un *subordinado* que está conectado a una interfaz con un relevador. Si Adams quiere que sub_Bravo modifique el estado del relevador, deberá enviarle un mensaje a Zero con el formato de mensajes del sistema CIAM, por ejemplo, de la siguiente manera:

"msg=Adams#sub_Bravo#SwitchRel"

Zero recibirá este mensaje y durante el método *OnExecute*, lo procesará para obtener el destinatario y se lo reenviará a sub_Bravo. Luego entonces, sub_Bravo recibirá el mensaje, lo procesará para obtener el remitente y el contenido, y responderá con un mensaje que podría ser de la siguiente manera:

"msg=sub_Bravo#Adams#RelSwitched"

Nuevamente, Zero recibirá este mensaje, lo procesará para obtener el destinatario y lo reenviará a Adams, el cual, de acuerdo a la respuesta, notificará el resultado en su interfaz de usuario. Además, Zero tiene en su interfaz de usuario una caja de memoria, en la que escribe todos los mensajes que le envían los clientes. Si Zero recibe un mensaje con el formato de mensajes del sistema CIAM, y el destinatario no se encuentra en su lista de dispositivos conectados, únicamente se agrega dicho mensaje a la caja de memoria. Si el mensaje no tiene el *formato de mensajes del sistema CIAM*, éste se agrega a la caja de mensajes sin afectar el funcionamiento general del sistema.

La siguiente imagen muestra la Interfaz de usuario del servidor Zero, con dos clientes conectados.

[8] Este identificador es denominado "*Nickname*" en el código fuente de Zero.

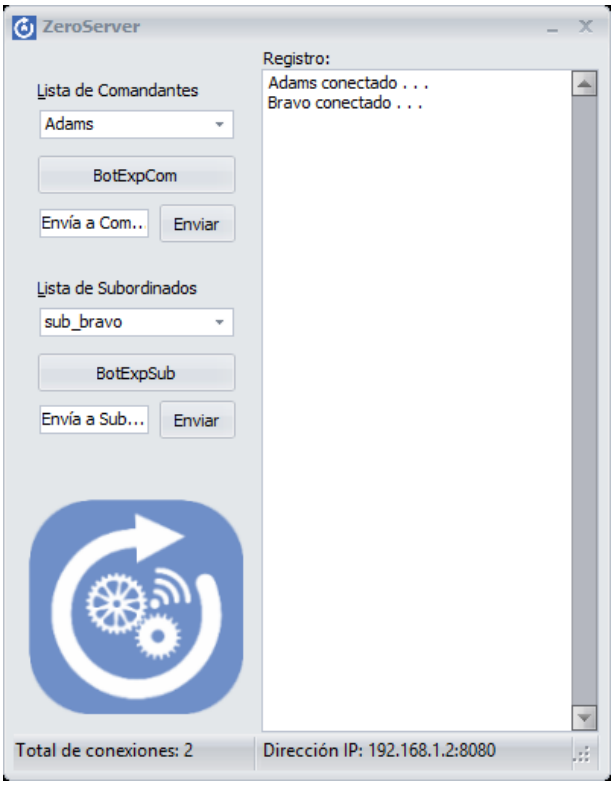


Figura 2.8 Interfaz de usuario del servidor Zero

Para limpiar la caja de memoria basta con hacer clic en el logo de los engranes. Si se desea desconectar un comandante se debe escoger de *la lista de comandantes* y presionar el botón *BotExpCom.* En el caso de que se desee desconectar un *subordinado,* éste se debe escoger de la *lista de subordinados* y presionar el botón *BotExpSub*.

Desde Zero se pueden enviar mensajes a los clientes conectados con el formato de mensajes del sistema CIAM, si desea enviar un mensaje a un *comandante,* se debe escribir en la caja de texto debajo del botón *BotExpCom* y presionar el botón *Enviar* adyacente a esa caja de texto. Para enviar un mensaje a un *subordinado,* se debe escribir en la caja de texto debajo del botón *BotExpSub* y presionar el botón *Enviar* adyacente a esa caja de texto. El mensaje será enviado al cliente seleccionado en la lista de *comandantes o subordinados,* según sea el caso, y el remitente cualquier caso es "Zero".

2.3 Hardware necesario para ejecutar un servidor TCP.

En teoría un servidor Indy podría ser capaz de dar servicio a todas las direcciones *IP* del planeta, esto es solo limitado en la realidad por el poder de procesamiento y la capacidad de memoria del hardware donde se ejecuta el servidor, así como la rapidez en la que se pueden transferir datos a través de la red TCP/IP.

Los servidores juegan un papel importante en Internet, por lo que deben tener altas prestaciones tanto en la rapidez de procesamiento de datos como en la capacidad de almacenamiento de estos. Por ejemplo, los servidores de la empresa Google, están instalados

en complejas instalaciones a prueba de terremotos, con respaldo de energía eléctrica, refrigeración, sistemas de protección, etc. [7].

Figura 2.9 Interior del centro de servidores de Google en Council Bluffs, Iowa, Iowa, U.S.A. (Fuente [7])

Un único servidor de este estilo puede llegar a atender a cientos de miles de clientes, lo que significa que se crean cientos de miles de hilos de procesamiento. La mayoría de estos hilos generalmente están esperando datos, y mientras hacen esto, se encuentran inactivos, por ejemplo, en un servidor con 500 hilos sólo unos 25 han de estar activos al mismo tiempo.

En el caso de Indy, se ha puesto a prueba una computadora, con un procesador Pentium III a 750[MHz] y con 256[MB] de memoria RAM, ejecutando un servidor Indy con 513 hilos y haciendo uso únicamente del 1% de la capacidad del procesador [8]. El verdadero problema con los hilos es la memoria disponible, pues cada uno se le asigna su propia pila, en esta misma computadora, unos mil hilos comienzan a dar problemas por esta razón. En este caso, Indy permite modificar el valor de la pila que le asigna a cada hilo, sin embargo, en este punto deberían considerarse otras alternativas más avanzadas.

El Servidor Zero del sistema CIAM, se ejecuta en una computadora personal con las siguientes características:

Descripción	Laptop Lenovo® IdeaPad U410 Touch.
Procesador	Intel® Core™ i5 a 1.80[GHz].
Sistema operativo	Windows 8.1 con arquitectura de 64 bits.
Memoria RAM	8GB PC3-12800 DDR3 1600 MHz.
Memoria Flash	1TB.
Conectividad de red	Disponible una entrada para cable Ethernet. Modulo Intel Centrino Wireless N-2230 802.11 b/g/n Wifi. Modulo Bluetooth® 4.0.

Tabla 2.1 Características del hardware del Servidor TCP.

Esta computadora es mucho más poderosa que la del ejemplo de los 513 hilos, por lo que se puede concluir que Zero podría dar servicio a unos cuantos miles de clientes. Cabe destacar que el código fuente del servidor del sistema CIAM es únicamente ejecutable por un equipo con los sistemas operativos Windows 7/8/8.1/10 y Windows Server® 2008/2012. Esto se debe a que ha sido desarrollado usando el marco de trabajo VCL.

2.4 El componente *TIdTCPClient*.

Para el diseño de los clientes *comandantes*, se utilizó el componente *TIdTCPClient*. Aquí se muestra una aplicación ejemplo, demostrando la forma en la que fueron programados los clientes *comandantes*. En este caso, el programa se diseñó usando el marco de trabajo *Firemonkey*, de manera que sea compatible con los sistemas operativos Windows y Android, y que se conecte de manera adecuada con Zero.

A diferencia de los servidores, los clientes Indy no trabajan con hilos de procesamiento por sí solos, entonces, con finalidad de hacer más eficiente la comunicación, se puede definir una nueva clase, "hija" de la clase de hilos *TThread* (más información en [9]). Ésta es una clase abstracta que permite la creación de hilos de ejecución independientes, cada nueva instancia de una clase "hija" de *TThread* es un nuevo hilo de ejecución.

Para este ejemplo se utilizará la siguiente interfaz de usuario.

Figura 2.10 Interfaz de usuario del cliente comandante ejemplo

En realidad, lo que se busca hacer es que el cliente se comporte de manera similar al servidor, pero a diferencia de éste, el cliente sólo necesita manejar una única conexión.

Antes de la implementación del código, se debe definir la clase a la que denominaremos, por comodidad, *TClientThread*; así:

```
TClientThread = class(TThread) //TClientThread Hereda atributos de TThread.
protected
  procedure Execute; override; //Definicion del método Execute
public
  constructor Create(CreateSupended: Boolean); //Definición de un método
                                               //contructor de una instancia
                                               //de TClientThread.
end;
```

Con este código, la clase *TClientThread* hereda todos los atributos de la clase *TThread*. Para crear un objeto de *TClientThread* (es decir, un hilo de ejecución) se usa el constructor *Create*, el cual es el método constructor por defecto de *TThread.* En su implementación se debe establecer la prioridad del hilo e indicar en qué momento se libera dicho hilo. El siguiente código es la implementación del constructor *Create,* para crear un hilo con prioridad normal, obtenida de [10].

```
constructor TClientThread.Create(CreateSupended: Boolean);
begin
  inherited Create(CreateSupended);
  Priority:=tpNormal;
  FreeOnterminate := True;
end;
```

Si se establece la propiedad *FreeOnTerminate* como verdadera, no hay necesidad de utilizar un destructor cuando el hilo termina de ejecutarse. En el caso de la propiedad *Priority*, los otros valores que puede tomar se muestran la siguiente tabla.

Valor	***Propiedad***
tpIdle	El hilo se ejecuta sólo cuando el sistema está inactivo. Windows no interrumpirá los otros hilos para ejecutar un hilo con prioridad *tpIdle*.
tpLowest	La prioridad del hilo es de dos puntos por debajo de la normal.
tpLower	La prioridad del hilo es de un punto por debajo de la normal.
tpNormal	El hilo tiene prioridad normal, igual a la del proceso principal.
tpHigher	La prioridad del hilo es de un punto por arriba de la normal.
tpHighest	La prioridad del hilo es de dos puntos por arriba de la normal.
tpTimeCritical	El hilo se pone más alta prioridad.

Tabla 2.3. Valores de prioridad de un hilo de ejecución paralelo.

Para crear el hilo basta con invocar en algún punto al constructor *Create.* Si se le pasa el parámetro *CreateSuspended* como falso, el método *Execute* se invoca inmediatamente, si *CreateSuspended* es verdadero, *Execute* no será llamado hasta que se invoque el método *Resume*[9].

9 Tanto las propiedades *Priority* y *FreeOnTerminate*, así como los métodos *Create*, *Execute* y *Resume* están predefinidos para la clase *TThread.*

Para conectarse al servidor, se deben modificar las propiedades *Host* y *Port* del componente *TIdTCPClient* ya sea en el código fuente o bien en el inspector de objetos. Posteriormente se debe invocar el método *Connect* del mismo. Esto se puede hacer, por ejemplo, en el evento *OnClick* del botón *BotConectar*.

```
procedure TForm1.BotConectarClick(Sender: TObject);
begin
  IdTCPClient1.Host := '192.168.1.2'; //Dirección IP de Zero
  IdTCPClient1.Port := 8080;          //Puerto
  IdTCPClient1.Connect;               //Invoca método Connect
  BotConectar.Enabled := False;       //Deshabilita botón BotConectar
end;
```

Cuando se establece la conexión con el servidor, se dispara un evento que invoca al procedimiento *OnConnected* del componente *TIdTCPClient*. En el cual, según la filosofía del sistema CIAM, se puede aprovechar para anunciarle el identificador al servidor, el cual, en este ejemplo debe ser introducido manualmente en la caja de texto, etiquetada como *Identificador* (véase Figura 2.10*)*. Además, es recomendable que ente punto, también se invoque al constructor *Create* para crear el hilo de ejecución paralelo, el cual estará al pendiente de los mensajes que envíe el servidor. La definición del evento *OnConnected* debería ser como sigue.

```
procedure TForm1.IdTCPClient1Connected(Sender: TObject);
  begin
    IdTCPClient1.IOHandler.WriteLn
      ('nick=' + NickNameEdit.Text);   // Envia identificador al servidor
    TClientThread.Create(False);       // Invoca constructor del hilo de ejecución
    end;
```

Como se explicó con anterioridad, al llamar al constructor con parámetro *False*, se invoca inmediatamente el procedimiento *Execute*, el cual se usará de manera similar al procedimiento *OnExecute* de los servidores TCP de Indy. Se puede decir que, con esta configuración, el cliente TCP aquí mostrado funciona como un servidor TCP de Indy, conectado con un único cliente. En este método se verifica que el cliente aún esté conectado, si es así, se lee el buffer de datos recibidos, si éste no está vacío, se agrega el mensaje entrante a la caja de memoria. Así:

```
procedure TClientThread.Execute;
var
  MensEnt: String;
begin
  with Form1 do
  begin
    if not IdTCPClient1.Connected then exit;
    repeat
      MensEnt := IdTCPClient1.IOHandler.ReadLn;
      if trim(MensEnt) <> '' then Memo.Lines.Add(MensEnt);
    until not IdTCPClient1.Connected;
  end;
end;
```

Los componentes Indy soportan múltiples métodos para la lectura y escritura de datos, los cuales incluyen métodos para esperar, verificar o polear datos. En Indy 10, estos métodos están definidos en la clase *TIdIOHandler*. Toda conexión Indy, ya sea de un cliente o un de un servidor, utiliza la clase *IOHandler* como una propiedad. Con lo anterior se manejan de una manera eficiente los mensajes que el cliente *comandante* recibe. Para enviar un mensaje se usa el método *WriteLn* de la propiedad *IOHandler* del componente *TIdTCPClient*. En este el ejemplo, al pulsar el botón *BotEnviar*, se enviará cualquier cadena que contenga la caja de texto, ubicada arriba de dicho botón. El código es el siguiente:

```
procedure TForm1.BotEnviarClick(Sender: TObject);
begin
  IdTCPClient1.IOHandler.Writeln(MensajeEdit.Text);
end;
```

En la siguiente captura, se demuestra el funcionamiento de este programa, conectándose al servidor Zero y enviándole un mensaje.

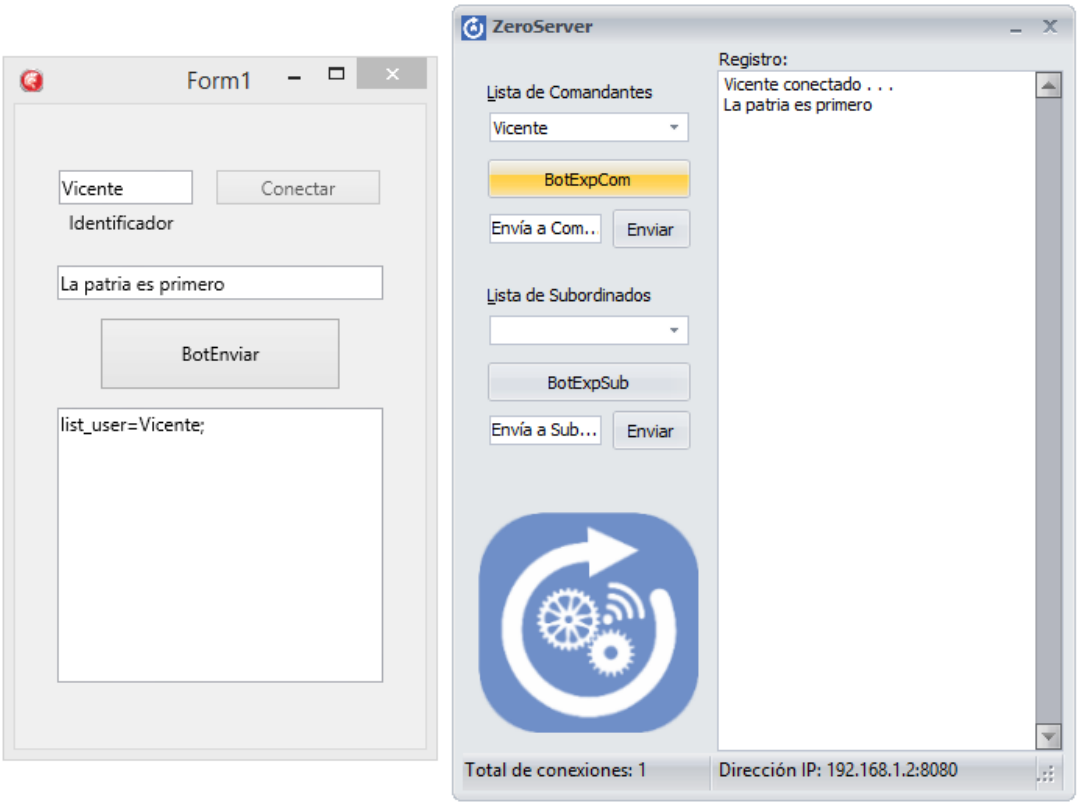


Figura 2.11 A la izquierda, ejecución en Windows del cliente comandante ejemplo. A la derecha el servidor Zero

Como se puede observar en la Figura 2.11, el *comandante* "Vicente" recibió un mensaje con la lista de usuarios conectados, precedido por la cadena "*list_user*". Además, el mensaje "*La patria es primero*" fue enviado por Vicente al servidor Zero, como dicho mensaje no tiene el formato de mensajes del sistema CIAM, Zero se limita a añadirlo a su caja memoria.

Para enviar un mensaje a otro dispositivo se debe usar el formato de mensajes de sistema CIAM. En la Figura 2.12, se demostrará la conversación entre dos de estos clientes, uno ejecutándose en un dispositivo con sistema operativo Windows y otro en un dispositivo móvil con sistema operativo Android, donde el primero enviará un mensaje al segundo, usando del formato de mensajes del sistema CIAM.

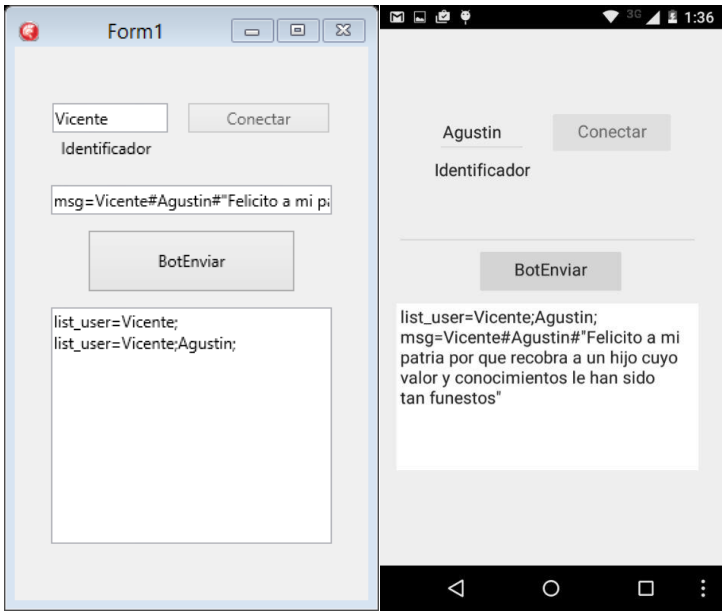


Figura 2.12 Ejemplo de conversación entre dos clientes comandantes. La captura de la derecha pertenece a un dispositivo con sistema operativo Android

En la siguiente imagen se muestra la secuencia del intercambio de mensajes antes mostrado.

Figura 2.13 Secuencia de envío de información de un cliente a otro

Como se puede observar, Agustin[10] recibe el mensaje como es, por lo que, en un programa más detallado, se deberían agregar procedimientos y funciones que el automaticen el procesamiento de la cadena recibida, todo esto se puede hacer en el evento *Execute* de la clase *TClientThread*. Esto con la finalidad de que el formato de mensajes de sistema CIAM sea transparente al usuario de la aplicación.

[10] Nótese la falta de acento en este nombre propio, pues Indy no da soporte a caracteres del ASCII extendido.

2.5 Hardware necesario para ejecutar un el cliente TCP.

Los *comandantes* son procesos que se conectan al servidor como clientes TCP, que además incluyen la interfaz de usuario que facilita la comunicación de los *comandantes* con el servidor. En teoría, el código fuente de los comandantes, es ejecutable en dispositivos con las siguientes características.

- Equipos con Microsoft® Windows 7/8/8.1/10/ Server (32-bit o 64-bit).
- Computadoras personales con sistema operativo MAC OS X 10.9/10.10
- Teléfonos móviles o tabletas electrónicas con sistema operativo Android 4.0 o más reciente, o bien sistema operativo iOS 7/8 (iPod, iPhone, o iPad).

El dispositivo que valide a un comandante debe contar con el hardware necesario para conectarse a una red TCP/IP, como puede ser la tecnología Wifi, o una tarjeta de red con entrada ethernet. En general, casi todos los dispositivos móviles modernos cuentan con esta funcionalidad, al igual que la mayoría de las computadoras personales de escritorio o portátiles.

Cabe mencionar que, durante el desarrollo del proyecto explicado en esta tesis, los *comandantes* sólo han sido probados en dispositivos con los sistemas operativos Windows y Android. Esto debido a que los equipos de Apple tienen un costo mayor y las aplicaciones requieren ser programadas en una computadora personal con sistema operativo de Apple.

CAPÍTULO 3. DISEÑO DE HARDWARE Y SOFTWARE ASOCIADO CON LOS MÓDULOS DE ACTUACIÓN Y MONITOREO REMOTO.

El presente capítulo describe el diseño de la tarjeta de desarrollo MINICONW_SH32, el cual está basado en las tarjetas de destino MINICON_XX, desarrolladas por el M.I. Antonio Salvá Calleja, profesor de carrera del Departamento de Control y Robótica de esta Facultad. Como esta tarjeta incluye un módulo ESP8622 Wifi, la descripción del hardware y firmware de este módulo también se explica en este capítulo.

Asimismo, se presenta el diseño de una biblioteca de subrutinas que permite al microcontrolador, presente en la tarjeta MINICONW_SH32, configurar e interactuar con el módulo wifi ESP8622 a través del puerto serial. La versión 12 de este módulo es la utilizada en la tarjeta MINICONW_SH32, sin embargo, la biblioteca funciona para cualquier versión del módulo que tenga el firmware descrito en el presente capítulo.

3.1 El módulo ESP8266 Wifi.

Los módulos *Wifi ESP8266* son pequeñas tarjetas electrónicas clasificadas como *SoC* (del inglés *System on a Chip*, sistema en un chip) que incluyen tecnología para conectarse a una WLAN utilizando el mecanismo de conexión *Wifi*. Todos los módulos *Wifi ESP8266* están basados en el chip *ESP8266* producido por el fabricante chino *Espressif.*

A mediados del año 2014 comenzaron salir noticias sobre la primera versión del módulo *ESP8266* (véase [11]), el ESP-01 que permitía a cualquier microcontrolador o sistema embebido conectarse fácilmente a una red Wifi mediante un conjunto de comandos basado en los comandos Hayes[11]. Desde entonces se han desarrollado múltiples versiones del módulo, cada una con diferentes características.

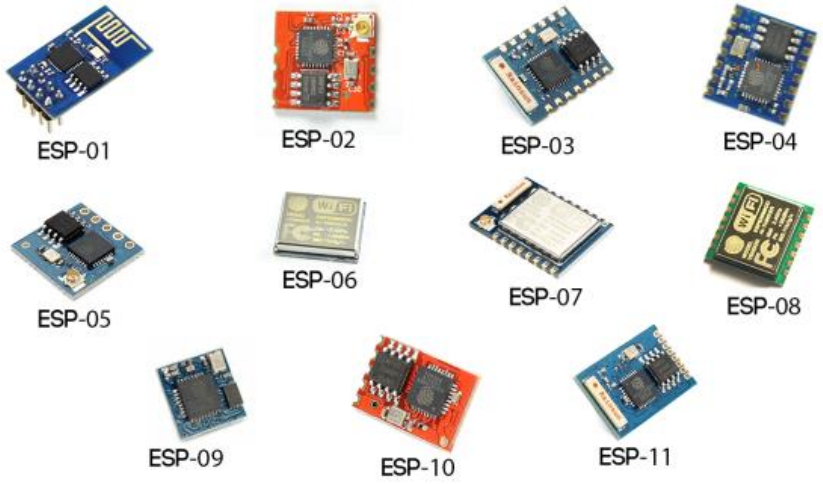


Figura 3.1 Diferentes versiones del Módulo ESP8266 Wifi (fuente [12])

[11] El conjunto de comandos Hayes, es un estándar abierto de comandos utilizados para configurar y parametrizar dispositivos de red, los caracteres AT preceden a todos los comandos de este estilo.

En general, cualquier módulo ESP8266 cuenta con las siguientes características:

- CPU base: LX106 de la compañía Xetensa, con arquitectura RISC de 32 bit, corriendo a 80 [MHz].
- 64 [KB] de memoria RAM para instrucciones, más 96 [KB] de RAM para datos.
- Memoria flash desde 512 [KB] hasta 16 [MB].
- Tecnología Wifi según la norma IEEE 802.11 b/g/n Wifi.
- Hasta 16 pines de propósito general (GPIO) y convertidor analógico digital.
- Periféricos de comunicación SPI, I²C y UART.
- Pines de reinicio y polarización.

A continuación, se muestra una tabla con algunas de las distintas versiones del Módulo.

Versión	Pines	Separación	Antena	Dimensiones [mm]	Notas
ESP-01	GPIO0/2/16	0.1 [in]	Grabada en PCB	14.3 × 24.8	Este es el modulo más común, existen múltiples variantes.
ESP-02	GPIO0/2/15	0.1 [in]	Conector U-FL	14.2 × 14.2	Existen múltiples variantes.
ESP-03	GPIO0/2/12 /13/14/15/16	2 [mm]	Cerámica	17.3 × 12.1	Este es el modulo más popular y utilizado en Internet.
ESP-04	GPIO0/2/12 /13/14/15/16	2 [mm]	Ninguna	14.7 × 12.1	Tipo de antena personalizable por el usuario.
ESP-05	Ninguno	0.1 [in]	Conector U-FL	14.2 × 14.2	Existen múltiples variantes.
ESP-06	GPIO0/2/12 /13/14/15/16	Miscelánea	Ninguna	14.2 × 14.7	El escudo de metal dice contar con certificación FCC Ce.
ESP-07	GPIO0/2 /4/5/12/13 /14/15/16	2 [mm]	Ambas, Cerámica y Conector U-FL	20.0 × 16.0	Algunas versiones tienen un error en serigrafía, la tapa dice contar con certificación FCC Ce.
ESP-08	GPIO0/2/12 /13/14 /15/16	2 [mm]	Ninguna	17.0 × 16.0	Igual al ESP-07, pero con antena personalizable por el usuario.
ESP-09	GPIO0/2/12 /13/14/15	miscelánea	Ninguna	10.0 × 10.0	Es el módulo más pequeño del mercado, 1[MB] memoria Flash.
ESP-10	Ninguno	2 [mm]	Ninguna	14.2 × 10.0	Solo interfaz UART.
ESP-11	GPIO0/1	1.27 [mm]	Cerámica	17.3 × 12.1	Poca información disponible.
ESP-12	ADC + GPIO0 /2/12/13 /14/15/16	2 [mm]	Grabada en PCB	24.0 × 16.0	El escudo de metal dice contar con certificación FCC, posee 4[MB] de memoria Flash.

Tabla 3.1 Características del hardware del Servidor TCP. Fuente: varias páginas de ventas.

3.1.1 Ventajas y desventajas de los módulos *Wifi ESP8266*.

Ventajas:

- Permiten, mediante una interfaz serial asíncrona, a cualquier aplicación en microcontrolador conectarse a Internet.
- Pueden funcionar como punto de acceso y comportarse como servidores TCP o UDP.
- Son capaces de funcionar como estaciones, que se conectan a un punto de acceso y conectarse, a un servidor TCP o UDP, como clientes.
- Son muy baratos, el precio de los módulos varía entre 2 USD y 5 USD.
- Son muy pequeños, la versión más pequeña tiene un área de apenas un centímetro cuadrado.
- Existen ejemplos y videos de ciertos módulos *ESP8266* estableciendo una conexión con un punto de acceso a más 300 [m] al aire libre usando la antena grabada en el circuito, esta distancia aumenta considerablemente con una antena externa [13].
- El fabricante *Espressif* ha liberado un SDK (paquete de desarrollo de software), con lo que el firmware de los módulos se puede modificar, aunque esto aún está en fase de desarrollo.
- En función del firmware grabado en la memoria de los módulos, éstos son capaces de establecer un modo de comunicación transparente, donde los datos trasmitidos por su receptor serial son encapsulados según el protocolo TCP y enviados a un servidor, del mismo modo, y los datos transmitidos por el servidor se desencapsulan en el módulo para luego ser reenviados por su trasmisor serial. En otras palabras, el módulo se comporta como si de un cable serial se tratara.

Desventajas:

- La información disponible está muy dispersa y la gran mayoría de esta es una mala traducción del chino al inglés.
- A pesar del gran éxito que han tenido estos módulos en occidente, aún es muy difícil de conseguirlos en la Ciudad de México, la mejor forma de conseguirlos es ordenarlos por Internet y tardan en llegar unas 5 semanas.
- Algunos módulos, son enviados con otro firmware diferente al oficial o son versiones piratas vendidas como originales, esto hace que el dispositivo funcione de manera inesperada.
- Para lograr un máximo alcance los módulos pueden llegar demandar hasta 500[mA] a la fuente de alimentación, esto encarece el diseño de dicha fuente.
- Algunos módulos tienen errores en la máscara de serigrafía de la tarjeta, o bien dicen haber obtenido la certificación FCC Ce, lo cual en algunos casos no es verdad.

3.1.2 Hardware recomendado para el funcionamiento del módulo *ESP8266* Wifi.

El hardware recomendado para el funcionamiento de un módulo *ESP8266* es un circuito simple, que requiere de muy pocos componentes externos y algunas conexiones entre pines del chip. Este circuito es el siguiente:

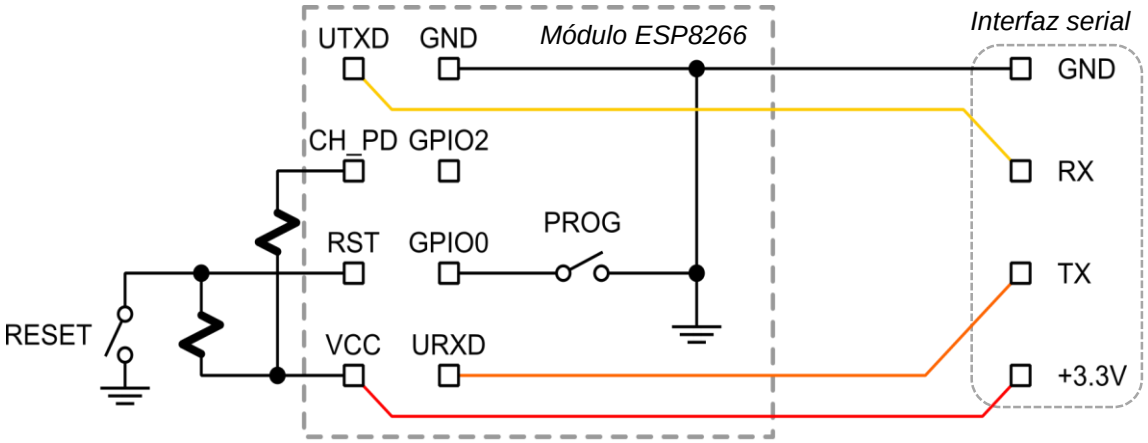


Figura 3.2 Circuito recomendado para el Módulo ESP8266, basado en la hoja de datos

El módulo funciona con una polarización de 3.3[V], y es capaz de comunicarse con cualquier dispositivo que soporte la comunicación serial asíncrona. Es importante destacar que a este dispositivo no se le debe aplicar una diferencia de potencial mayor a 3.3[V] a ninguna de sus interfaces, ya que esto podría dañarlo. Este circuito está basado en las indicaciones de la hoja de datos descargable de [14].

3.1.3 Firmware del módulo ESP8266

Existen diversas versiones en Internet del firmware para los módulos ESP8266, algunas son personalizadas, habilitan diversas formas de funcionamiento y la mayoría de esta son de código abierto o están en fase de desarrollo. El firmware más estable es el proporcionado por el fabricante *Espressif,* el cual está bien documentado y es actualizado regularmente. Este se puede descargar directamente de la su página de Internet desde [15].

Asimismo, existe muchas herramientas gratuitas que permiten descargar el firmware mediante el puerto serial. Además, es posible modificar el firmware a voluntad, gracias a que existe un paquete de desarrollo de software liberado por el fabricante, aunque esto requiere más conocimientos de programación y podría generar errores o comportamientos inesperados.

Este firmware permite configurar e intercambiar información con el módulo *ESP8266* mediante comandos *AT*. El firmware está conformado por archivos binarios, y se puede descargar en el módulo mediante la herramienta oficial de *Espressif, Flash Download Tool*, descargable de la página web en [16]. Para cambiar el firmware se deben tener descargados los archivos binarios y la herramienta *Flash Download Tool,* luego, seguir los siguientes pasos:

1. Abrir el archivo ejecutable correspondiente a *Flash Download Tool*.
2. Colocar las rutas de los archivos binarios que conforman el firmware en los espacios asignados y a continuación la dirección donde deben ser almacenados. Esta información se proporciona en la siguiente tabla:

Archivo	**Dirección**	**Descripción**
esp_init_data_default.bin	0xFC000	Opcional. Restablece los parámetros de valores de RF.
blank.bin	0x7E000	Inicializa el área de parámetros de usuario.
blank.bin	0xFE000	
boot.bin	0x00000	Inicializa el área de parámetros del firmware.
user1.1024.new.2.bin	0x01000	Inicializa el área para firmware de usuario.

Tabla 3.2. Descripción de los archivos que conforman el firmware del ESP8266.

3. Elegir la opción de 8Mbit Flash o mayor, correspondiente al tamaño del firmware. Con estas configuraciones, la configuración de la herramienta *Flash Download Tool* debe ser la siguiente:

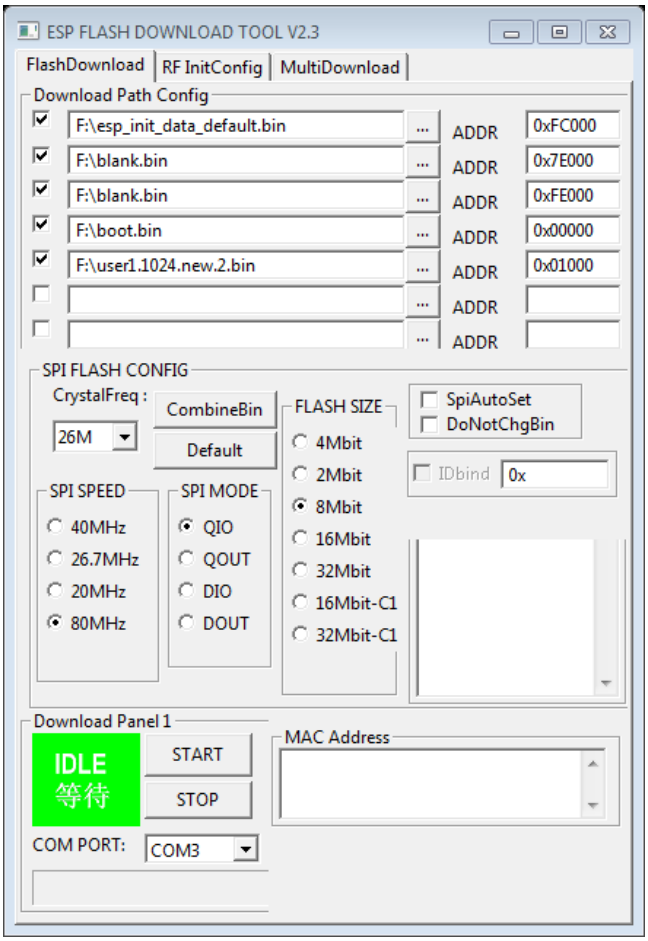


Figura 3.3 Interfaz de usuario de la herramienta Flash Download Tool

4. Poner el módulo en modo programación, conectando el pin GPIO0 a tierra.
5. Presionar el botón START. Al momento de presionarlo el pin de RESET del chip debe conectarse a tierra.

Una vez terminados estos pasos, el módulo ESP8266 ya tendrá almacenado el firmware en su memoria no volátil. A partir de este momento, se puede interactuar con el módulo a través del puerto serie, utilizando los comandos AT, los cuales se describen con mayor profundidad en el archivo *ESP8266 AT Instruction Set*. Este archivo se puede obtener de [14].

Para comprobar que el firmware ha sido grabado correctamente, se puede usar la ventana del emulador de terminal incluida en el software PUMMA_EST. Como este último está configurado para interpretar los caracteres que re reciben con una rapidez de 9600 bits por segundo, por lo que se debe cambiar la configuración para que acepte caracteres a 115200 bits por segundo.

Al enviar el comando de prueba "AT", el módulo debe responder "OK", y al enviar "AT+GMR", debe responder con la información de la versión del firmware que tiene grabada en su memoria. Cabe mencionar que cada comando debe escribirse seguido de los caracteres de control retorno de carro y salto de línea, los cuales se envían utilizando las teclas "ENTER", seguido de "CTRL+ENTER".

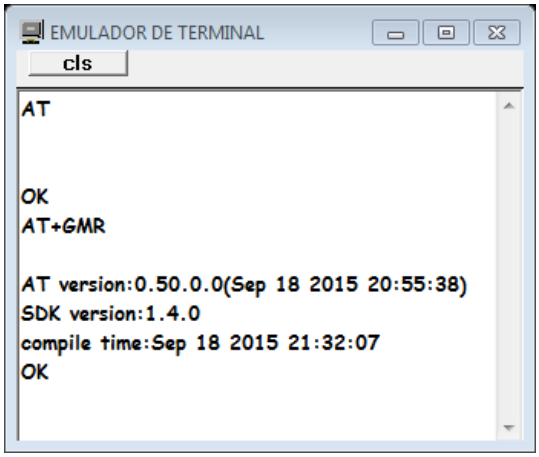


Figura 3.4 Comprobación de la versión del nuevo firmware en el emulador de terminal del PUMMA_EST

3.1.4 Modo de transmisión transparente del módulo ESP8266 Wifi.

La lista de comandos AT procesables por el módulo ESP8266 con el firmware de Espressif incluye 63 comandos, divididos en tres grandes grupos:

- Comandos AT Básicos
- Comandos AT para funciones de Wifi
- Comandos AT relacionados con los protocolos TCP/IP

Para el proyecto de esta tesis, se utilizaron específicamente aquellos comandos que configuran al módulo ESP8266 para conectarse al router inalámbrico descrito en el capítulo

1 de este documento, y luego, habilitan el modo denominado transmisión transparente, el cual permite que el módulo reciba datos como un cliente TCP y los repita por el puerto serial, o bien, que reciba datos por el puerto serial y los repita como cliente TCP. Para ello, se requiere enviar por el puerto serie del módulo los comandos AT que se describen a continuación.

- Establecer modo Wifi:

AT+CWMODE=1	// Modo estación
Respuesta: OK	

- Conexión al router inalámbrico:

AT+CWJAP="SSID", "password"	// SSID y password del router inalámbrico
Respuesta: OK	

- Conexión al servidor TCP:

AT+CIPSTART="TCP", "192.168.1.2", 8080	//Protocolo, dirección IP del servidor y puerto.
Respuesta: OK	

- Habilitar modo de transmisión transparente:

AT+CIPMODE=1
Respuesta: OK

- Iniciar envío de datos:

AT+CIPSEND	// Después de recibir el carácter “>”, se ha establecido el modo de comunicación transparente.
Respuesta: >	

3.2 El microcontrolador MC9S08SH32

Se eligió el microcontrolador MC9S08SH32 por ser compacto y funcional, ya que permite realizar tareas básicas de automatización y monitoreo, propias de un sistema embebido. Entre sus características principales se encuentran las siguientes:

- Soporta hasta 32 fuentes de interrupción o reinicio.
- Memoria FLASH de 32.768 kilobytes y RAM de 1024 bytes.
- Empaquetado compacto (28-SOIC) de bajo consumo de potencia.
- Opciones de fuente de reloj: externa o interna a una frecuencia de bus entre 2[MHz] y 20[MHz].
- Incluye periféricos tales como: Convertidor analógico-digital, comparadores analógicos, temporizador de dos canales, interfaz comunicación serial asíncrona, SPI e IIC, entre otros.

3.3 Diseño del hardware de la tarjeta de desarrollo MINICONW_SH32.

La tarjeta de desarrollo MINICONW_SH32 está diseñada para funcionar como un sistema embebido y adaptarse a múltiples proyectos, por su diseño compacto y posibilidad de conexión inalámbrica. El circuito impreso de la tarjeta MINICONW_SH32 fue diseñado mediante el software CAD *Proteus*, tiene unas dimensiones de 9[cm] de ancho por 6[cm] de alto. Incluye un microcontrolador MC9S08SH32, un módulo ESP-12 Wifi y una fuente de alimentación de 5[W] de potencia. En las siguientes imágenes se muestra la apariencia de la tarjeta.

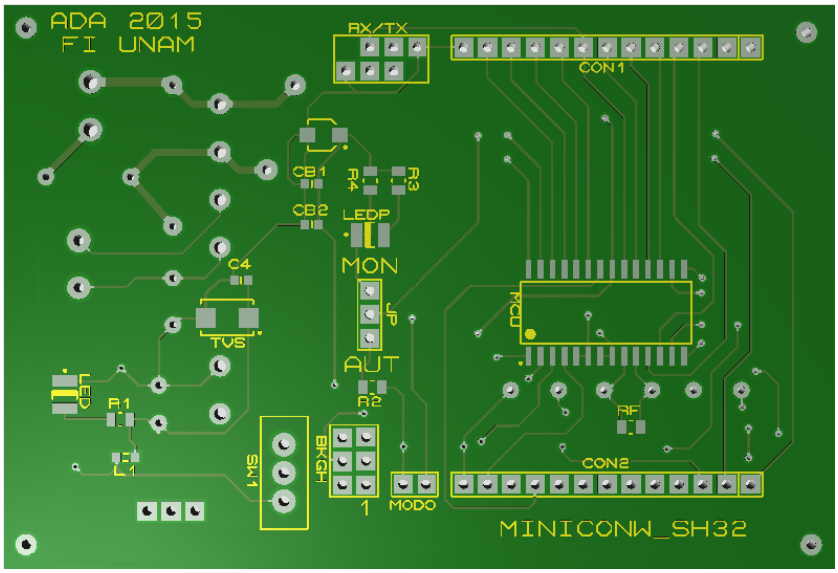


Figura 3.5 Anverso de la tarjeta MINICONW_SH32

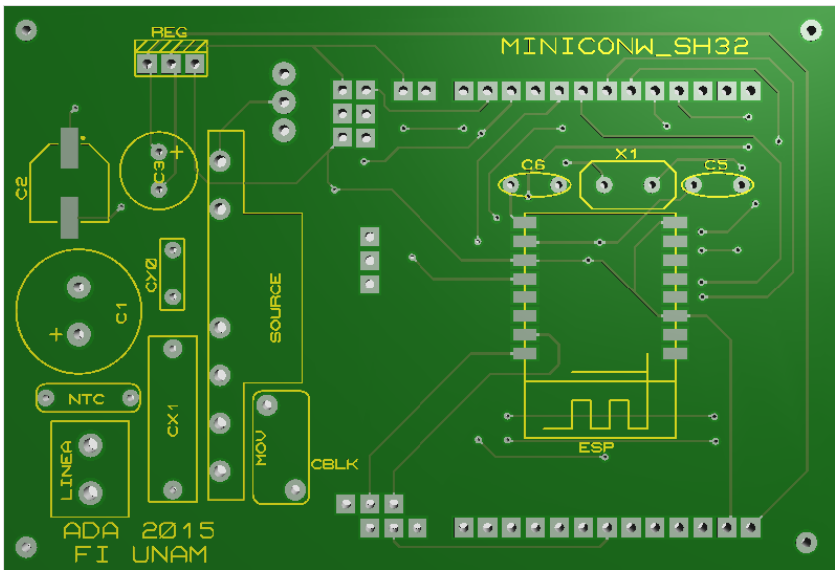


Figura 3.6 Reverso de la tarjeta MINICONW_SH32

A continuación, se muestra el esquemático de la tarjeta MINICONW_SH32.

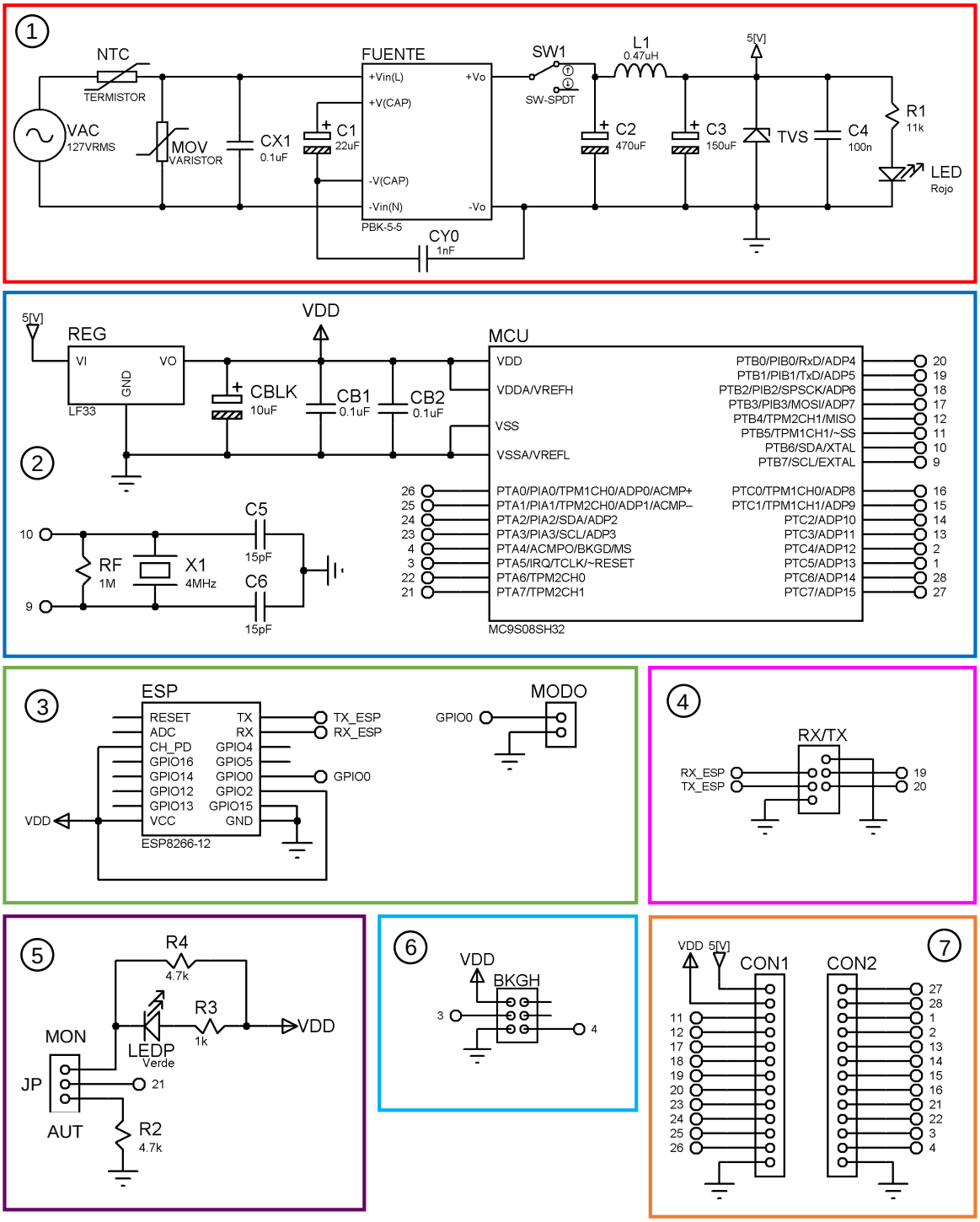


Figura 3.7 Bloques de la tarjeta MINICONW_SH32. 1: Fuente de alimentación - 2: Bloque del microcontrolador - 3: Bloque del módulo Wifi - 4: Interfaz de comunicación serial - 5: Terminal seleccionadora del modo de operación - 6: Background header - 7: Pines disponibles de la tarjeta

A continuación, se muestra la apariencia real de la tarjeta MINICONW_SH32.

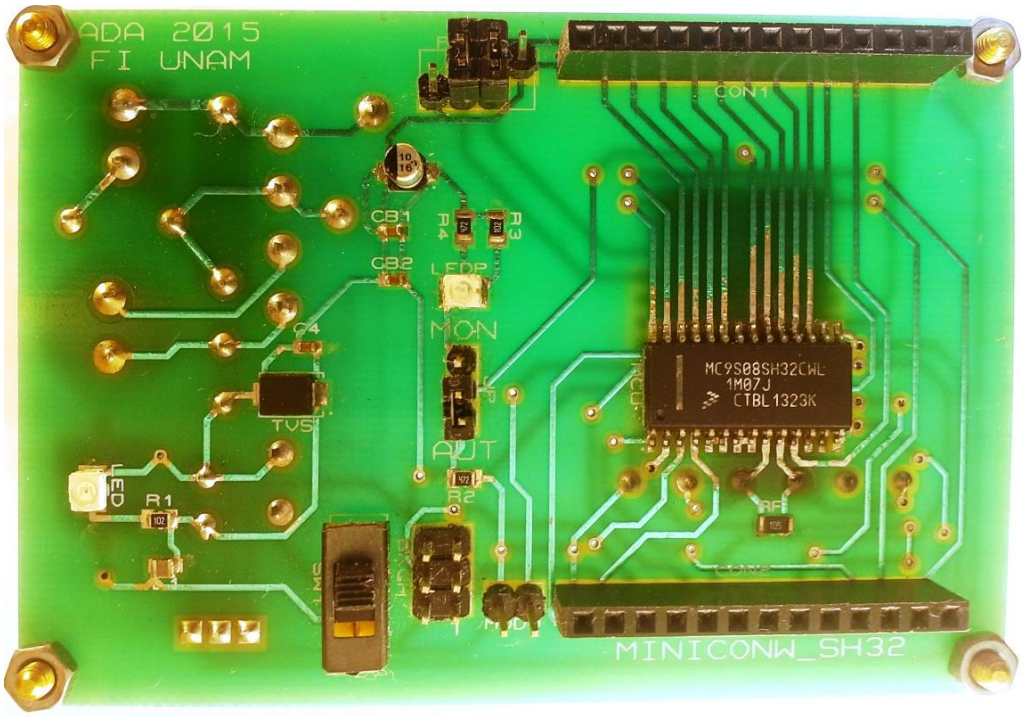


Figura 3.8 Una tarjeta MINICONW_SH32 armada

3.3.1 Fuente de alimentación.

La fuente de poder que polariza el circuito de la tarjeta MINICONW_SH32 es un circuito simple, cuyo componente principal es el integrado PBK-5-5, cuya hoja de especificaciones se puede descargar de [17], y constituye un convertidor de AC a DC, fabricado por la compañía CUI. Sus características principales son:

- Proporciona 5[W] de potencia continua.
- Empaquetado compacto tipo SIP.
- La tensión eléctrica de alimentación puede ser de 85 a 264 [V] de corriente alterna o de 100 a 400 [V] de corriente directa.
- La tensión de salida es de 5[V] de corriente directa.
- Aislamiento hasta 3,000 [V] de corriente alterna.
- Protecciones para corto circuito, y eventos de sobretensión y sobrecorriente.
- Eficiencia de hasta 75%.

Adicionalmente, el fabricante recomienda ciertos componentes externos para el funcionamiento y protección del PBK-5-5, los cuales se enlistan en la siguiente tabla:

Componente	Especificaciones
C1	22µF/400V
C2	470µF/16V
L1	0.47µH
C3	150µF/35V
C4	100nF/50V
CX1	0.1µF/275Vac
CY0	1nF/400Vac
Termistor (NTC)	5D-9
Varistor (MOV)	S14K350
Diodo supresor de transitorios de tensión (TVS)	SMBJ7.0A

Tabla 3.3: Componentes de la fuente de poder.

El circuito de la fuente mostrado en la Figura 3.7, bloque 1, cuenta con una etapa de protección a la entrada, formada por un termistor y un varistor. El termistor funciona como dispositivo limitador de corriente de arranque y el varistor que protege al circuito ante eventos de picos de tensión. A la salida cuenta con un diodo supresor de transitorios de tensión, el cual suprime toda tensión que supere su tensión de ruptura. Los capacitores, así como el circuito Pi formado por los capacitores C2 y C3 y el inductor L1 sirven como filtros.

3.3.2 Bloque del microcontrolador (MCU).

El circuito asociado al MCU de la tarjeta MINICONW_SH32, mostrado en la figura 3.7, bloque 2, está basado en el circuito mínimo funcional que aparece en la hoja de datos del MC9S08SH32, descargable de [18]. El MCU presente en la tarjeta contiene en su memoria no volátil el firmware denominado NBCP8_BIBAS8, que lo habilita como es un dispositivo CHIPBAS8. Este firmware valida entre otras facilidades las siguientes:

- El MCU puede ser manejado, vía serie, desde una computadora PC que ejecute el software manejador PUMMA_EST. Esto para fines de prueba y depuración de programas. Bajo esta circunstancia se dice que la tarjeta opera en modo monitor.
- El MCU puede ser programado en lenguaje BASIC, empleando para ello al compilador cruzado MINIBAS8A, que es parte de PUMMA_EST.
- El MCU opera a una frecuencia de bus de 20 MHz y se puede usar la instancia de interrupción IRQ del MCU.
- Una vez que un programa ha sido grabado en el MCU, éste se puede ejecutar de manera *autónoma*, simplemente cambiando la posición de un Jumper asociado a la terminal excluyente denominada JP.

La tarjeta MINICONW_SH32 hace uso del denominado Led de PUMMA, éste testifica mediante un parpadeo de cadencia apreciable, que la tarjeta pueda ser manejada desde una PC para fines de carga y depuración de programas, esto es, en modo monitor. Cuando se ejecuta un programa en forma autónoma este led permanece apagado. De acuerdo a la posición del jumper conectado a la terminal JP, la tarjeta opera en modo monitor cuando el Jumper colocado en los postes JP está del lado con la leyenda "MON". Si el Jumper está del lado con la leyenda "AUT", se ejecutará de manera autónoma el programa previamente grabado en el MCU. Este software de base y el concepto CHIPBAS8 fueron ideados y diseñados en el Departamento de Control y Robótica de esta facultad.

3.3.3 Bloque del módulo Wifi ESP8266.

Para este proyecto se utilizó la versión 12 del módulo ESP8266, descrito en la sección 3.1 de este capítulo. Este módulo ha de proveer de conexión inalámbrica a la tarjeta, además es capaz de establecer un modo de comunicación trasparente entre los protocolos serial y TCP. En la siguiente figura se muestra la apariencia este módulo.

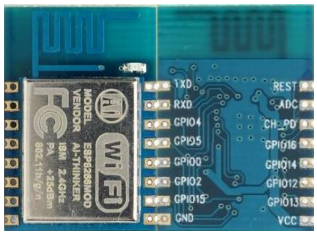


Figura 3.9 Fotografía del módulo ESP8266, versión 12. Anverso y reverso

El circuito asociado se muestra en la figura 3.7, bloque 3. Esta es una simplificación del circuito recomendado por el fabricante, que contiene los componentes mínimos para su funcionamiento en la tarjeta MINICONW_SH32. La terminal denominada MODO está asociada a los modos de operación del módulo: modo programación y modo ejecución. Como se explicó en la sección 3.1 de este capítulo, para el modo programación se requiere conectar el pin GPIO0 a tierra, mediante la terminal del MODO se puede lograr esta conexión, colocando un *jumper* que conecte los pines de la terminal. Para el modo ejecución, o modo normal de operación del módulo Wifi, el pin GPIO0 debe estar abierto o sin conexión.

3.3.4 Interfaz de comunicación serial.

Para interactuar con el puerto serial, ya sea del MCU o del módulo Wifi, se encuentra disponible el arreglo postes necesarios en la superficie de la tarjeta, en la disposición mostrada en la figura 3.6, bloque 4. Este arreglo de postes se utiliza si se desea comunicar al módulo Wifi con el MCU, se debe conectar mediante un *jumper* el pin RX_ESP con el pin 19, y el pin TX_ESP con el pin 20. Los pines 19 y 20 corresponden al pin TxD y al pin RxD del puerto serial del MCU, respectivamente.

3.3.5 Terminal seleccionadora del modo de operación.

El circuito contiene una terminal denominada JP, formada por tres postes. Esta parte del circuito contiene el Led de Pumma, distinguido por el color verde, asociado con los dos modos de funcionamiento de las tarjetas MINICON_XX: monitor y autónomo. Dicho Led es un testigo útil para el usuario, cuyo funcionamiento es controlado por el firmware almacenado en la memoria FLASH del MCU, denominado software de base.

Si se desea operar la tarjeta MINICONW_SH32 en el modo monitor, ya sea para programarlo o realizar pruebas con el manejador PUMMA_EST, se debe colocar un *jumper* que conecte el poste central con el extremo correspondiente denotado en la tarjeta por la etiqueta "MON". Por otro lado, para operar la tarjeta MINICONW_SH32 en el modo autónomo, una vez se ha grabado el programa en la memoria FLASH del MCU, se debe colocar un *jumper* que conecte el poste central con el extremo correspondiente denotado en la tarjeta por la etiqueta "AUT".

3.3.6 Terminal del Background header.

El arreglo de postes denominado *Background header*, denotado en la tarjeta MINICONW_SH32 por la etiqueta "BKGH", permite modificar el software de base del MCU mediante una interfaz BDM (del inglés *Background Debug Mode*, modo de depuración en segundo plano). Con esta herramienta, conectada a la terminal Background header, se graba en memoria flash el firmware de base del microcontrolador cuando éste está completamente nuevo.

3.3.7 Pines disponibles de la tarjeta.

La tarjeta de desarrollo MINICONW_SH32 cuenta con ciertos pines expuestos, disponibles al usuario, para las distintas funcionalidades de la tarjeta. A continuación, se muestra un esquema de dichos pines y su disposición.

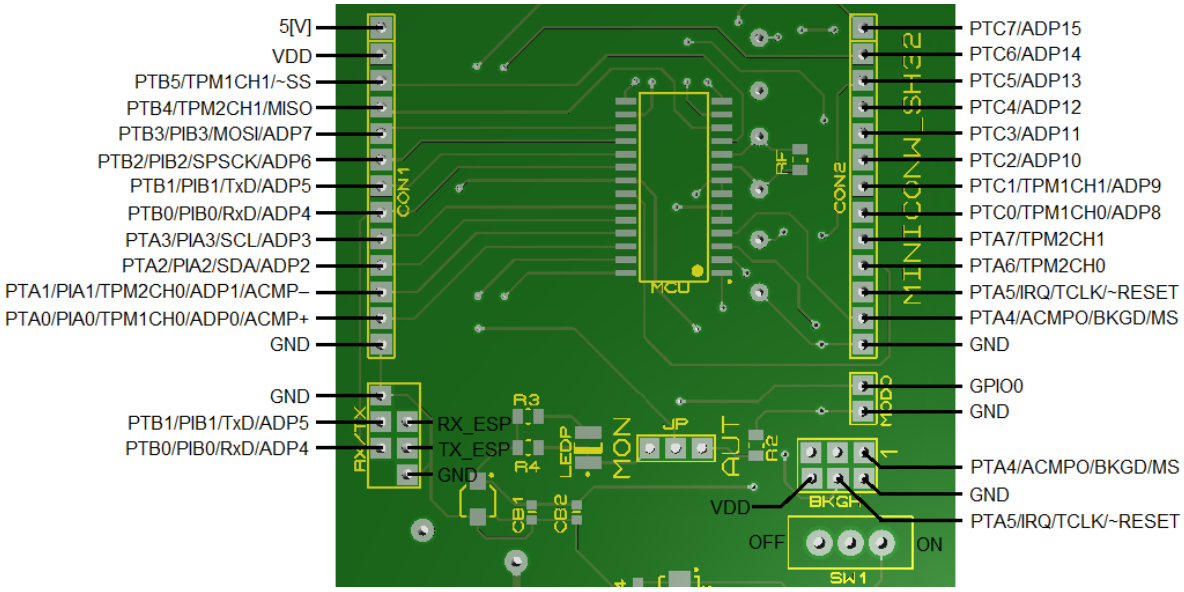


Figura 3.10 Pines de la tarjeta MINICONW_SH32 expuestos al usuario

3.4 Descripción de la biblioteca de subrutinas *bicoesp8266.*

En el contexto de este proyecto de tesis, el módulo Wifi es esclavo del MCU, y este último se encarga de la configuración del primero. El módulo ESP8266 Wifi debe ser configurado para que éste se conecte a la red inalámbrica y que posteriormente establezca un modo de comunicación transparente con el servidor Zero. En otras palabras, una vez realizada esta configuración, el módulo Wifi se comportará como un cable serial virtual, conectando al servidor Zero con el MCU de manera inalámbrica. Para facilitar esta configuración se ha diseñado una biblioteca de subrutinas, denominada *bicoesp8266*, que ha de incluirse en todo programa que desee utilizar esta funcionalidad, con ayuda de la sentencia #include. Esta biblioteca de subrutinas consta de dos archivos, denominados "bicoesp8266_con_sh32.b" y "bicoesp8266_rut.b". Estos archivos se incluyen en el apéndice C de este documento.

El archivo "bicoesp8266_con_sh32.b" incluye direcciones de registros propios del MCU MC9S08SH32, así como declaraciones de variables auxiliares, por lo que este archivo debe incluirse al principio del programa. El archivo "bicoesp8266_rut.b" contiene las subrutinas diseñadas para configurar al módulo ESP8266 Wifi y para procesar el envió/recepción de mensajes según el formato de mensajes del sistema CIAM.

El archivo "bicoesp8266_rut.b" debe incluirse antes de la colocación de vectores de interrupción que programa pueda llegar a necesitar, sin embargo, el código de la biblioteca bicoesp8266 configura el MCU en modo autónomo y hace uso de la rutina de interrupción por recepción serial, por lo que forzosamente se deben agregar los vectores de interrupción por *reset* y por recepción serial a la colocación de vectores de interrupción.

Las subrutinas de la biblioteca bicoesp8266 se describen a continuación.

Subrutinas para configuración del módulo ESP8266 Wifi:

- *Subrutina ConWFM*. Esta subrutina envía los comandos que un módulo ESP8266 Wifi requiere para conectarse a una red inalámbrica Wifi. Antes de invocar, se deben asignar el nombre de dicha red y su contraseña en las variables *SSID$* y *Pass$*, respectivamente. Al retornar, el módulo ESP8266ya se habrá conectado a la red Wifi.
- Subrutina IniTCP. Esta subrutina envía por el puerto serial los comandos necesarios que un módulo Wifi basado en el chip ESP8266 requiere para conectarse a un servidor TCP y configurarse en el modo de transmisión transparente. La dirección IP y el puerto correspondiente al servidor TCP, deberán estar almacenados en las variables *DirIP$* y *Puerto$* respectivamente. Al retornar, el módulo ESP8266 ya se habrá conectado al servidor TCP.

Subrutinas para el procesamiento de cadenas con el formato de mensajes de sistema CIAM:

Como se explicó en el capítulo 1 el formato de mensajes del sistema CIAM es el siguiente:

"**msg=**Remitente**#**Destinatario**#**Contenido"

Por lo tanto, las siguientes subrutinas han sido diseñadas tomando en cuenta este formato.

- *Subrutina CusInput*. Esta subrutina recibe, por interrupción del puerto serial asíncrono, una cadena de hasta 32 caracteres y la almacena en una variable de tipo *string$* denominada Cadena$.
- *Subrutina TrimCadena.* Esta subrutina elimina los caracteres especiales, tales como espacios en blanco, saltos de línea y retorno de carro, al inicio y final del contenido de la variable *Cadena$*.
- *Subrutina ValCadena.* Esta subrutina verifica que la variable *Cadena$* contenga una cadena con el formato de mensajes del sistema CIAM, más específicamente, verifica que *Cadena$* tenga una longitud apropiada, inicie con el encabezado "msg=" y que contenga los dos numerales que funcionan como caracteres de control. Si lo anterior se cumple, *InvalidC~* tomará el valor 0, de lo contrario *InvalidC~* valdrá 1.
- Subrutina ProCadena. Siempre y cuando la variable Cadena$ tenga un mensaje con el formato de mensajes del sistema CIAM, esta subrutina obtendrá lo correspondiente al *Remitente* y al *Contenido* de dicho mensaje, y almacenará esta información en las variables denominadas *Comandante$* y *Comando$*, respectivamente.
- Subrutina EnviaRes. Envía por el puerto serial una cadena de caracteres con el formato de mensajes del sistema CIAM, tomando como *Remitente* el identificador almacenado en la variable *Nick$*, como *Destinatario* el contiendo de la variable *Comandante$* y como *Contenido* lo almacenado en la variable *Resp$*.

El código comentado de la biblioteca bicoesp8266 se puede encontrar en el apéndice C de este trabajo de tesis.

3.4.1 Ejemplificación de uso de la biblioteca de subrutinas *bicoesp8266*

Se diseñó un programa ejemplo que puede ser utilizado como plantilla para otras aplicaciones ejecutables por la tarjeta MINICONW_SH32, que utilicen el sistema CIAM.

Para este programa plantilla, se consideran las siguientes condiciones: Las credenciales de identificación de red son: SSID: “CIAM” y contraseña: “a1b2c3d4e5”. La dirección IP y puerto del servidor Zero son: “192.168.1.2” y “8080”, respectivamente. El identificador o Nick con los que esta tarjeta se identifica en el servidor es: “sub_delta”.

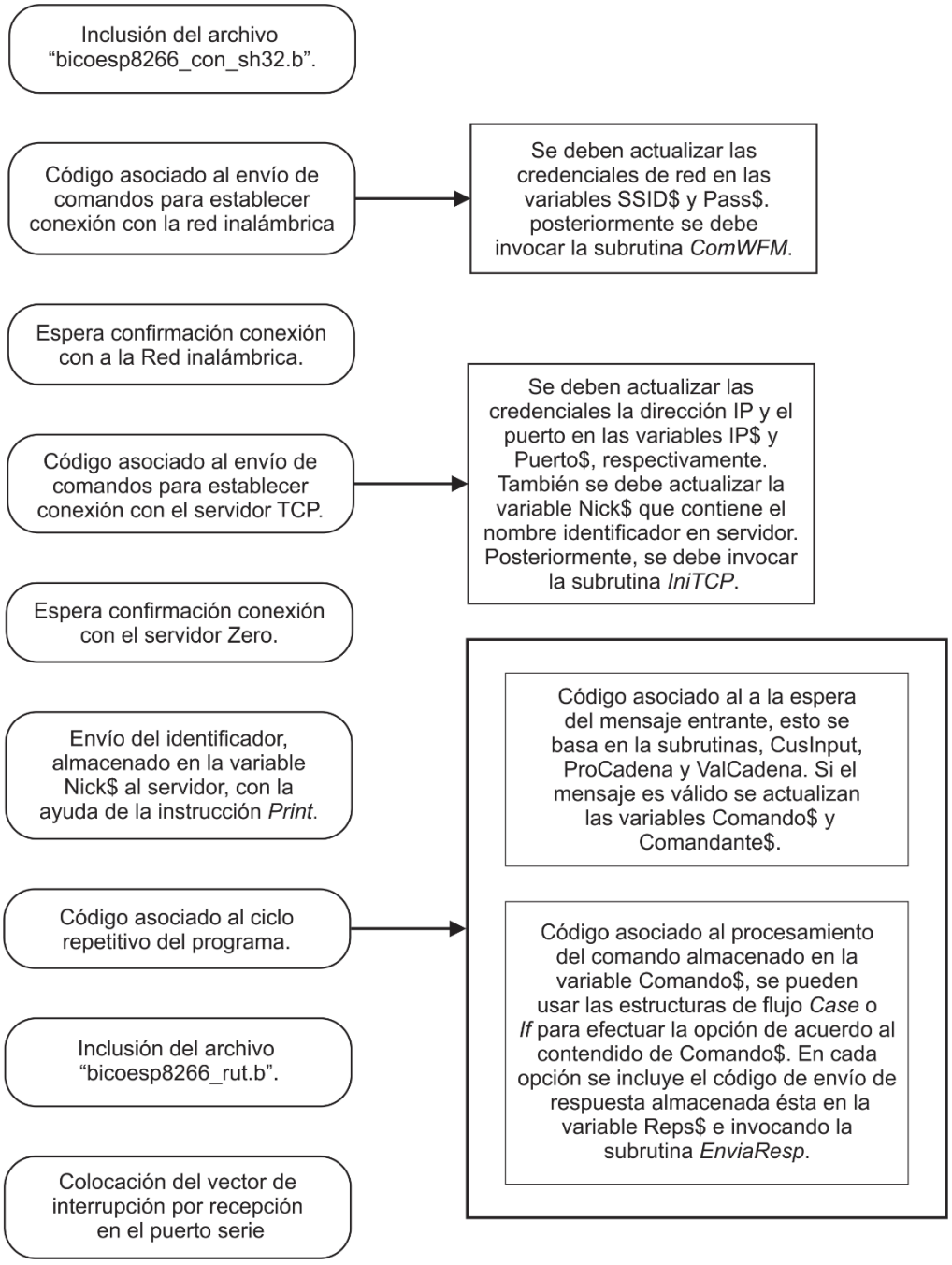


Figura 3.11 Estructura de los programas que usen el sistema CIAM

A continuación, se muestra el flujo de ejecución del programa plantilla.

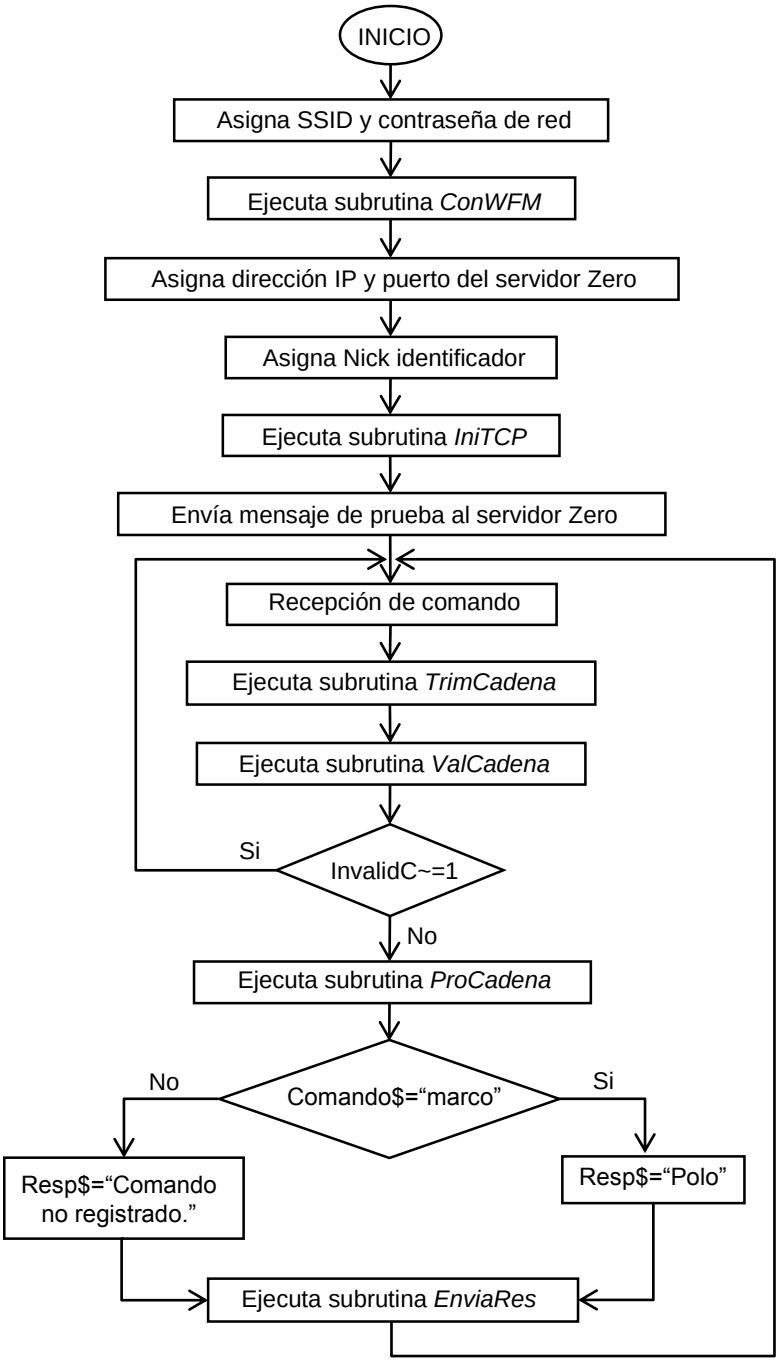


Figura 3.12 Flujo de ejecución del programa ejemplo

A continuación, se muestra el código fuente del programa plantilla.

```
#include "G:\SH32\bicoesp8266_con_sh32.b" 'Inclusión de código de configuraciones

             SSID$="CIAM"        'Network SSID
             Pass$="a1b2c3d4e5"  'Network password
             gosub ConWFM        'Establece conexión a la red inalámbrica
             DirIP$="192.168.1.2"      'Dirección IP del servidor Zero
             Puerto$="8080"            'Puerto del servidor Zero
             Nick$="sub_delta"         'Asignación del Nickname
             gosub IniTCP    'Establece conexión con Servidor Zero

             Print "sub_delta conectado . . ." 'Envió de mensaje de prueba a Zero

''''''''''''''''''''''''''INICIA PROGRAMA PRINCIPAL''''''''''''''''''''''''''''''
             while(1)
loop:        gosub CusInput                    'Recepción del mensaje
             gosub TrimCadena                  '
             gosub ValCadena                   '
             if InvalidC~=1 then               '
               goto loop                       '
             endif                             '
             gosub ProCadena                   'Procesamiento del mensaje
             if Comando$="marco" then          '
               Resp$="Polo"                    '
               gosub EnviaRes                  '
             else                              '
               Resp$="Comando no registrado."  '
               gosub EnviaRes                  '
             endif                             'Procesamiento del comando
             wend
''''''''''''''''''''''''''FIN PROGRAMA PRINCIPAL''''''''''''''''''''''''''''''

#include "G:\SH32\bicoesp8266_rut.b"      'Inclusión de código de subrutinas

''''''''''''''''''COLOCACIÓN DE VECTORES DE INTERRUPCIÓN''''''''''''''''''

             dataw &hd7dc servrecepsci
             dataw &hd7fe &h8000
```

Ejemplificación del programa plantilla:

Supóngase que al servidor TCP se conectan el subordinado “sub_delta” y el comandante “Adams”, donde este último es un proceso ejecutándose en un dispositivo Android. Cuando “Adams” envíe el comando “marco” con el formato de mensajes del sistema CIAM al servidor Zero, este último identificará y reenviará el mensaje a “sub_delta”, el cual como el comando que recibió es “marco”, éste responderá “Polo”. Si el comando no es “Polo”, “sub_delta”, responderá “Comando no registrado” igualmente con el formato de mensajes del sistema CIAM.

La siguiente es una captura de pantalla de la interfaz de usuario del servidor Zero ejemplificando dicho escenario.

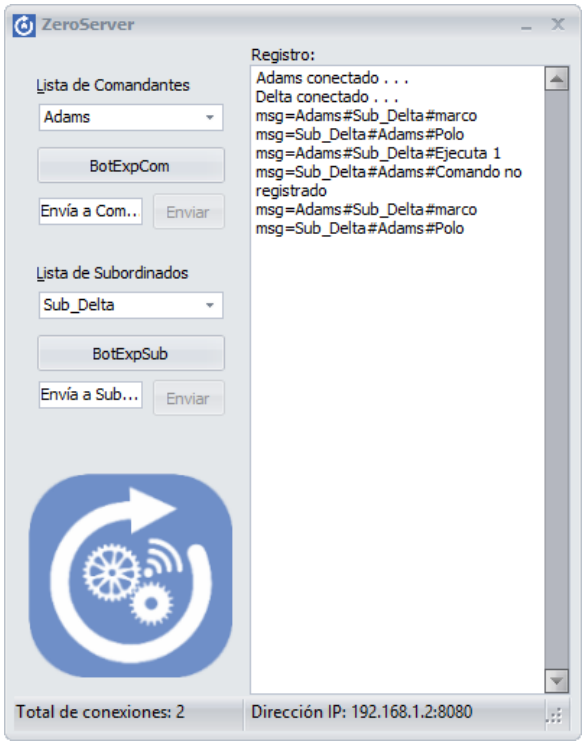


Figura 3.13 Demostración del funcionamiento del programa ejemplo, en la interfaz de usurario del servidor Zero

CAPÍTULO 4. EJEMPLOS DE APLICACIÓN.

El presente capítulo expone algunas aplicaciones del sistema CIAM, ilustrando su funcionamiento en una situación real, de manera que mediante ejemplos simples se pueda demostrar la capacidad y el potencial del sistema CIAM para realizar diversas tareas dentro del campo de la ingeniería electrónica. Se realizaron tres ejemplos de aplicación, orientados a tres entornos donde la ingeniería juega un papel importante: la electrónica de potencia, la instrumentación virtual y la automatización del hogar.

Para cada uno de estos tres ejemplos, se diseñó:

- Un circuito modular que se inserta en una tarjeta MINICONW_SH32, y contiene el hardware adecuado para cada aplicación.
- El software que ejecuta el MCU, diseñado en el entorno de desarrollo PUMMA_08+, en los lenguajes BASIC y ensamblador, para interactuar con el circuito modular correspondiente a cada aplicación, que utiliza como plantilla la biblioteca de subrutinas *bicoesp8266*.
- Un programa compatible con los sistemas operativos Android y Windows, diseñado en el entorno de desarrollo RAD Studio XE8, para operar en el sistema CIAM como un *cliente comandante* que envía los mensajes a procesar por la tarjeta MINICONW_SH32.

El programa que valida a cada *cliente comandante* ofrece al usuario la capacidad de configurar parámetros de red, estos son: su nombre identificador, la dirección IP y puerto al que se va a conectar. Estas configuraciones se pueden realizar accediendo a la pantalla de configuración accesible desde la pantalla de bienvenida, en Windows esta última cuenta con la apariencia mostrada en la siguiente figura.

Figura 4.1 Pantalla de bienvenida al programa que valida un cliente comandante

En la Figura 4.1, se muestra la pantalla de bienvenida de los programas que validan a cada cliente comandante ejecutándose en el sistema operativo Windows. Haciendo un clic en el escudo de la UNAM se pueden cambiar las configuraciones de red antes mencionadas, mientras que, haciendo un clic en el escudo de la facultad, se establece la conexión del programa con el servidor Zero.

4.1 Ejemplo de aplicación orientado a la Electrónica de Potencia.

En este ejemplo se demuestra la variación de la intensidad luminosa de un foco incandescente, así como del cambio de estado, activo o inactivo, de un tomacorriente. Si bien existen varias maneras de realizar estas acciones, en este trabajo se regula la intensidad del foco variando el ángulo de disparo de un triac, y el estado del tomacorriente se modifica mediante un relevador.

4.1.1 Circuito modular

Este circuito consta de tres partes principales que se muestran en la Figura 4.3, y están descritas a continuación:

1. Interfaz con triac (optoacoplada). Un triac es un dispositivo semiconductor con tres electrodos: Entrada, salida y compuerta. Cuando se aplica una corriente a la compuerta del triac, éste permite el paso de corriente eléctrica entre las terminales de entrada y salida del mismo, cuando no hay corriente en la compuerta, el triac se comporta como circuito abierto. El triac está conectado en serie con un foco incandescente y ambos son alimentados por a la corriente alterna de 127[v_{rms}] a 60[Hz] de la red eléctrica comercial en México.

   Para controlar la compuerta del triac, se hace uso del pin 0 del puerto C del MCU, presente en la tarjeta MINICONW_SH32 dedicada a este ejemplo de aplicación. Este pin está conectado a un optoacoplador, para proteger a la tarjeta de la alta tensión que maneja el triac. En este proyecto de tesis, se utilizó el triac BT134, el cual soporta 25[A] amperes de corriente alterna a 600[V] de tensión pico y cuya corriente de compuerta puede ser de 4[mA] hasta 25[mA] (véase la hoja de datos de este dispositivo en [19]).

2. Detector de cruces por cero. Este circuito, genera un pulso de 0.83[ms] de duración cuando la señal de tensión de alterna de 60[Hz], aplicada a la entrada del mismo, toma el valor de 0[V]. La señal del detector de cruces por cero está conectada al pin IRQ de MCU y permite sincronizar el disparo del triac con la señal de alterna. Este circuito está basado en el mostrado en [20]. La intensidad luminosa se lleva a cabo de la siguiente manera:

   Cuando ocurre un cruce por cero, en la interrupción IRQ, se verifica en bajo el pin conectado a la interfaz con el triac para evitar que este último permita la conducción de corriente eléctrica. Además, se habilita la interrupción por conteo de sobreflujos del temporizador del MCU. Esta segunda interrupción se dispara cuando el registro de

sobreflujos del temporizador es igual a otro registro de comparación predefinido y, durante la ejecución de la misma, se verifica en alto el pin conectado a la interfaz con el triac, a partir de este momento el triac conducirá hasta el siguiente cruce con cero. Modificando el valor del registro de comparación predefinido antes mencionado, se varía el tiempo que el triac estará modo de conducción, esto permite modificar el valor eficaz de la señal de corriente alterna, modificando así la intensidad luminosa del foco. En la Figura 4.2 se puede observar el comportamiento antes mencionado.

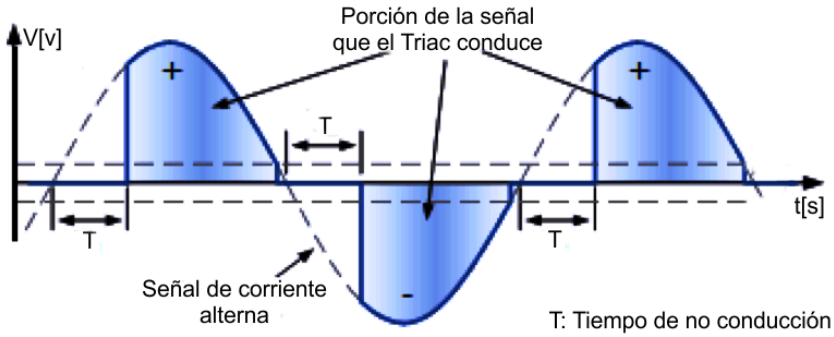
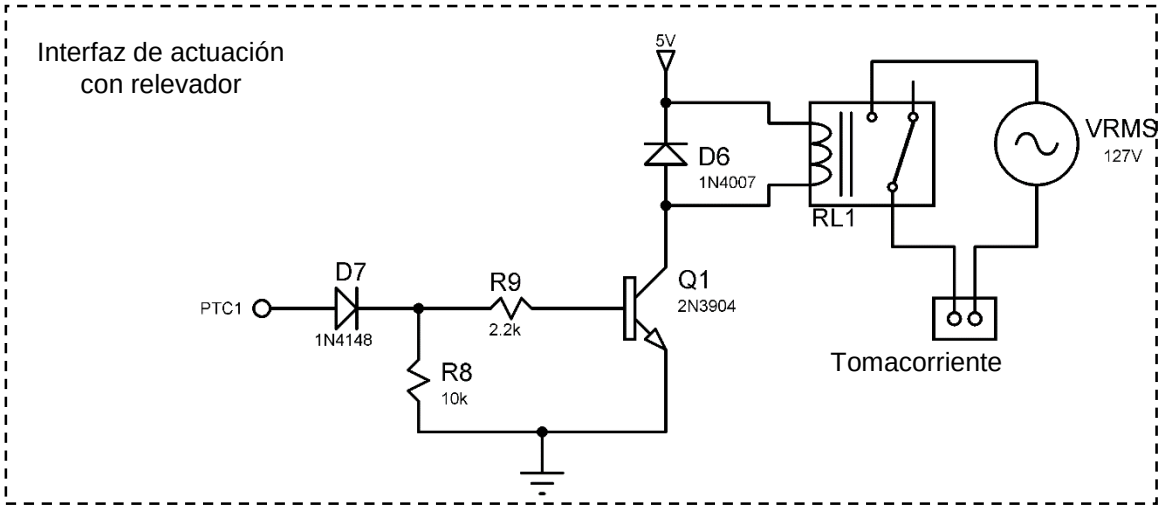


Figura 4.2 Gráfica del intervalo de conducción del triac

3. Interfaz de actuación con relevador. Esta interfaz es un circuito de actuación para un relevador, que incluye un diodo de protección para evitar daños a un transistor, debidos a la sobretensión inducida al desenergizar la bobina. El transistor del circuito permite acoplar la bobina del relevador, que funciona con 5[v] de corriente directa, con un pin del MCU, que trabaja a 3.3[v] de polarización. La terminal normalmente abierta del relevador, empleado en este proyecto de tesis, soporta cargas que demanden hasta 10[A] de corriente alterna o directa y está conectada a un tomacorriente monofásico residencial de uso común en México (ver Figura 4.4).

En las Figura 4.3 y 4.4 se muestran diagrama esquemático del circuito modular y su prototipo correspondientes al ejemplo de aplicación de electrónica de potencia, respectivamente.



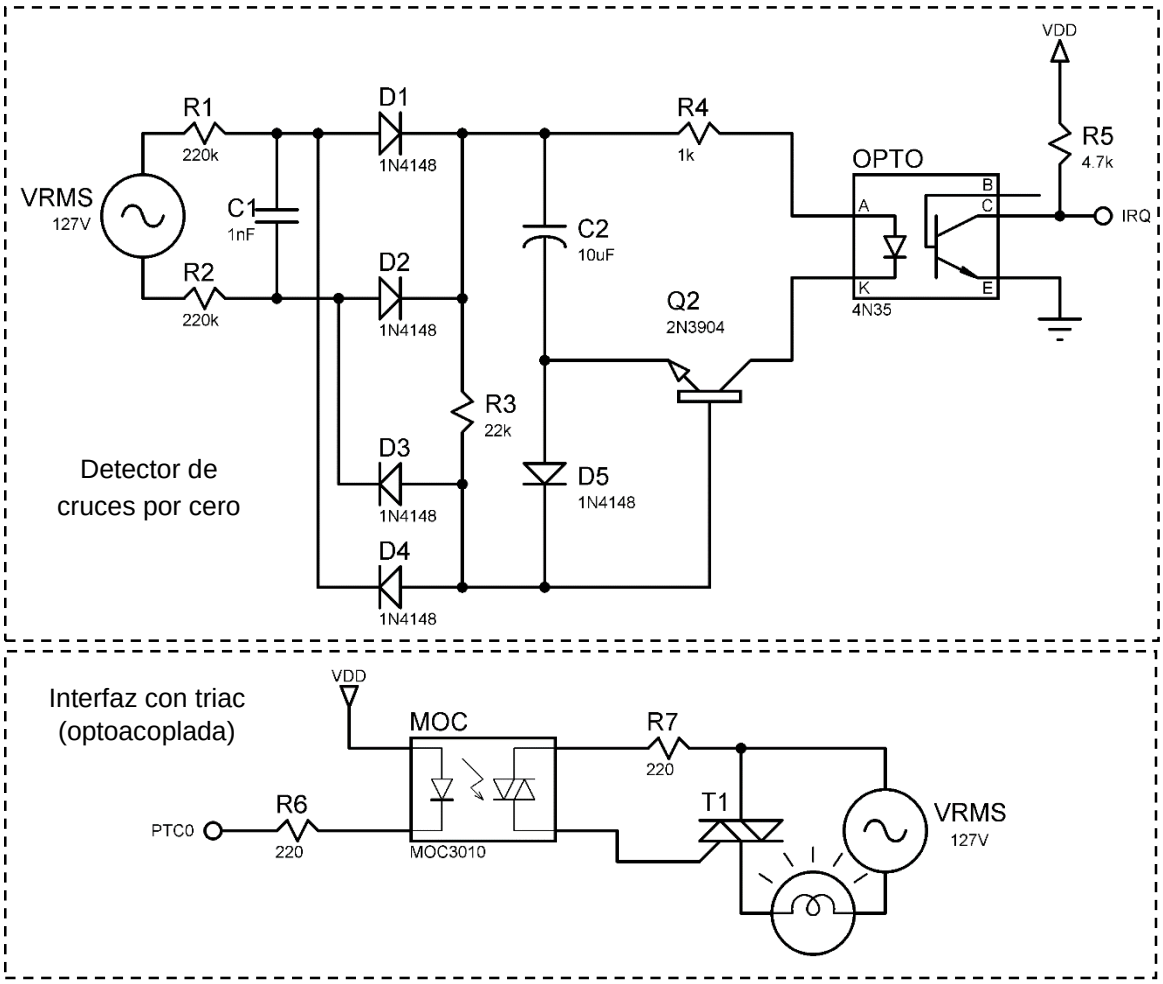
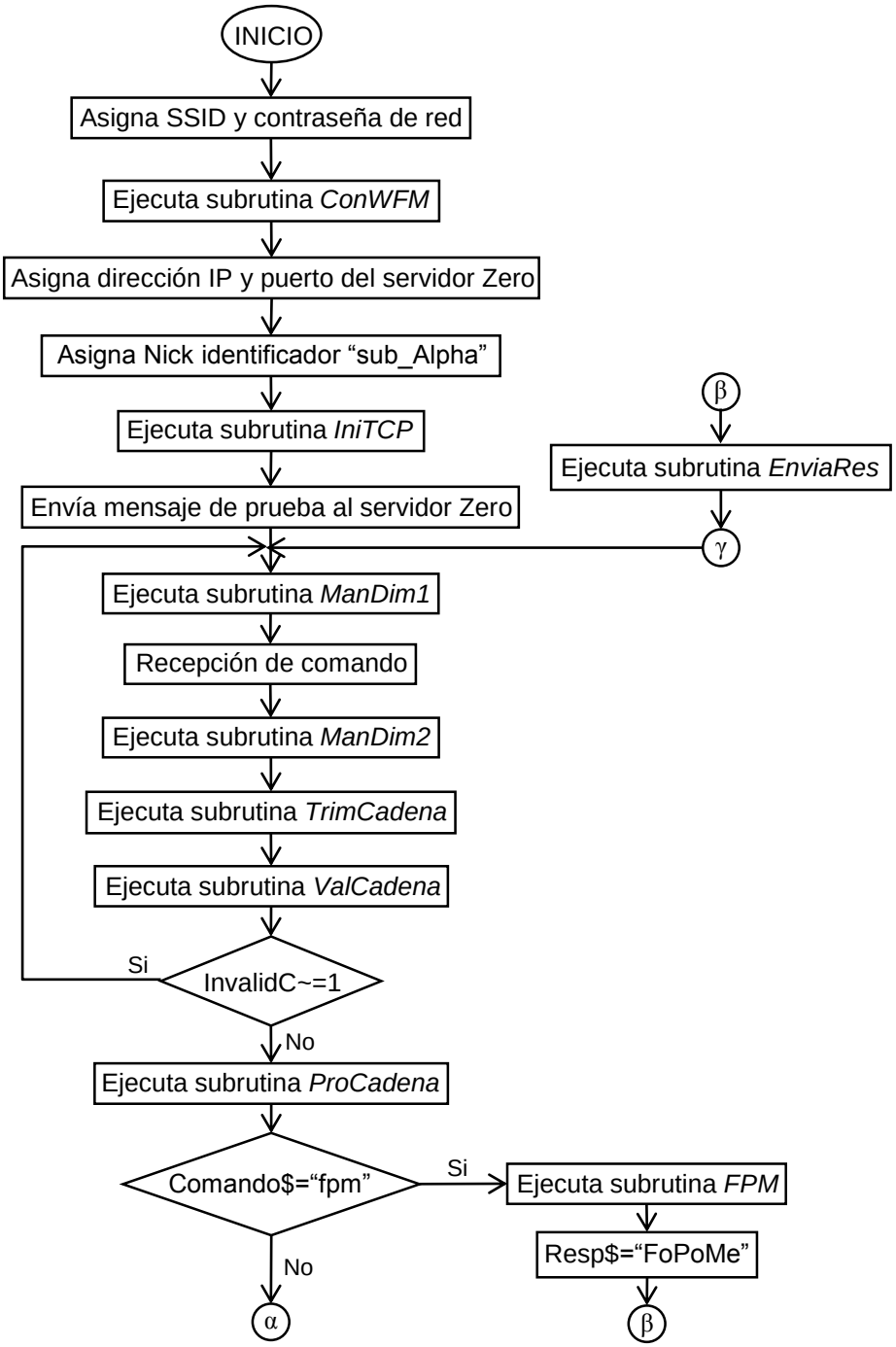


Figura 4.3 Circuito utilizado para el ejemplo de electrónica de potencia

Figura 4.4 Fotografía del prototipo construido

4.1.2 Programa ejecutable por el MCU de la tarjeta MINICONW_SH32

El programa que ejecuta el MCU, presente en la tarjeta MINICONW_SH32 dedicada a este ejemplo de aplicación, tienen el siguiente el flujo de ejecución.



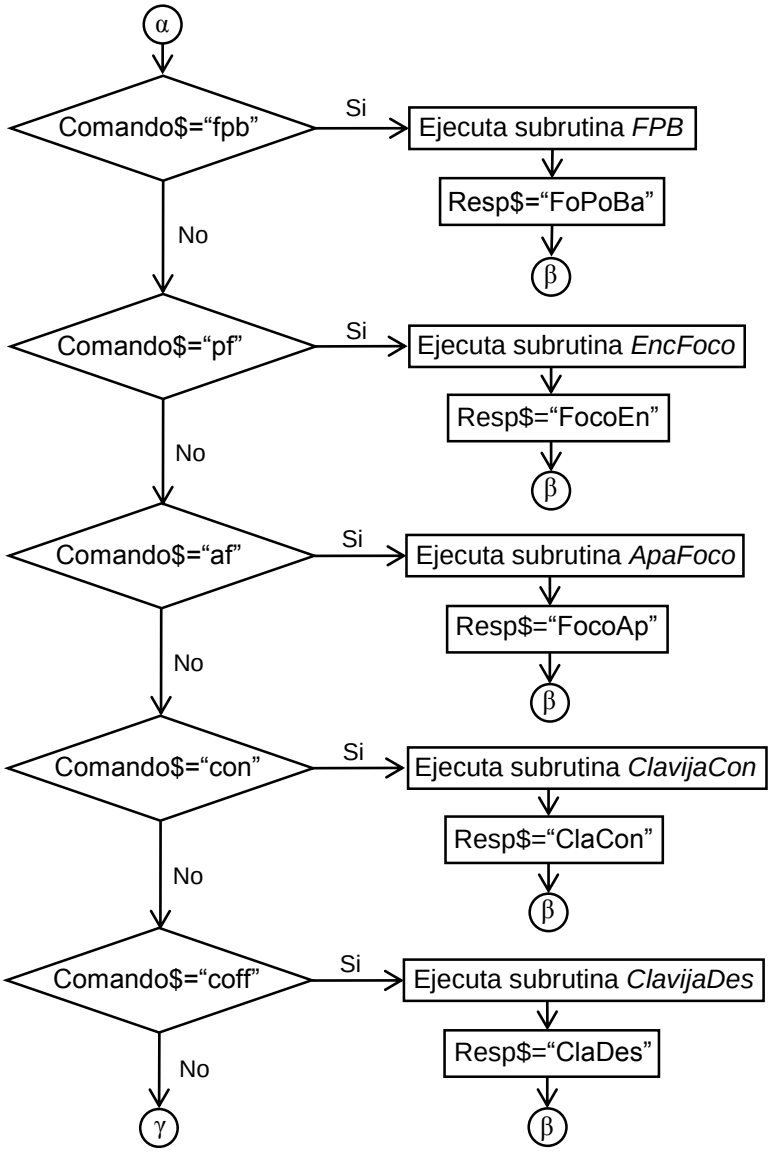


Figura 4.5 Flujo de ejecución del programa para el ejemplo de electrónica de potencia

El programa de este ejemplo de aplicación está basado en el programa plantilla descrito en el capítulo de este documento, por lo que la mayoría de las subrutinas es pertenecen a dicha plantilla, el resto de las subrutinas se describen a continuación.

- *Subrutinas ManDim1 y ManDim2*. Estas subrutinas se utilizan para para evitar errores de recepción de comandos debidos a los retardos causados por las rutinas de interrupción por solicitud externa (IRQ) y de interrupción por conteo de sobreflujos del temporizador del MCU (TofSer), debido a que ambas se ejecutan cada 8[ms]. La subrutina *ManDim1* espera y almacena el primer carácter del comando recibido, y a continuación, deshabilita las rutinas de interrupción. La subrutina *ManDim2* rehabilita las rutinas de interrupción si la variable *DimTr*~=1.

- *Subrutinas FPM y FPB*. En estas subrutinas se modifica el valor del registro de comparación de sobreflujos, modificando así el tiempo de espera para el disparo del triac después del cruce por cero. En el caso de la subrutina *FPM* (potencia media) se establece un tiempo de 4[ms] espera, mientras que en el caso de la subrutina *FPB* (potencia baja) es de 6[ms] para en el caso de la subrutina *FPB*. Esto es así, pues mientras menor sea el tiempo de espera para el disparo del triac, mayor será el tiempo de conducción del mismo y viceversa. Además, en estas subrutinas se asigna *DimTr*~=1, y se limpia la bandera de interrupción global.
- *Subrutinas EncFoco y ApaFoco*. Estas subrutinas asignan *DimTr*~=0, deshabilitan las rutinas de interrupción *IRQSer* y *TofSer* y se modifica el estado lógico del bit 0 del puerto C, el cual está conectado a la interfaz optoacoplada con el triac. Dicho bit, se verifica en bajo, en el caso de la subrutina *EncFoco*, y en alto, en el caso de la subrutina *ApaFoco*.
- *Subrutinas ClavijaCon y ClavijaDes*. En estas subrutinas se cambia el estado lógico del bit 1 del puerto C, el cual está conectado a la interfaz con el relevador. Se verifica en alto en el caso de la subrutina *ClavijaCon*, y en bajo, en el caso de la subrutina *ClavijaDes*.
- *Rutina de interrupción por solicitud externa (IRQSer).* Es disparada al detectar por un flanco de bajada en el pin IRQ del MCU. En esta, se limpia la bandera de interrupción por IRQ y la de sobreflujos del temporizador, reinicia la cuenta actual del registro de sobreflujos del temporizador y se verifica en alto el bit 0 del puerto C.
- *Rutina de interrupción por sobreflujos del temporizador (TofSer).* Es disparada cuando la cantidad de sobreflujos del temporizador 2 es la que se especificó en el registro TPM2MODL. Durante la interrupción se verifica en bajo el bit 0 del puerto C. cabe destacar que no se limpia la bandera local de interrupción por sobreflujos del temporizador, es decir, la rutina *IRQSer* es la que da "permiso" a la rutina *TofSer* de ejecutarse.

4.1.3 Interfaz de usuario

La Interfaz de usuario asociada al ejemplo de electrónica de potencia incluye, por un lado, la interfaz de interacción con el foco, que el usuario puede ejecutar oprimiendo las flechas para aumentar o disminuir la intensidad luminosa del mismo. Por otro lado, contiene la interfaz de accionamiento del relevador, la cual consta de una fotografía de un tomacorriente, el estado se testifica mediante el un led virtual, ubicado arriba de dicha fotografía.

Cabe mencionar que el accionamiento de las flechas tiene un funcionamiento circular, ya que al oprimir la flecha hacia arriba aumenta la intensidad luminosa del foco, pasando del estado de máxima intensidad a estado apagado, y viceversa. La interfaz de usuario se muestra en la siguiente captura.

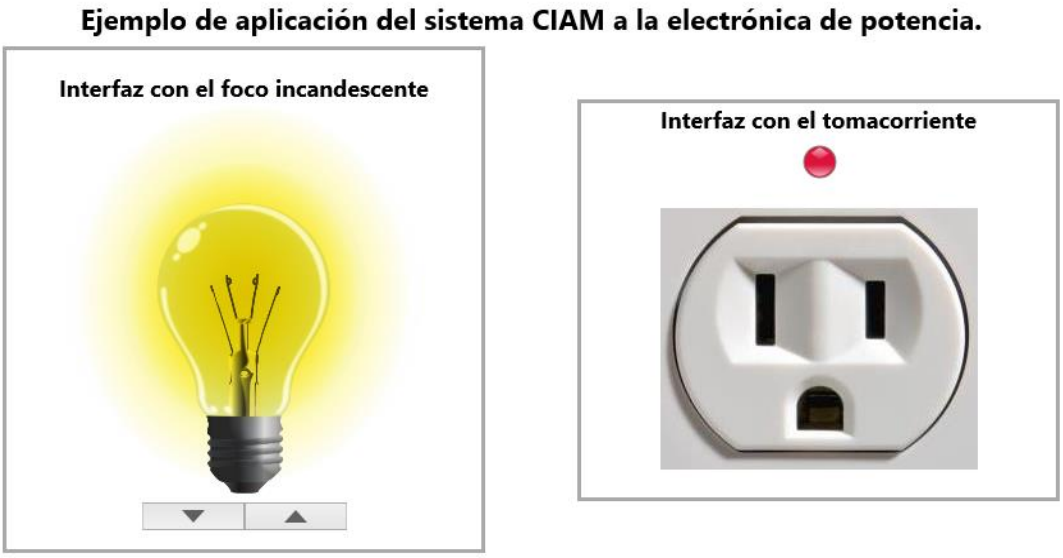


Figura 4.6 Interfaz de usuario del ejemplo Electrónica de potencia

La siguiente tabla muestra las acciones disponibles en la interfaz de usuario

Acción del usuario	Efecto
Pulsar el botón 	Envía mensaje al subordinado para aumentar intensidad
Pulsar el botón 	Envía mensaje al subordinado para disminuir intensidad
Pulsar la imagen del tomacorriente	Envía mensaje al subordinado para cambiar estado del relevador

Tabla 4.1. Posibles acciones dentro de la interfaz de usuario.

Para evitar que el comandante pierda la sincronización con el subordinado, el comandante guarda el estado del foco y del tomacorriente en variables internas, que no se actualizan sino hasta que se recibe respuesta desde el módulo remoto. Por ejemplo, si el foco se encuentra apagado y se oprime el botón para aumentar intensidad, el comandante envía el mensaje "fpb", con lo que el subordinado entenderá que debe ajustar el disparo del triac a una potencia baja, y responderá con el mensaje "FoPoBa". Luego entonces en la interfaz de usuario se mostrará una imagen representando a un foco con baja luminosidad.

Dicho de otra manera, el mensaje que envíe el comandante, de acuerdo con la tabla 4.1, estará en función del valor de las variables internas que guardan el estado del foco o del tomacorriente, según sea el caso. Estas variables internas, se actualizan a la respuesta que recibe el *comandante* desde el *subordinado.*

4.2 Ejemplo de aplicación orientado a la instrumentación virtual

En este ejemplo se muestra, de una manera simple, el potencial del sistema CIAM para el monitoreo de variables. Las acciones ejecutadas por el módulo de instrumentación virtual pueden ser fácilmente extrapoladas al ámbito industrial, ya que muchos de los procesos incluyen el monitoreo de variables tanto digitales como analógicas. Por ejemplo, la lectura de sensores o transductores, así como el estado lógico de una compuerta. Asimismo, se puede realizar el accionamiento de distintos procesos mediante salidas digitales, como puede ser el arranque de un motor, el cierre o apertura de una electroválvula, o cualquier proceso que pueda ser ejecutado por un microcontrolador.

4.2.1 Circuito modular

El circuito dedicado a este ejemplo de aplicación, cuyo esquemático se muestra en la Figura 4.7, consta de tres partes principales:

1. Entradas digitales: Se destinó la primera mitad de los bits del puerto C del MCU para servidor de entradas digitales mediante un interruptor DIP.
2. Entradas digitales: El resto de los pines del puerto C se destinó como salidas digitales conectadas a leds.
3. Entrada analógica: Se utilizó un potenciómetro para ejemplificar el funcionamiento del monitoreo de una señal analógica, conectado al canal 0 del convertidor analógico digital del MCU.

A continuación, se muestra el circuito utilizado para el ejemplo de instrumentación virtual.

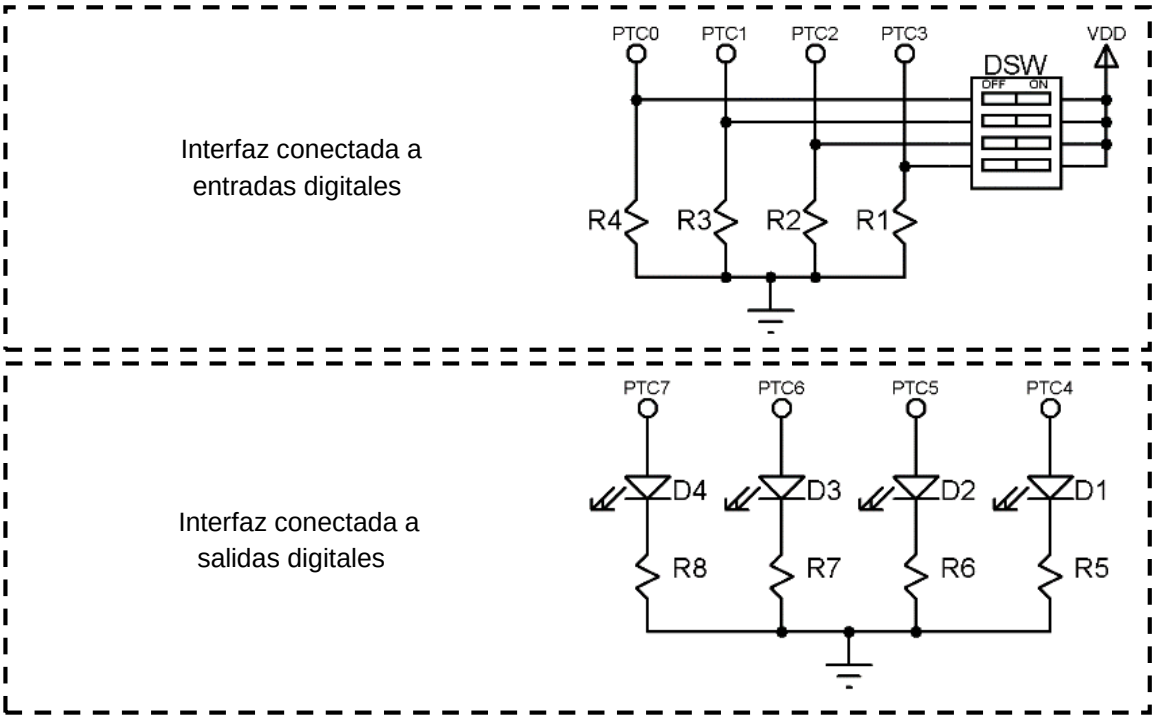


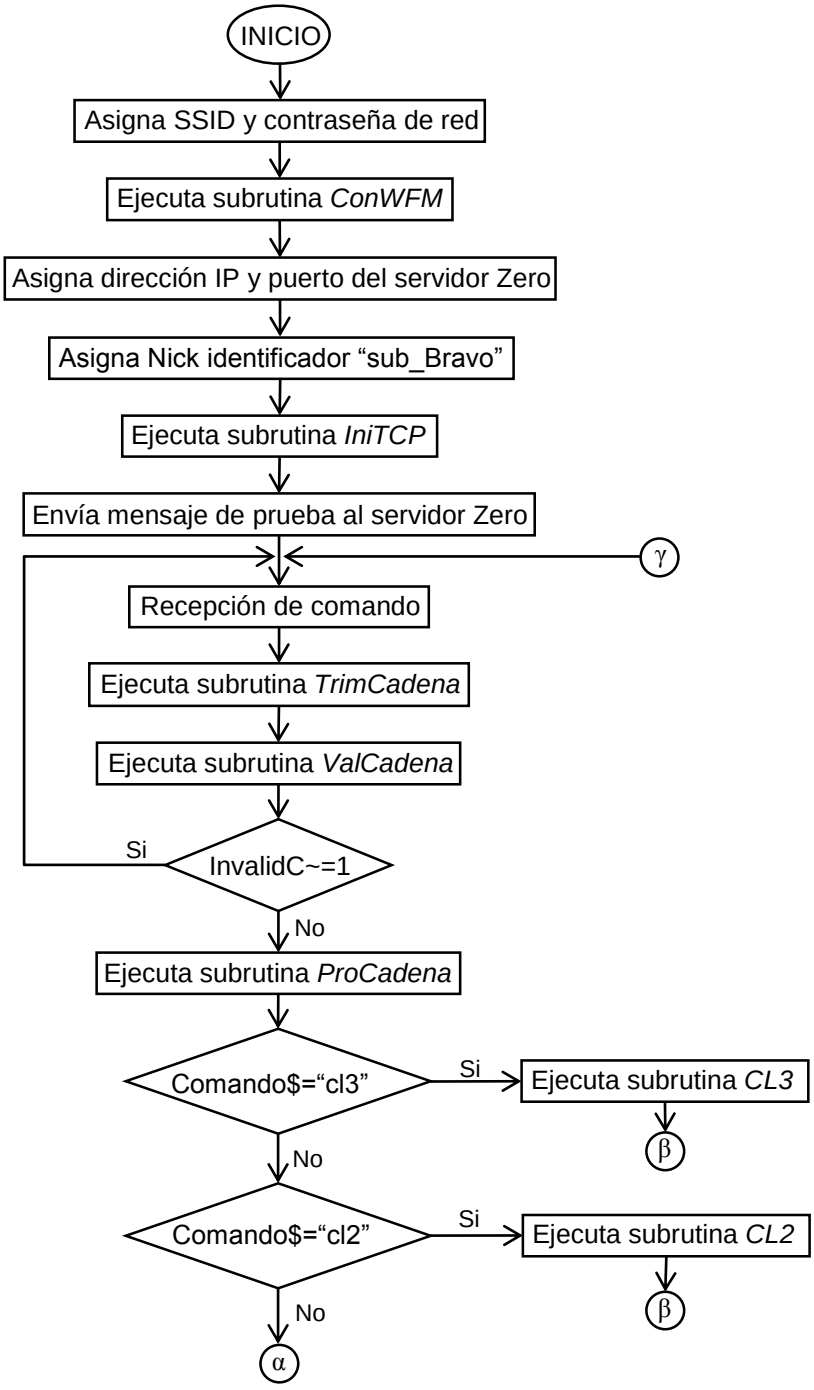
Figura 4.7 Circuito utilizado para el ejemplo de instrumentación virtual

Se construyó el prototipo mostrado en la siguiente figura. En ésta, se muestra como el circuito modular embona perfectamente en la interfaz de pines de la tarjeta de desarrollo MINICONW_SH32.

Figura 4.8 Prototipo asociado al ejemplo de instrumentación virtual

4.2.2 Software ejecutable por el MCU de la tarjeta MINICONW_SH32

El programa que ejecuta el MCU para este ejemplo se describe con ayuda del siguiente diagrama de flujo.



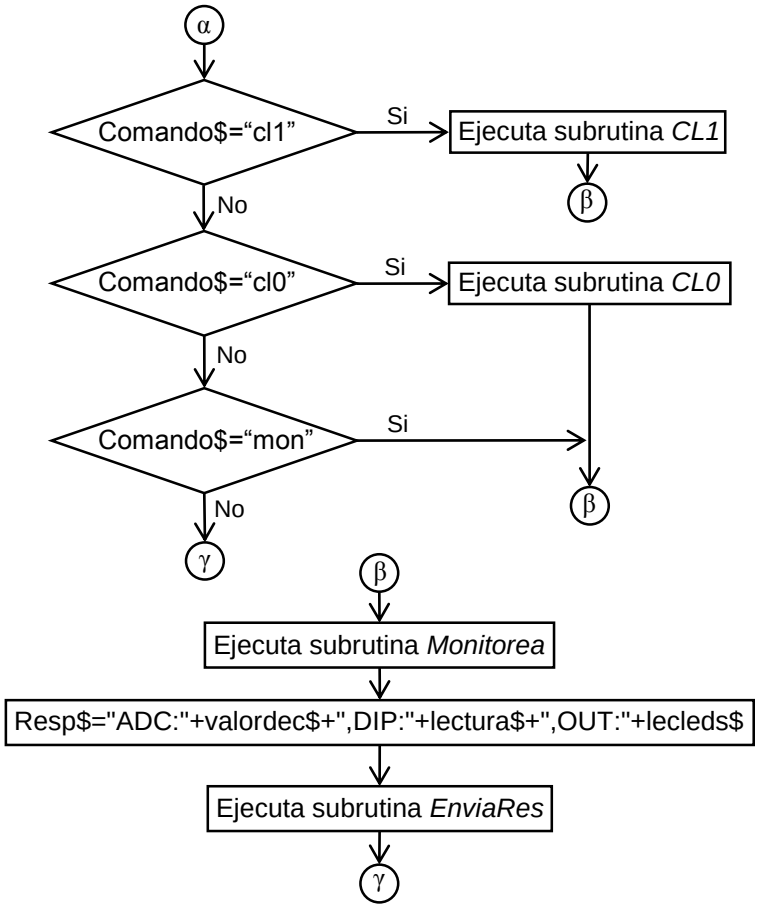


Figura 4.9 Flujo de ejecución del programa para el ejemplo de instrumentación virtual

Las subrutinas de aplicación utilizadas en este ejemplo se explican a continuación.

- *Subrutina CL3*. Lee el estado del puerto C, se obtiene el estado del bit más significativo de dicho puerto y a continuación se cambia a su estado lógico contrario.
- *Subrutina CL2*. Lee el estado del puerto C, se obtiene el estado del bit 6 de dicho puerto y a continuación se cambia a su estado lógico contrario.
- *Subrutina CL1*. Lee el estado del puerto C, se obtiene el estado del bit 5 de dicho puerto y a continuación se cambia a su estado lógico contrario.
- *Subrutina CL0*. Lee el estado del puerto C, se obtiene el estado del bit 4 de dicho puerto y a continuación se cambia a su estado lógico contrario.
- *Subrutina Monitorea*. Realiza la lectura del valor de los cuatro bits más significativos del puerto C, del canal 0 del convertidor analógico digital, de los cuatro bits menos significativos del mismo puerto, y las almacena en las variables *lecleds$*, *valordec$* y *lectura$*, respectivamente. Hace uso de las subrutinas *conv8bitbas* y *convadec*.

- *Subrutina conv8bitbas*. Se realiza el proceso para obtener la lectura del canal 0 del convertidor analógico digital del MCU, otorgando un valor con resolución de 8 bits y almacenándolo en la variable de tipo byte *valconv~*.
- *Subrutina convadec*. Convierte un byte en formato hexadecimal a decimal y almacena el valor en la variable *valordec$*. Hace uso de la subrutina *letra*.
- *Subrutina letra*. Cambia el valor de una letra en hexadecimal a su valor correspondiente en decimal y lo almacena en la variable *v%*.

4.2.3 Interfaz de usuario

Este programa envía cada 500 [ms] el comando que permite monitorear el estado de los componentes conectados al módulo remoto. El módulo remoto responde cada solicitud con el estado actual del mismo, según el cual la aplicación de usuario actualiza los valores en su interfaz gráfica. Esta interfaz cuenta con una interfaz de accionamiento de cuatro salidas digitales localizadas en el módulo remoto, así como la interfaz de monitoreo de cuatro entradas digitales y una entrada analógica.

La interfaz de salidas digitales corresponde a una interfaz caracterizada por un DIP switch virtual, donde el accionamiento de cada switch individual produce el cambio en el estado lógico de un led presente en el módulo remoto. La interfaz de monitoreo de entradas digitales consta de cuatro leds virtuales que testifican el estado lógico de un DIP switch real, presente en el circuito del módulo remoto. Finalmente, la lectura del convertidor analógico-digital, se aprecia en un voltímetro virtual, el cual simula un voltímetro analógico real.

La interfaz de usuario se muestra en la siguiente figura.

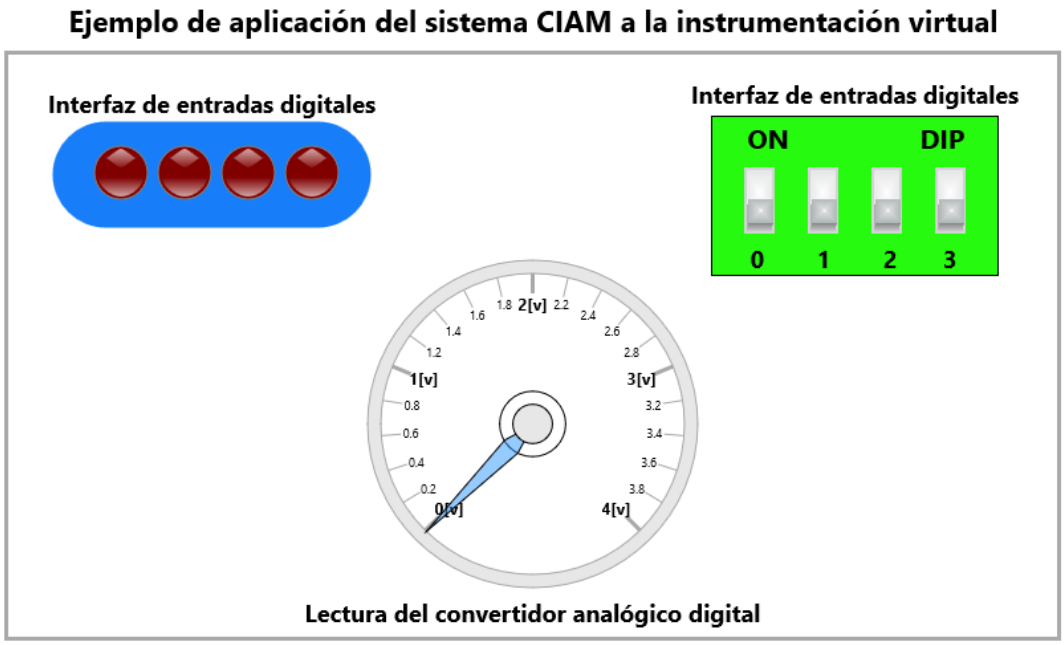


Figura 4.10 Interfaz de usuario asociada al ejemplo de instrumentación virtual

4.3 Ejemplo de aplicación orientado a la automatización del hogar

En este ejemplo se aprovecharon las características compactas de la tarjeta MINICONW_SH32, para utilizarla como sistema embebido en un área del hogar, para automatizar la iluminación exterior, así como para el monitoreo del estado de alguna puerta o ventana. El objetivo de este ejemplo es demostrar el funcionamiento, sin intervención necesaria de un usuario, del sistema CIAM.

Se construyó un aparato en el que se incluye la tarjeta MINICONW_SH32 y un circuito modular compatible con ésta. Dicho circuito posee un reloj en tiempo real (RTC, por sus siglas en inglés) y un sensor de movimiento, que dispara la interrupción por solicitud externa en el MCU de la tarjeta, el cual automáticamente realiza la lectura de la hora actual del RTC, luego, realiza una comparación de este dato con un rango de horas predefinido, que se supone abarca las horas de oscuridad promedio, en un día normal en la ciudad de México. Si la hora obtenida pertenece a este rango, se encenderá una lámpara.

Por otro lado, se incluyó un sensor magnético que se puede acoplar a una puerta o ventana, que funciona como un simple interruptor cerrado o abierto, del cual el MCU puede verificar el estado cuando el usuario lo solicite.

4.3.1 Circuito modular

Este circuito contiene, los componentes necesarios para el correcto funcionamiento del RTC, así como la interfaz requerida para el envío de los datos correspondientes a la hora a través del módulo IIC del MCU.

Para este ejemplo, se utilizó el chip DS1307 como reloj en tiempo real, la hoja de datos se puede consultar en [21], Este chip se escogió por su bajo consumo, facilidad de uso y confiabilidad a largo plazo. Es un circuito integrado que está diseñado para proporcionar, en un código binario decimal (BCD), la hora exacta y fecha actual. Está diseñado para mantener el registro horario hasta el año 2100, contemplando los ajustes para el año bisiesto, así como el número de días de cada mes. Cuenta con 56 bytes de memoria RAM, de los cuales los primeros 8 están destinados a almacenar la hora, la fecha y el valor de frecuencia de una señal cuadrada, la cual se ve reflejada en uno de los pines del chip.

Se incluye, además, la interfaz a un sensor magnético cuyo estado se verifica utilizando un pin del puerto B del MCU, así como el sensor de movimiento, cuya salida activa la interrupción por solicitud externa del MCU. El elemento sensor se denomina sensor infrarrojo pasivo (PIR, por sus siglas en inglés). Es un sensor electrónico que registra los cambios de radiación de luz infrarroja en su campo de detección. Se acompaña de un lente de Fresnel, un lente compacto y de material liviano que permite aumentar la amplitud, rango y patrones de sensibilidad. Asimismo, el circuito contiene la interfaz opto acoplada para la actuación de una lámpara, mediante un triac.

El circuito descrito anteriormente se muestra a continuación.

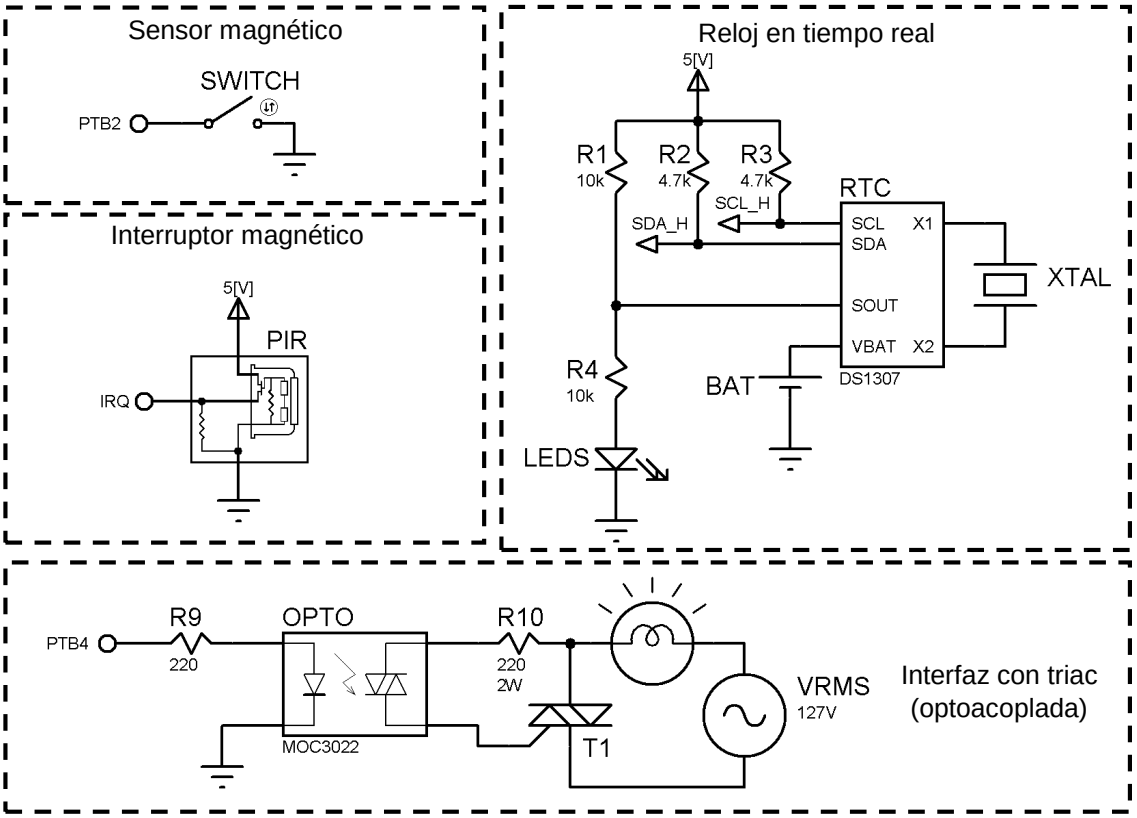


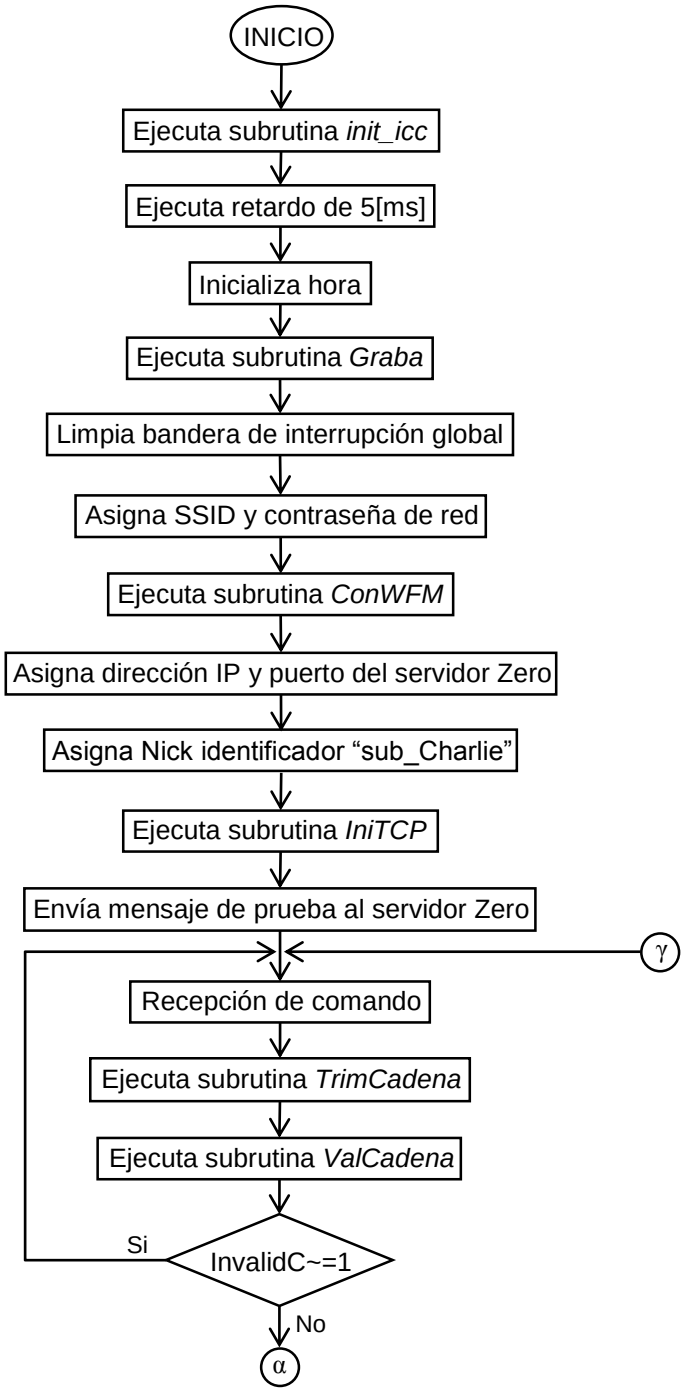
Figura 4.11 Circuito utilizado para el ejemplo de automatización del hogar

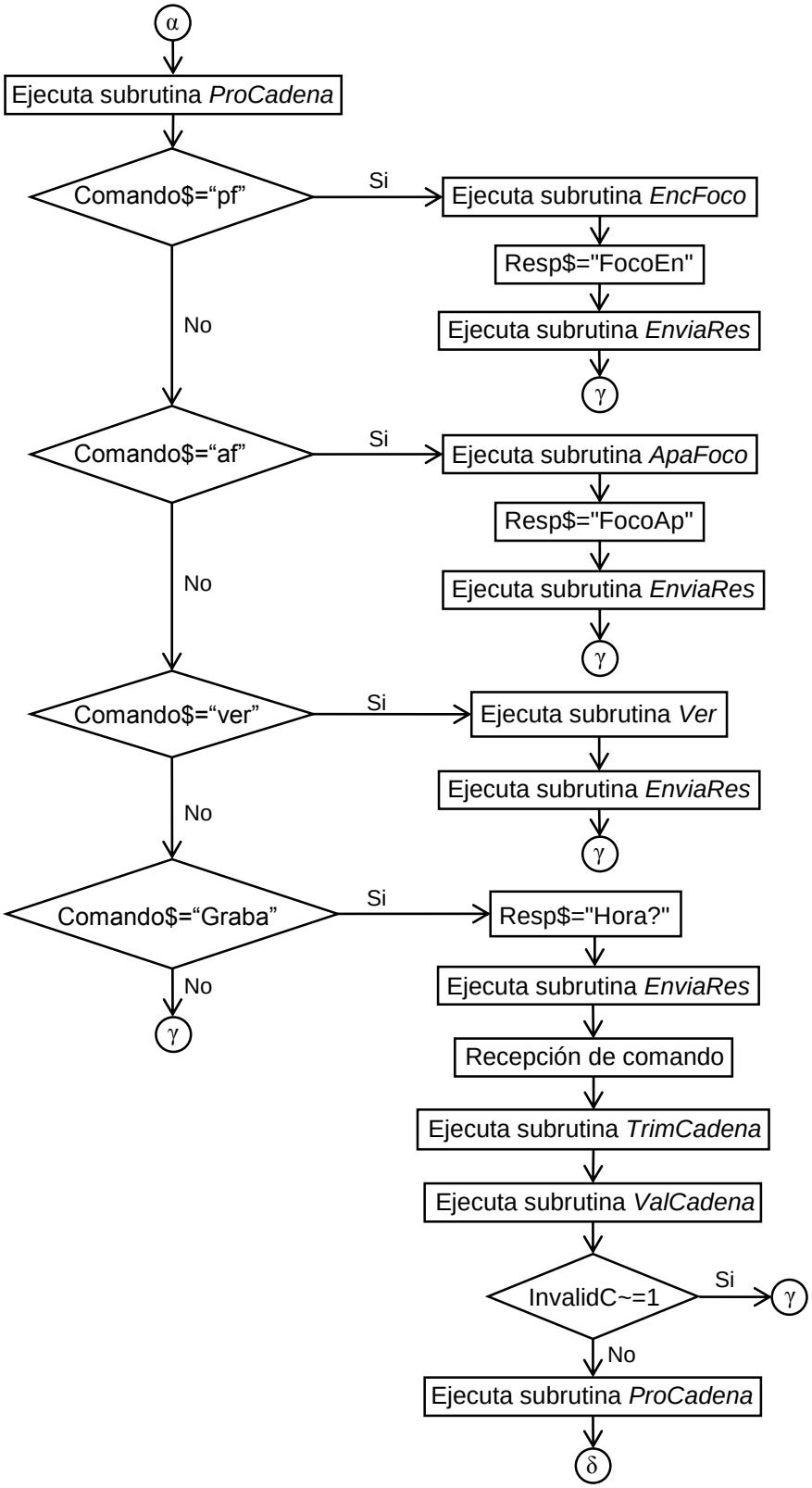
Se construyó el siguiente un prototipo para demostrar el funcionamiento ejemplo de automatización del hogar.

Figura 4.12 Fotografía del prototipo asociado al ejemplo de automatización del hogar

4.3.2 Software ejecutable por el MCU de la tarjeta MINICONW_SH32

El programa que ejecuta el MCU para este ejemplo se describe en el siguiente diagrama de flujo.




α
Ejecuta subrutina *ProCadena*
Comando$=“pf”
Si
Ejecuta subrutina *EncFoco*
Resp$="FocoEn"
Ejecuta subrutina *EnviaRes*
γ
No
Comando$=“af”
Si
Ejecuta subrutina *ApaFoco*
Resp$="FocoAp"
Ejecuta subrutina *EnviaRes*
γ
No
Comando$=“ver”
Si
Ejecuta subrutina *Ver*
Ejecuta subrutina *EnviaRes*
γ
No
Comando$=“Graba”
Si
Resp$="Hora?"
Ejecuta subrutina *EnviaRes*
Recepción de comando
Ejecuta subrutina *TrimCadena*
Ejecuta subrutina *ValCadena*
InvalidC~=1
Si
γ
No
Ejecuta subrutina *ProCadena*
δ
No
γ

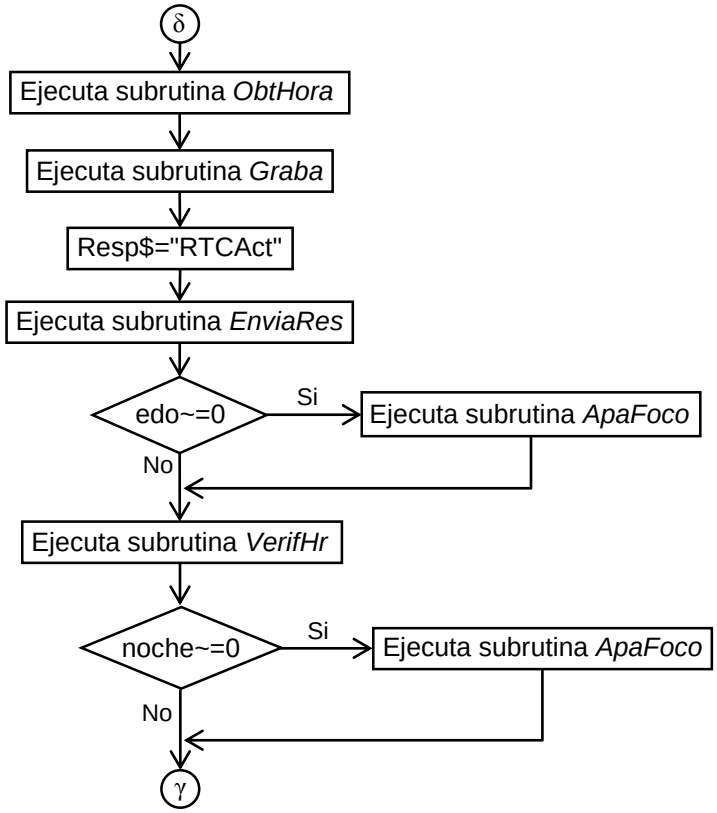


Figura 4.13 Flujo de ejecución para el ejemplo de automatización del hogar

Las subrutinas de aplicación utilizadas en este ejemplo se explican a continuación.

- *Subrutina ApaFoco*. Verifica en bajo el pin 4 del puerto B, asigna edo~=0 y limpia la bandera de interrupción por solicitud externa (IRQF).
- *Subrutina EncFoco*. Verifica en alto el pin 4 del puerto B y asigna edo~=1.
- *Subrutina Ver*. Lee el estado del puerto B, enmascara la lectura para obtener el valor del pin 2, y lo almacena en la variable switch~, según el cual se almacena en la variable Resp$, "Cerrada" en caso de que el valor de switch~ sea 0, y "Abierta" en caso contrario.
- *Subrutina ObtHora*. Obtiene los valores correspondientes a la hora, minutos y segundos, almacenados en la cadena de la variable Comando$. Cada dato se almacena por separado en una variable tipo byte. Esta subrutina hace uso de la subrutina *str2bcd*.
- *Subrutina str2bcd*. Convierte un dato tipo *string*, de dos caracteres numéricos a formato BCD.
- *Subrutina LeeHora*. Realiza la lectura de la lista de bytes correspondientes a la hora actual de un RTC conectado al módulo IIC del MCU. Para este ejemplo, se almacenan únicamente los datos correspondientes a la hora, minutos y segundos. Hace uso de las

subrutinas *iic_start, iic_outb, iic_stop, iic_getbyte* las cuales estan basadas en las surutinas docuemntdas en presentadas en [22].

- *Subrutina Graba*. Realiza el grabado de la lista de bytes correspondientes a la fecha y hora actuales en un RTC conectado al módulo IIC del MCU. Hace uso de las subrutinas *iic_start, iic_outb, iic_stop* y algunas subrutinas de retardo.
- *Subrutina init_icc*. Inicializa el módulo IIC del MCU con baud rate de 100 [kbps], a una frecuencia de bus de 20 [MHz], como esclavo y establece que el master (el MCU) no genera señal de *acknowledge* (ACK).
- *Subrutina iic_start*. Envía señalización de *start* al bus, para iniciar una transmisión o lectura de datos. Esta subrutina permite asegurar si se envió la señal de *start* con éxito, o si el bus está ocupado, y por lo tanto no se puede iniciar una transmisión.
- *Subrutina iic_outb*. El master transmite un byte (después de que la señal de *start* se ha enviado). Asimismo, permite asegurar una transmisión exitosa si se recibió señal ACK del esclavo receptor, o si la transmisión falló y no se recibió señal ACK del esclavo receptor.
- *Subrutina iic_stop*. Envía al bus IIC la señal de *stop*, requerida para delimitar una transmisión de otra, o una transmisión de una lectura de datos. Posteriormente, el master pasa a modo esclavo.
- *Subrutina iic_getbyte*. Ejecuta la lectura de un byte en la memoria del esclavo. Incluye una señal de *start*, verificación mediante el bit de control, recepción y señal de *stop*. Hace uso de la subrutina *iic_inb* para iniciar la lectura de un byte de datos.
- *Subrutina iic_inb*. El master recibe un byte desde un esclavo, después de una señal de *start*. Si la bandera de acarreo es igual a cero (C=0), el byte fue recibido exitosamente, de lo contrario, se presentó un error y se perdió el arbitraje del bus.
- *Rutina de interrupción IRQSer*. Al activarse, limpia la bandera de interrupción por solicitud externa (IRQ), y si edo~=1, sale de la rutina. Invoca la subrutina *VerifHr*, y si noche~=1, se verifica en alto el bit 4 del puerto B e inicia el proceso para activar la interrupción por sobreflujos del temporizador (TOVF), en el cual se limpia la bandera de interrupción del TOVF y se reinicia el contador de TOVF (registro TMP2CNT). Finalmente, se desactiva la interrupción IRQ.
- *Subrutina VerifHr*. Invoca la subrutina *LeeHora* y compara la hora obtenida con un rango de valores entre 19 y 23, y entre 00 y 06. Si el valor está dentro de cualquiera de estos rangos, se asigna noche~=1.
- *Rutina de interrupción TofSer*. Genera un intervalo de 2 segundos e invoca la subrutina *Contador*.
- *Subrutina Contador*. Si edo~=1, sale de la subrutina, de lo contrario, el contador cont2~ se incrementa. Al alcanzar el valor cont2~=5, se verifica en bajo el pin 4 del puerto B y se activa la interrupción IRQ.

4.3.3 Interfaz de Usuario

En este programa se muestra la interfaz para actualizar la hora del reloj en tiempo real (RTC) presente en el módulo remoto, cada vez que la actualización del RCT se realice satisfactoriamente, se alternará el color de un LED virtual, presente en la misma interfaz de usuario. Se incluyen también, un icono que permite verificar el estado de un sensor magnético puerta y un par de botones que se usan para el encendido o apagado de una lámpara. La lámpara, sensor magnético y el RTC forman parte del hardware conectado a la tarjeta MINICONW_SH32, dedicada a este ejemplo de aplicación. En la siguiente figura se muestra la interfaz de usuario aquí descrita.

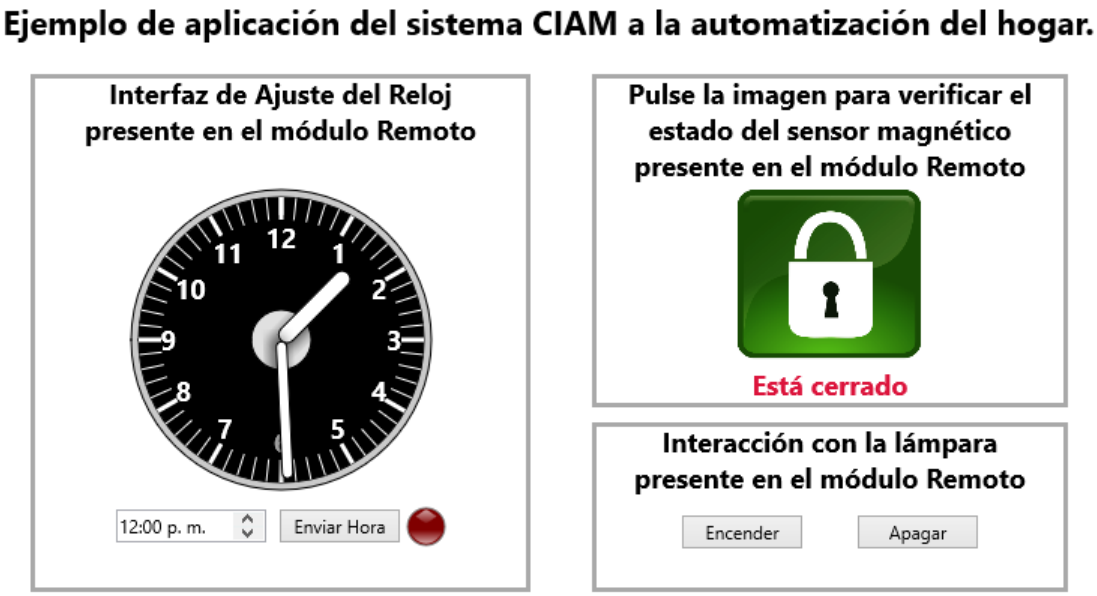


Figura 4.14 Interfaz de usuario asociada al ejemplo de automatización del hogar

La siguiente tabla muestra las acciones disponibles en la interfa de usurio:

Acción del usuario	Efecto
Clic en la imagen del reloj.	Envía mensaje al módulo remoto que actualizará la hora en el reloj del mismo, según la hora actual.
Clic en el botón “Enviar Hora”.	Envía mensaje al módulo remoto que actualizará la hora en el reloj del mismo, según la hora mostrada en la caja de texto adyacente al botón.
Clic en la imagen del candado.	Envía mensaje para solicitar el estado de la puerta.
Clic en el botón “Encender”	Envía mensaje para encender lámpara controlada por el módulo de actuación remoto.
Clic en el botón “Apagar”	Envía mensaje para Apagar lámpara controlada por el módulo de actuación remoto.

Tabla 4.2 Posibles acciones dentro de la interfaz de usuario.

CONCLUSIONES Y TRABAJO FUTURO

El uso de la tecnología inalámbrica permite aumentar la versatilidad de tareas, reduciendo la necesidad de cables físicos, que limitan la movilidad y expansibilidad de los proyectos. Si bien existen diversas vías de comunicación inalámbrica, el uso de la tecnología Wifi es un método muy utilizado hoy en día, la mayoría de los dispositivos incluyen un módulo Wifi interno, lo que permite que el sistema CIAM pueda incurrir en una amplia gama de procesos, tanto industriales, como domésticos, e incluso en el área de la salud.

El monitoreo de diversos indicadores, mediante transductores y sensores, de manera no invasiva, puede marcar la diferencia en el tratamiento y recuperación de pacientes cuya salud física debe estar en constante chequeo. Haciendo uso del sistema CIAM se puede aumentar la calidad de vida de estas personas, pues la limitación en la movilidad debido a cables conectados a su cuerpo presenta una incomodidad generalizada que hoy en día se puede suprimir. Asimismo, la medición registrada por el módulo remoto ubicado cerca del paciente, puede llegar rápidamente, vía Internet, al médico encargado del tratamiento, y se pueda realizar un diagnóstico preliminar o evitar una complicación mayor debida a la falta de información.

El uso de una red de comunicación TCP/IP basada en el direccionamiento por direcciones IP, permite que el sistema CIAM forme parte de la red de redes, y por lo tanto se puedan intercambiar datos desde cualquier lugar del mundo mediante un dispositivo conectado a Internet. No obstante, al ser parte de un grupo tan vasto de dispositivos conectados, pueden presentarse situaciones en las que otros usuarios conectados puedan hacer uso de la información transmitida a través de la red. Por tanto, es necesario incluir algoritmos de seguridad y encriptación de datos, para que los datos intercambiados dentro del sistema CIAM no se vean alterados, así como para proteger la integridad de los dispositivos conectados a la red.

El diseño de una tarjeta de desarrollo, con las facilidades necesarias para interactuar con sus componentes principales es una herramienta muy útil para la creación de muchos proyectos basados en microcontrolador. La característica de que contenga una fuente de poder interna reduce el número de módulos aislados que se requiere conectar, así como el tamaño que ocupa el módulo de actuación. No obstante, la tarjeta de desarrollo MINICONW_SH32 puede mejorarse, en cuanto a sus capacidades y su estructura, así como en su apariencia, de manera que sea más accesible a usuarios que no estén familiarizados con el circuito.

La facilidad de comunicación que permite el sistema CIAM con el MCU, a través del puerto serie permite que este se pueda programar de manera inalámbrica. Actualmente, los MCU de tipo Chipbas8, incluidos dentro de las tarjetas de la serie MINICON, se programan mediante la interfaz serial, utilizando el ambiente de desarrollo PUMMA_08+. A través del sistema CIAM se podrán programar inalámbricamente, sin necesidad de retirarlos de su entorno de ejecución, ya que se seguiría el mismo paradigma de envío de bytes, correspondientes al

programa, por el puerto serie, pues el módulo Wifi utilizado en este trabajo funciona como si de un cable serial se tratara.

El diseño de una biblioteca de subrutinas es un recurso útil tanto para expandir los proyectos cuya comunicación está limitada a una conexión física, como para estructurar nuevos diseños, ya que conforma una base sobre la cual se puede trabajar para dichos objetivos. Sin embargo, se puede aumentar la versatilidad de esta biblioteca, permitiendo al programador utilizar los recursos de la biblioteca bicoesp8266 mediante funciones y procedimientos. Actualmente, se encuentra en desarrollo una nueva versión del compilador cruzado empleado en este trabajo, que soporta el uso de funciones y procedimientos, el cual permitirá evolucionar la realización de programas como los presentados en este trabajo, haciendo que el sistema CIAM sea más competitivo en el mercado actual.

El entorno de desarrollo RAD Studio, para aplicaciones programadas en lenguaje Object Pascal, permite la realización de programas de aplicación mediante componentes, accesibles a los programadores, que además contiene las herramientas que dichos programas sean ejecutables en distintos dispositivos, con diferentes sistemas operativos, como son Windows, IOS y Android. Esta herramienta hizo posible la realización de aplicaciones de usuario que interactúan dentro del sistema CIAM como centro de mando para dirigir y monitorear diversas acciones.

APÉNDICE A. Información general sobre Internet.

El presente apéndice, es una lista de conceptos clave utilizados para la explicación del proyecto de esta tesis. Dichos conceptos han sido recopilados de varios foros de discusión en internet y del libro Indy in Depth.

- *Protocolo.*

Un protocolo puede definirse como un conjunto de reglas que rigen el comportamiento de algún procedimiento. Específicamente, es el conjunto de reglas que rigen el intercambio de información entre dos entidades de un sistema, conectadas entre sí.

En términos de *TCP/IP*, es un conjunto de guías generales que permite a un equipo comunicarse dentro de una red. En general, se refiere a una de estas dos cosas:

a) El tipo de *socket de Internet*. En este caso, el protocolo especifica de qué tipo de socket se trata, los tipos de socket más comunes son los sockets *TCP*, *UDP* e *ICMP*.
b) Un protocolo de comandos de aplicación. Se refiere a los comandos y respuestas que se utilizan para realizar ciertas funciones, algunos de estos protocolos son *HTTP* (páginas web), *FTP* (transferencia de archivos) y *SMTP* (correo electrónico).

- *Socket de Internet.*

También llamado *socket TCP/IP*, un *socket* es el conducto virtual de comunicaciones que se crea entre dos procesos, los cuales pueden ser locales (ejecutarse en el mismo dispositivo) o bien de manera remota. Un *socket* es muy parecido a una llamada telefónica, para llevar a cabo una conversación primero se debe hacer la llamada y la otra parte debe contestar, de otro modo la conexión (es decir el *socket*) no podrá establecerse.

- *Protocolo TCP (Transmission Control Protocol).*

El Protocolo de Control de Transmisión es uno de los protocolos fundamentales en Internet y uno de los más importantes de la familia *TCP/IP*. Junto con el protocolo *UDP*, es el protocolo de transporte de datos más utilizado. La mayoría de los servicios de una red *TCP/IP* utilizan *TCP* para el transporte de datos. *TCP* es un protocolo basado en conexión, y para funcionar primero se debe establecer una conexión con un servidor. Este protocolo garantiza la entrega acertada de datos enviados y recibidos en una conexión, y además garantiza que los datos llegarán en el orden que fueron enviados.

- *Protocolo UDP (User Datagram Protocol).*

Es un protocolo basado en el intercambio de paquetes de datos enteros, no con bytes individuales, como *TCP*. Estos paquetes, denominados datagramas, incorporan suficiente información de direccionamiento en su cabecera que permite el envío de dichos datagramas a través de la red, sin que exista una conexión previa, por lo que este protocolo puede operar sin que el origen y el destino estén sincronizados. Se distingue por no tener confirmación de que los paquetes lleguen a su destino correctamente, ni control de flujo, lo que permite que los paquetes alcancen su destino en un orden distinto al que fueron enviados.

❖ *Familia de protocolos de Internet.*

Es un conjunto de protocolos de comunicación en los que se basa Internet y que permiten la transmisión de datos entre diversos dispositivos. En la práctica, a este conjunto se le denomina conjunto de protocolos *TCP/IP*, en referencia a los dos protocolos más importantes que lo componen, y que son los más utilizados del conjunto. Estos son: *TCP* (*Transmission Control Protocol*), protocolo de control de transmisión e, *IP* (*Internet Protocol*), protocolo de Internet.

❖ *Redes TCP/IP*.

Una red *TCP/IP* es una red de dispositivos electrónicos que se conectan entre sí, siguiendo las reglas de la familia de protocolos *TCP/IP*. Las redes *TCP/IP* están clasificadas en tres grupos:

1. Redes *LAN*: De inglés *Local Area Network*, las *LANs* son redes de área local que normalmente se limitan a espacios pequeños, una casa, un departamento o un edificio.
2. Redes *MAN*: De inglés *Metropolitan Area Network*, las *MANs* son las redes más grandes que corresponden a una ciudad entera. Podemos decir que una *MAN* es una red de *LANs*.
3. Redes *WAN*: De inglés *Wide Area Network*, las *WANs* son redes de área amplia que abarca varias ubicaciones físicas, proveyendo servicio a una zona, un país, o incluso, a varios continentes.

❖ *Dirección IP*.

Todo dispositivo conectado a una red *TCP/IP* posee una dirección única dentro de la red, en la práctica a las direcciones *IP* se les denomina simplemente *IP*. El organismo encargado de asignar las direcciones *IP* es el *Internet Corporation for Assigned Names and Numbers* (*ICANN*). Actualmente se utilizan dos versiones de este direccionamiento, estas son:

1) Direccionamiento *IPv4*:

En esta versión del direccionamiento, cada IP está compuesta por un número binario de cuatro bytes y representado en notación punto, por ejemplo, 192.168.1.1. Cada sección, separada por el punto, representa un solo byte de la dirección de 32 bits. De esta manera, existen 2^{32}=4,294,967,296 direcciones posibles.

El direccionamiento *IPv4* está clasificado por el ICANN mediante la arquitectura de clases (*classful network architecture*). Esta arquitectura divide al IPv4 en 5 clases: A, B, C, D y E. Una *IP* pertenecerá a cierta clase siempre y cuando sus bits iniciales correspondan con los bits iniciales de esa clase. Las *IP* pertenecientes a las clases A, B y C son privadas y generalmente sirven para crear redes locales, mientras que las direcciones de la clase D son públicas y son únicas en el mundo, generalmente su uso implica una renta mensual.

Clase	Bits iniciales	Intervalo	Uso
A	0	0.0.0.0 a 127.255.255.255	Para redes muy grandes, por ejemplo: bancos, compañías internacionales, etc.
B	10	128.0.0.0 a 191.255.255.255	Para redes medianas, por ejemplo, la red de un campus universitario.
C	110	192.0.0.0 a 223.255.255.255	Se usa en redes para negocios pequeños a medianos y de uso residencial.
D	1110	224.0.0.0 a 239.255.255.255	La clase D contiene la totalidad de las direcciones públicas, estas direcciones nunca se repiten.
E	1111	240.0.0.0 a 255.255.255.255	El uso de estas direcciones es exclusivo de investigación y experimentación.

Tabla A.1: Detalle de las clases de IP.

2) Direccionamiento *IPv6* [12]*:*

Debido al creciente número de dispositivos capaces de conectarse a Internet, así como el desperdicio de direcciones *IP*, el direccionamiento *IPv4* ya no es suficiente, por lo que se está implementando esta nueva versión del direccionamiento. *IPv6* tiene las mismas funciones que IPv4 y aunque no son compatibles entre sí, se espera que en un futuro cercano *IPv6* conviva eficientemente con *IPv4*.

Cada *IPv6* está compuesta por un número binario de 128 bits y representada mediante ocho segmentos de dos bytes cada uno, separados por dos puntos y escrito en sistema hexadecimal. Esto quiere decir, que el direccionamiento *IPv6* puede tener hasta 2^{128}, esto es cerca de 6.7×10^{17} de direcciones por cada milímetro cuadrado de la superficie de la Tierra. En la siguiente figura se muestra un ejemplo de *IPv6*.

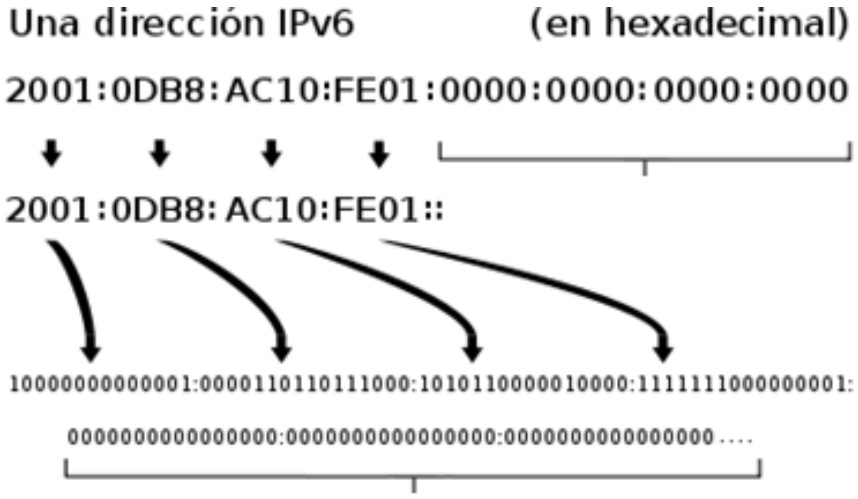


Figura A.15 Ejemplo de una dirección IPv6

[12] El direccionamiento *IPv5* no transcendió más allá del ámbito experimental.

❖ *Máscara de subred.*

El diseño de redes de clases (*classful network architecture*) sirvió durante la expansión de Internet, sin embargo, este diseño no es escalable y fue reemplazado por una arquitectura de redes sin clases basada en máscaras de subred, lo que permitió crear subredes de longitud variable. La máscara de subred es una combinación de 32 bits, separada en cuatro octetos, y escrita en notación decimal, que consiste en una secuencia de unos, seguidos de una secuencia de ceros, y son referenciadas por el número de unos, por ejemplo, una máscara de subred de *20 bits* es 255.255.240.0 (*véase figura A.2*).

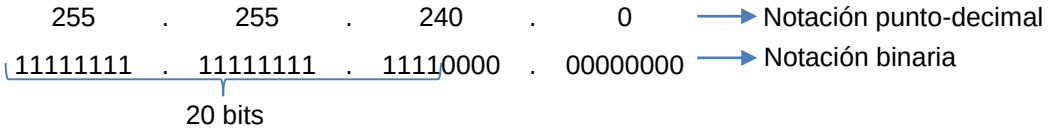


Figura A.2 Detalle de una máscara de subred de 20 bits.

La máscara de subred establece los parámetros de una subred, estos son, las direcciones IP asignables a los nodos que conforman la subred (*hosts*), el identificador de red, y la dirección de difusión, de los cuales, estos dos últimos no se pueden asignar a ningún *host*.

El identificador de red es la dirección IP inicial de una subred. Para conocer el identificador de red de una IP, se realiza la operación binaria *AND*, bit a bit, entre dicha dirección IP y la máscara de subred. Por ejemplo, un dispositivo cuya dirección IP sea 192.168.1.58, con la máscara de subred, 255.255.240.0, pertenece a la red 192.168.0.0 (*véase figura A.3*).

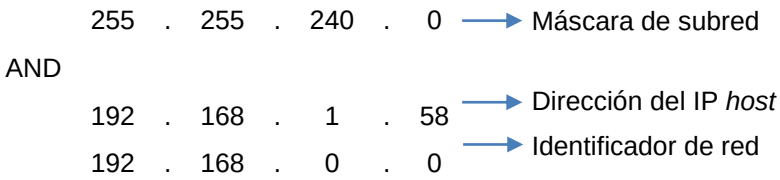


Figura A.3 Obtención del identificador de red de una dirección IP

El número de *unos* de la máscara de subred indica el número de bits iniciales que deben coincidir en todas las direcciones IP de la subred con el identificador de red de dicha subred, por ejemplo, todas las direcciones IP pertenecientes a la red con el identificador 192.168.0.0, y con máscara de subred 255.255.240.0, tienen la siguiente forma: 192.168.0000****.*********, donde la parte en rojo está en código binario y el asterisco puede ser cero o uno.

La última dirección IP de la subred corresponde a la dirección de difusión, cuando se envían datos a esta dirección, estos serán recibidos por todos los dispositivos que conforman la subred. Para la subred con identificador 192.168.0.0, cuya máscara es 255.255.240.0, la dirección de difusión es 192.168.15.255 (*véase figura A.4*).

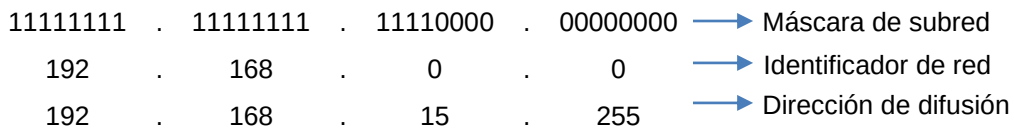


Figura A.4 Detalle de la dirección de difusión en una subred

Por lo tanto, el número de ceros de la máscara de subred corresponde a la cantidad de direcciones que se pueden asignar a los hosts que conforman la subred, excluyendo la primera y última dirección, por ejemplo, para la subred cuya máscara es 255.255.240.0, la cual tiene 12 ceros, se pueden asignar $(2^{12}) - 2 = 4094$ direcciones, es decir, la subred abarca 4094 dispositivos conectados.

Gracias a la máscara de subred podemos considerar a una subred *TCP/IP* como un conjunto de direcciones *IP*, o bien, un conjunto de dispositivos cuyo identificador de red y máscara de subred son los mismos.

- *Protocolo NAT (Network Address Translation).*

En general, las redes *LAN* se crean a partir de una subred compuesta de direcciones *IP* privadas, en la cual todos los dispositivos pueden comunicarse entre sí, si un dispositivo de esta subred desea conectarse a una red pública, enviará el tráfico a otro dispositivo de la misma red que traducirá la dirección *IP* de privada a pública, esto se logra mediante el protocolo *NAT*. En otras palabras, *NAT* es un protocolo que sirve para intercomunicar redes privadas.

En una misma subred no pueden existir dos direcciones iguales, pero sí se pueden repetir en dos redes privadas que no tengan conexión entre sí o que se conecten mediante el protocolo de traducción de direcciones de red (*NAT*). Cabe mencionar que el uso de direcciones *IP* publicas generalmente implica pagar un costo mensual.

- *Host.*

Un *Host,* o anfitrión, es un dispositivo conectado a una red *TCP/IP*, que provee y utiliza servicios de ella, un anfitrión puede ser un cliente o un servidor. Todo anfitrión posee una dirección *IP*.

- *Cliente.*

Un cliente es un proceso que solicita una conexión para comunicarse. Generalmente los clientes se comunican con un sólo servidor. Si algún proceso necesita "hablar" a diferentes servidores, éste crea múltiples clientes.

Haciendo una analogía con el servicio postal, los procesos serían las personas que desean enviar una carta, y la colocan dentro del buzón de correos, si la persona desea comunicarse con más de una persona, ésta escribirá múltiples cartas.

- *Servidores.*

Un servidor el proceso que responde a las solicitudes de conexión de los clientes. Normalmente los servidores manejan un gran número de solicitudes por esta razón suelen ser máquinas de cómputo especializadas, capaces de procesar grandes cantidades de información de manera muy eficaz.

Siguiendo con la analogía del servicio postal, el servidor sería la oficina postal, donde se reciben numerosas cartas y paquetes que deben ser procesados y encaminados hacia su destino final.

- *Puerto de red.*

Un puerto es un indicador lógico, denotado por un número de 16 bits entero, no signado, contenido en el rango entre 0 y 65535, que indica un proceso específico o un tipo de servicio que puede ofrecer una aplicación, asociado a una dirección IP. Cuando un cliente desea

solicitar un servicio de una aplicación, debe conocer, tanto la dirección IP del host donde se encuentra dicha aplicación, como el puerto en el cual está prestando el servicio.

Típicamente, las aplicaciones comunes tienen puertos fijos, de manera que no importando en qué dispositivo estén corriendo, el puerto está fijo para ese tipo de aplicación. Por ejemplo, la aplicación *HTTP* (páginas web) usa el puerto 80 y cuando lo único que se desea es descargar una página web, es suficiente dar la dirección IP donde ésta está almacenada, pues existe el mutuo acuerdo de que se usará el puerto 80. Los números de puerto menores a 1024 están reservados, y sólo deberían usarse cuando se está solicitando o implementando un protocolo bien conocido. Algunos ejemplos de estos son:

- 25: Correo electrónico (SMTP - Simple Mail Transfer Protocol).
- 69: Transferencia de archivos (TFTP - Trivial File Transfer Protocol)
- 110: Protocolo del servicio postal (POP3 - Post Office Protocol).

Recurriendo al ejemplo del correo postal, que además del envío de cartas, ofrece otros servicios como el envío de paquetes, giros postales, cobro de créditos, envío de dinero, entre otros, cuando el cliente solicita los servicios del correo postal, debe especificar cuál servicio desea utilizar. Por lo tanto, el cliente debe conocer la dirección donde se encuentra proporcionando servicios un correo postal en específico (es decir, la dirección IP, en el caso de un servidor), y el servicio que desea utilizar (es decir, el puerto en el que se encuentra "escuchando" el servidor para proporcionar el servicio).

❖ *Dispositivos de red.*

Existen diversas tecnologías para conectarse a una red *TCP/IP*, la más utilizada es lo que comúnmente conocemos como Ethernet, esto es, un cable que une a dos nodos a través de conectores, clavijas o transceptores. De esta manera se crea una red punto a punto. Para hacer redes más grandes se utilizan dispositivos de red, a continuación, se mencionan algunos de ellos:

- *Switch* o conmutador: Este dispositivo permite crear una red LAN, la topología de conexión es de tipo estrella y se comporta lógicamente como un multiplexor, redirigiendo el tráfico de un punto a otro en específico. La dirección IP que puedan llegar a tener asignada, se utiliza únicamente para configuración remota.

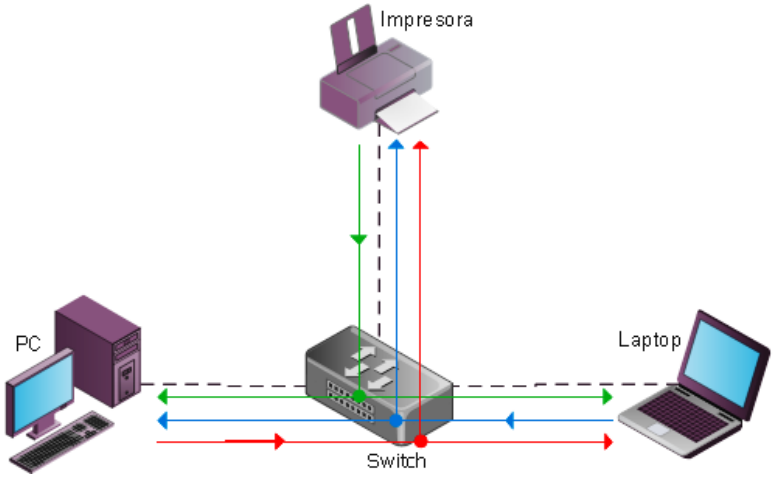


Figura A.5 Representación de un dispositivo switch

- *Access Point* o punto de acceso: Estos dispositivos se comportan lógicamente igual que un switch, pero las conexiones son de manera inalámbrica. La mayoría de los *AP* usan los estándares IEEE 802.11. Para darse a conocer a otros dispositivos, los *AP* incluyen en los paquetes de información que trasmiten su nombre identificador o *SSID* (*Service Set Identifier*) y son capaces de solicitar credenciales de identificación a los dispositivos con los que desean establecer una conexión. Al igual que el *switch*, la dirección *IP* de un *Access Point*, se usa exclusivamente para configurarlo.

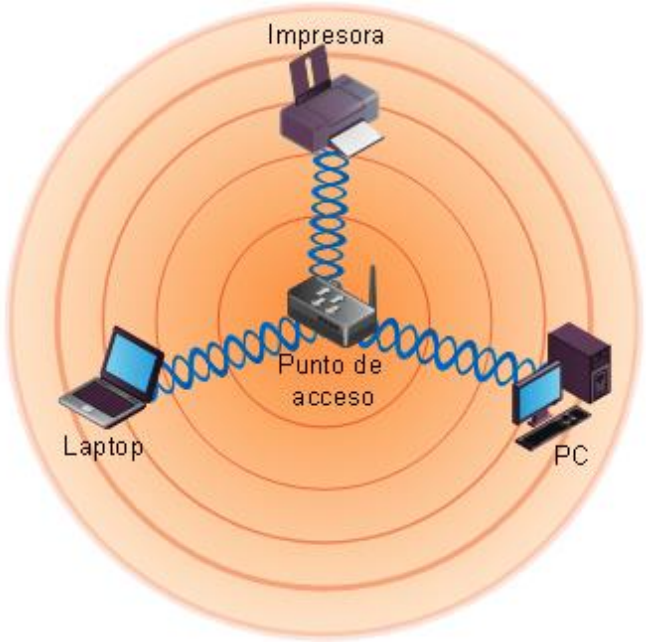


Figura A.6 Representación de punto de acceso

- *Router* o enrutador: este dispositivo permite llevar datos desde una *LAN* a otra, basándose en las direcciones *IP*, en otras palabras, el router permite interconectar redes locales. Cabe destacar que cada puerto del Router tiene una dirección *IP* asignada, correspondiente a la subred que tiene conectada en dicho puerto, esto con la finalidad de lograr la interconexión de las distintas *LANs*.

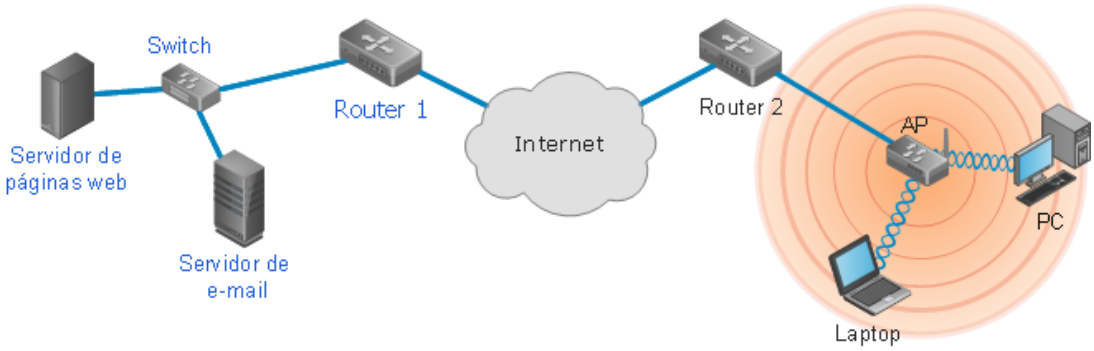


Figura A.7 Representación del router

❖ *Internet.*

Internet es un conjunto de redes de dispositivos interconectados entre sí, según la familia de protocolos *TCP/IP*. En otras palabras, Internet es una red de redes de dispositivos físicos, cuyos protocolos que la rigen hacen que Internet se comporte como una red lógica única de alcance mundial.

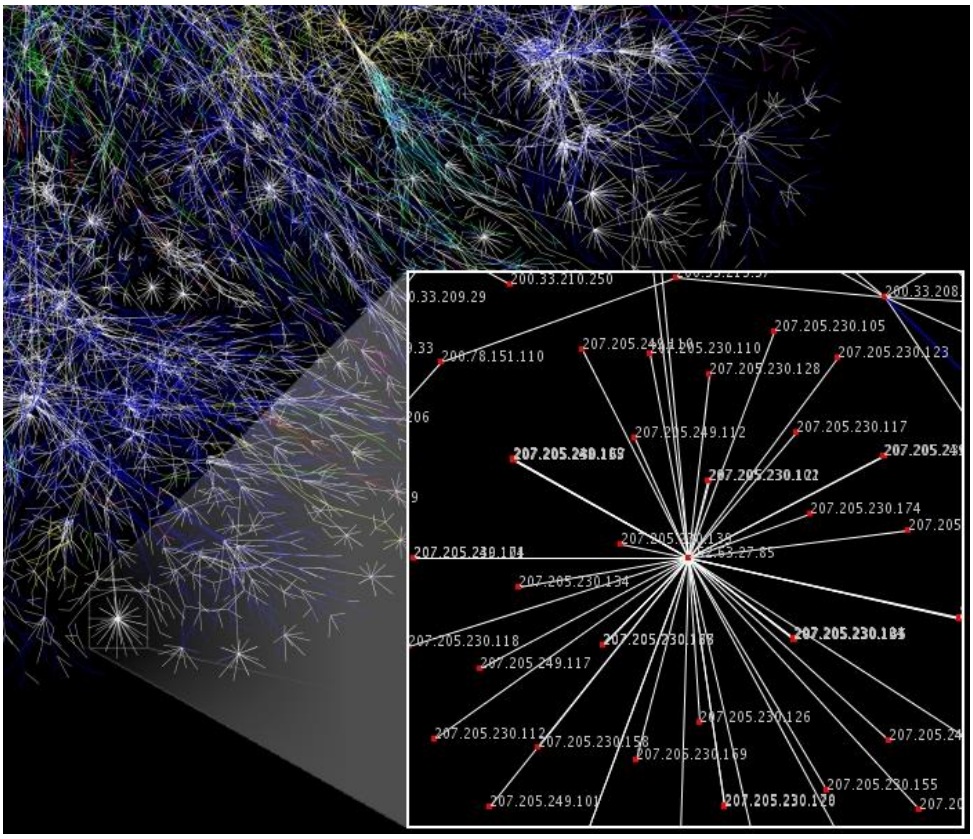


Figura A.8 Representación parcial de Internet, imagen obtenida del sitio opte.org. Cada nodo representa un dispositivo asociado a una dirección IP. La distancia entre dos nodos, es proporcional al tiempo que tardan en comunicarse entre sí.

❖ *ISP (Internet Service Provider).*

Del inglés, proveedor de servicios de Internet, es la empresa encargada de brindar conexión a Internet a sus clientes, por ejemplo, Telmex, Maxcom, Cablecom, Axtel, entre otras. Estas empresas tienen consignada una porción del direccionamiento *IP* público y lo reparten entre las personas físicas o morales que se lo soliciten a cambio de una renta mensual.

APÉNDICE B. CÓDIGO FUENTE DE LOS EJEMPLOS DEL CLIENTE Y EL SERVIDOR TCP.

En el presente apéndice se muestra código fuente de los ejemplos creados para demostra el funcionamiento del componente *TIdTCPSever y TIdTCPClient* del proyecto Indy, tal y como fueron descritos en los subtemas 2.1 y 2.4 de este documento. El lenguaje de este código es el Onject Pascal y el ambiente de desarrollo es el RAD Studio XE8.

✓ Código fuente del programa *IdTPCServerExample*

```
unit IdTPCServerExampleUnit1;

interface

// Declaración de bibliotecas.
uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, Vcl.Forms,
  System.Classes, Vcl.Graphics, Vcl.Controls,  Vcl.Dialogs, Vcl.StdCtrls,
  IdBaseComponent, IdComponent, IdCustomTCPServer, IdTCPServer, IdContext;

// Instanciación del objeto Form1.
type
  TForm1 = class(TForm)
    IdTCPServer1: TIdTCPServer;
    BotIniciar: TButton;
    Memo1: TMemo;
    procedure BotIniciarClick(Sender: TObject);
    procedure IdTCPServer1Execute(AContext: TIdContext);
    procedure IdTCPServer1Connect(AContext: TIdContext);
    procedure IdTCPServer1Disconnect(AContext: TIdContext);
end;

//Declaración de variable Globales.
var
  Form1: TForm1;

//Implementación
implementation
{$R *.dfm}

// Método OnClick del botón "Iniciar"
 procedure TForm1.BotIniciarClick(Sender: TObject);
 begin
   IdTCPServer1.Bindings.Add.IP := '192.168.1.2'; //Asigna IP
   IdTCPServer1.Bindings.Add.Port := 8080;        //Asigna puerto
   IdTCPServer1.Active := true;                   //Inicia servicio TCP
   BotIniciar.Caption := 'Iniciado...';           //
   BotIniciar.Enabled := False;                   //Dehabilita botón "iniciar"
 end;

 // Método OnConnect
 procedure TForm1.IdTCPServer1Connect(AContext: TIdContext);
 var
   Anfitrión : String;
 begin
   Anfitrión  := AContext.Connection.Socket.Binding.PeerIP;
   Memo1.lines.Add('El cliente ' + Anfitrión  + ' se ha conectado'); //Mensaje testigo
   end;                                                              // de la conexión
```

```
// Método OnExceute
procedure TForm1.IdTCPServer1Execute(AContext: TIdContext);
Var
  CadenaEntrante: String;
begin
  try
    begin
      CadenaEntrante := Trim(AContext.Connection.iohandler.ReadLn);
      if CadenaEntrante <> '' then
        Memo1.lines.Add('El cliente dice: ' + CadenaEntrante)
      else AContext.Connection.Disconnect;     //Muestra mensaje en caja de memoria.
    end;
  except
    on e: Exception do
    begin
      AContext.Connection.iohandler.Writeln('Error=' + e.message);
    end;
  end;
end;

// Método OnDisconnect
procedure TForm1.IdTCPServer1Disconnect(AContext: TIdContext);
var
  Anfitrión : String;
begin
  Anfitrión  := AContext.Connection.Socket.Binding.PeerIP;
  Memo1.lines.Add('El cliente ' + Anfitrión  + ' se ha desconectado'); //Mensaje testigo
end;                                                                   //de la desconexión

end.
```

✓ Forma del programa *IdTPCServerExample*

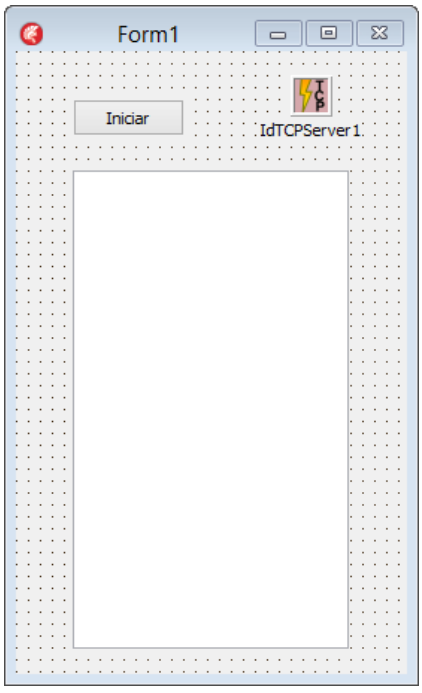


Figura B.1 Forma del ejemplo del servidor TCP

✓ Código fuente del programa *IdTPCClientExample*

```
unit IdTCPClientExampleUnit;

interface

uses
  System.SysUtils, System.Types, System.UITypes, System.Classes,IdBaseComponent,
  IdComponent, IdTCPConnection, IdTCPClient, System.Variants, FMX.Graphics,
  FMX.Controls, FMX.Types, FMX.Forms, FMX.Dialogs, FMX.StdCtrls, FMX.Edit,
  FMX.Controls.Presentation, FMX.ScrollBox, FMX.Memo;

type
// Instanciación del objeto Form1.
  TForm1 = class(TForm)
    IdTCPClient1: TIdTCPClient;
    Memo: TMemo;
    BotConectar: TButton;
    MensajeEdit: TEdit;
    BotEnviar: TButton;
    NickNameEdit: TEdit;
    Etiqueta1: TLabel;
    procedure BotConectarClick(Sender: TObject);
    procedure IdTCPClient1Connected(Sender: TObject);
    procedure IdTCPClient1Disconnected(Sender: TObject);
    procedure BotEnviarClick(Sender: TObject);
  end;

//Instanciación de un hijo de ejecución.
  TClientThread = class(TThread) // TClientThread hereda atributos de TThread.
  protected
    procedure Execute; override; // Definición del método Execute
  public
    constructor Create(CreateSupended: Boolean); // Definición del contructor
  end;

var
  Form1: TForm1;

//Implementación
Implementation

{$R *.fmx}

//Método on OnClick para conectarse al servidor.
procedure TForm1.BotConectarClick(Sender: TObject);
begin
  IdTCPClient1.Host := '192.168.1.2';       //Dirección IP de Zero
  IdTCPClient1.Port := 8080;                //Puerto
  IdTCPClient1.Connect;                     //Invoca método Connect
  BotConectar.Enabled := False;             //Deshabilita botón BotConectar
end;

//Método on OnClick para enviar mensaje al servidor.
procedure TForm1.BotEnviarClick(Sender: TObject);
begin
   IdTCPClient1.IOHandler.Writeln(MensajeEdit.Text);
end;

//Método on OnConnected
procedure TForm1.IdTCPClient1Connected(Sender: TObject);
begin
  IdTCPClient1.IOHandler.Writeln
    ('nick=' + NickNameEdit.Text);  // Envia identificador al servidor
  TClientThread.Create(False);      // Invoca constructor del hilo de ejecución
end;
```

```
//Método on OnDisconnected
procedure TForm1.IdTCPClient1Disconnected(Sender: TObject);
begin
  ShowMessage('desconectado');
  BotConectar.Enabled := True;
end;

//Método contructor del hilo
constructor TClientThread.Create(CreateSupended: Boolean);
begin
  inherited Create(CreateSupended);
  TClientThread.Priority := tpNormal;
  FreeOnterminate := True;
end;

//Método ascociado a la ejecución del hilo de procesamiento paralelo
procedure TClientThread.Execute;
var
  MensEnt: String;
begin
  with Form1 do
  begin
    if not IdTCPClient1.Connected then
      exit;
    repeat
      MensEnt := IdTCPClient1.IOHandler.ReadLn;   //Recibe mensaje
      if trim(MensEnt) <> '' then                 //Si mensaje no es vacío
        Memo.Lines.Add(MensEnt);                  //lo agrega al memo.
    until not IdTCPClient1.Connected;
  end;
end;

end.
```

✓ Forma del programa *IdTPCClientExample*

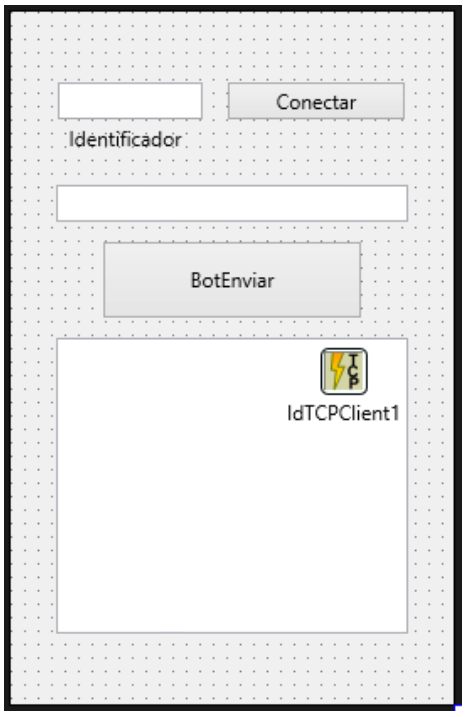


Figura B.2 Forma del ejemplo del cliente TCP

APÉNDICE C. Código fuente de la biblioteca de subrutinas *Bicoesp8266.*

Archivo "*bicoesp8266_con_sh32.b*"

```
dim bufrecep(29) as byte      'Se define buffer de recepción de 30
                              'localidades con este arreglo.




dim apunbufrecep as integer   'Se define variable que contiene
                              'siguiente dirección disponible para
                              'almacenamiento del siguiente byte
                              'recibido.
    iniens
scibdh equ $38
scibdl equ $39
scic1  equ $3a
scic2  equ $3b
scis1  equ $3c
scis2  equ $3d
scic3  equ $3e
scid   equ $3f

sopt1 equ $1802

      lda #$20
      sta sopt1               'Deshabilita COP
      cli                     'Limpia bandera de interrupción global
    finens
```

Archivo "*bicoesp8266_rut.b"*

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina CusPrint
' Antes de invocar se debe almacenar en la varible OutCad$ la cadena a
' transmitir por el puerto serial del microcontrolador.
' Al retornar ya se habrán transmitido todos los caracteres de OutCad$
' a través del puerto serial.

CusPrint:    for icp~=1 to len(OutCad$)+0
                car$=mid$(OutCad$,icp~,1)
                Outcar~=asc(car$)

                iniens
                  lda Outcar~
                  jsr pon#car
                finens

                next icp~
                return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina EnviaRes
```

```
' Antes de invocar se deben actualizar las variables Comandante$ y
Resp$.
' Al retornar ya se habrá transmitido por el puerto SCI una cadena con
' el formato "msg=nick#Comandante#Resp".
EnviaRes:
            OutCad$="msg="
            gosub CusPrint
            OutCad$=nick$
            gosub CusPrint
            gosub PoGato
            OutCad$=Comandante$
            gosub CusPrint
            gosub PoGato
            OutCad$=Resp$
            gosub CusPrint
            gosub PoCRLF
            gosub ret50ms
            return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina CusInput
' Recibe por interrupción string comando que termina con los caracteres
' <enter> + <line feed> (chr(13)+chr(10)).
' Al invocarse se pasa a una espera por el llenado por interrupción del
' buffer de recepción que contendrá el comando recibido.
' Al retornar:
' Cadena$ <- String que contiene el comando recibido.

CusInput:
            contcar~=0              'Se define 'variable contadora de
                                    'bytes recibidos por interrupción.

            iniens
              ldhx #bufrecep
              sthx apunbufrecep     'Inicializa apuntador de buffer de
                                    'recepción que contiene siguiente
                                    'dirección disponible para
                                    'almacenamiento del siguiente byte
                                    'recibido.

              bset 5,scic2          'RIE <-- 1, habilita interrupciones
                                    'del SCI por recepción.

espfincmd:    brset 5,scic2,espfincmd    'Espera por fin de recepción
                                         'de comando.

                    ldhx #bufrecep
                    sthx apunbufrecep    'Reinicializa apuntador de
                                         'buffer derecepción.
            finens

            bytedebuf%=0
            Cadena$=""
            for i~=1 to contcar~ -1
              iniens
                ldhx apunbufrecep
                lda ,x
```

```
                sta bytedebuf%+1
                aix #$01
                sthx apunbufrecep
              finens

              Cadena$ = Cadena$ + chr$(bytedebuf%)
            next i~
            return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina PoCRLF
' Envía por el puerto serial el código ASCII del caracter de control
' "retorno de carro", seguido del caracter de control "salto de línea".

PoCRLF:           miLF~=10
                  miCR~=13

                  iniens
                    lda miCR~
                    jsr pon#car
                    lda miLF~
                    jsr pon#car
                  finens

                  gosub ret5ms
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina PoComa
' Envía por el puerto serial el código ASCII del caracter "coma" (,).

PoComa:           MiComa~=44

                  iniens
                    lda MiComa~
                    jsr pon#car
                  finens

                  gosub ret5ms
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina PoComi
' Envía por el puerto serial el código ASCII del caracter "comillas"
(").

PoComi:           MisComillas~=34

                  iniens
                    lda MisComillas~
                    jsr pon#car
                  finens

                  gosub ret5ms
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina PoGato
' Envía por el puerto serial el código ASCII del caracter "gato" (#).

PoGato:           MiGato~=35
                  iniens
                    lda MiGato~
                    jsr pon#car
                  finens
                  gosub ret5ms
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina TrimCadena
' Antes de invocar, la cadena a recortar deberá estar almacenada en la
' variable Cadena$.
' Al retornar ya se habrán eliminado caracteres especiales, tales como
' los espacios en blanco, saltos de línea y retorno de carro al inicio
' y final del contenido cadena. No se modifica nada si Cadena$=""

TrimCadena:       PosIn~=0
                  if Cadena$="" then
                    goto StopTrim
                  endif
                  longcad~=len(Cadena$)
                  for it~=1 to longcad~
                    car$=mid$(Cadena$,it~,1)
                    car~=asc(car$)
                    if car~=&h0a then
                      goto OtroMas
                    elseif car~=&h0d then
                      goto OtroMas
                    elseif car~=&h20 then
                      goto OtroMas
                    else
                      PosIn~=it~
                      exit for
                    endif
OtroMas:          next  it~
                  if PosIn~=0 then
                    goto StopTrim
                  endif
                  for it~=longcad~ to 1 step -1
                    car$=mid$(Cadena$,it~,1)
                    car~=asc(car$)
                    if car~=&h0a then
                      goto OtroMen
                    elseif car~=&h0d then
                      goto OtroMen
                    elseif car~=&h20 then
                      goto OtroMen
                    else
                      PosEnd~=it~
                      exit for
                    endif
                  print PosEnd~
OtroMen:          next  it~
```

```
                  Cadena$=mid$(Cadena$,PosIn~,PosEnd~-PosIn~+1)
StopTrim:         return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina ValCadena
' Antes de invocar la cadena a verificar deberá estar almacenada en la
' variable Cadena$.
' Al retornar, la variable InvalidC~ será falsa, si Cadena$ contiene un
' comando válido procesable por el programa,de lo contrario InvalidC~
' será verdadera.

ValCadena:        InvalidC~=0
                  CadenaLong~=len(Cadena$)
                  if CadenaLong~<9 then
                    InvalidC~=1
                    goto basta
                  endif
                  PruError$=mid$(Cadena$,1,3)
                  if PruError$<>"msg" then
                    InvalidC~=1
                    goto basta
                  endif
                  CantG~=0
                  for ig~=1 to CadenaLong~
                    car$=mid$(Cadena$,ig~,1)
                    if car$="#" then
                      CantG~=CantG~+1
                    endif
                  next ig~
                  if CantG~<>2 then
                    InvalidC~=1
                    goto basta
                  endif
basta:            return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina ProCadena
' Antes de invocar, la cadena a ser procesada deberá almacenarse en la
' variable Cadena$.
' Al retornar ya se habrán almacenado en las variables Comando$ y
' Comandante$, los nombres correspondientes al comando a ejecutar y al
' dispositivo que solicitó la orden, respectivamente.

ProCadena:        CadenaLong~=len(Cadena$)
                  CadTemp$=mid$(Cadena$,5,CadenaLong~-4)
                  CadTempLong~=len(CadTemp$)
                  for ig~=1 to CadTempLong~
                    car$=mid$(CadTemp$,ig~,1)
                    if car$="#" then
                      PG~=ig~
                      exit for
                    endif
                  next ig~
                  Comandante$=mid$(CadTemp$,1,PG~-1)
                  CadTemp$=mid$(CadTemp$,PG~+1,CadTempLong~-PG~)
                  CadTempLong~=len(CadTemp$)
```

```
                  for ig~=1 to CadTempLong~
                    car$=mid$(CadTemp$,ig~,1)
                    if car$="#" then
                      PG~=ig~
                      exit for
                    endif
                  next ig~
                  Comando$=mid$(CadTemp$,PG~+1,CadTempLong~-PG~)
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina ConWFM
' Envía por el puerto serial los comandos AT que un módulo Wifi
' basado en el chip ES8266, necesita para conectarse a un punto de
acceso.
' Antes de invocar el SSID y contraseña del punto de acceso deberán
estar
' almacenadas en las variables SSID$ y Pass$ respectivamente.
' Antes de retornar, se espera departe del módulo Wifi, el caracter de
' confirmación "K".
' Al retornar ya se habrán enviado los comandos AT
' necesarios habrá recibido el caracter de confirmación.

ConWFM:           gosub ret500ms
                  OutCad$="ATE0"          'Deshabilita el eco.
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  OutCad$="AT+CWAUTOCONN=0"
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  OutCad$="AT+CWMODE_DEF=1"
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  OutCad$="AT+RFPOWER=40"
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  OutCad$="AT+CWQAP"
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  OutCad$="AT+CWJAP_CUR="
                  gosub CusPrint
                  gosub PoComi
                  OutCad$=SSID$
                  gosub CusPrint
                  gosub PoComi
                  gosub PoComa
                  gosub PoComi
                  OutCad$=Pass$
                  gosub CusPrint
                  gosub PoComi
                  gosub PoCRLF
```

```
                  iniens
EspK:               jsr lee#car
                    cmp #$4b
                    bne EspK
                  finens

                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Subrutina IniTCP
' Envía por el puerto serial los comandos AT que un módulo Wifi basado
' en el chip ES8266, necesita para conectarse a un servidor TCP, e
iniciar
' una transmisión transparente TCP a SCI.
' Antes de invocar la dirección IP y su puerto correspondiente deberán
' estar almacenadas en las variables DirIP$ y Puerto$ respectivamente.
' El nombre de anfitrión deberá estar almacenado en la variable nick$
' Al retornar ya se habrán enviado los comandos AT pertinentes.

IniTCP:           OutCad$="AT+CIPSTART="
                  gosub CusPrint
                  gosub PoComi
                  OutCad$="TCP"
                  gosub CusPrint
                  gosub PoComi
                  gosub PoComa
                  gosub PoComi
                  OutCad$=DirIP$
                  gosub CusPrint
                  gosub PoComi
                  OutCad$=","+Puerto$
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  OutCad$="AT+CIPMODE=1"
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  OutCad$="AT+CIPSEND"
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  OutCad$="nick="+nick$
                  gosub CusPrint
                  gosub PoCRLF
                  gosub ret50ms
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Las subrutinas ret500us, ret5ms, ret50ms y ret500ms son subrutinas de
' retardo, que mantienen ocupado al microcontrolador el tiempo que
indica
' su etiqueta.

ret500us:
                  iniens
                    pshh
```

```
                    pshx
                    ldhx #$03e5
vuelta:             nop
                    nop
                    aix #$ff
                    cphx #$0000
                    bne vuelta
                    pulx
                    pulh
                  finens

                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

ret5ms:           for i% =1 to 10   'Inicio de ciclo for
                    gosub ret500us  'Llamado a la subrutina ret500us
                  next i%           'Fin de ciclo for
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

ret50ms:          for j% =1 to 100  'Inicio de ciclo for
                    gosub ret5ms    'Llamado a la subrutina ret5ms
                  next j%           'Fin de ciclo for
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''

ret500ms:         for k% =1 to 10   'Inicio de ciclo for
                    gosub ret50ms   'Llamado a la subrutina ret50ms
                  next k%           'Fin de ciclo for
                  return
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Rutina de servicio de interrupción al recibirse un byte por el
' puerto serie del MCU.
' El byte recibido se carga en un buffer que está en RAM a partir de la
' dirección inicial del arreglo.

servrecepsci:
                  iniens
                    pshh
                    ldhx apunbufrecep
                    lda scis1       'Lectura fantasma de scis1 para que
                                    'que se regrese a cero la bandera
testigo
                                    'RDRF al leerse el registro scid.
                    lda scid        'a <-- byte recibido, rdrf <-- 0
                    cmp #$0a        'Checa si el byte recibido es $0a
(Line feed),
                    bne siguet      'si es el caso deshabilita
                                     'la interrupción y sale.

                    bclr 5,scic2    'Deshabilita interrupciones por
                                     'recepción del SCI.
                    bra salservrecep

siguet:             sta ,x          'Almacena byte recibido en lugar
```

```
                                 'disponible en buffer.
                    aix #$01
                    sthx apunbufrecep     'Actualiza apuntador del
                                           'siguiente lugar disponible
                                           'en buffer.

'Incrementa contador de caracteres recibidos.
                    lda contcar~
                    inca
                    sta contcar~
''''''''''''''''''''''''''''''''''''''''''''''

salservrecep:       pulh
                  finens
                  retint
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

REFERENCIAS

[1] C. Z. H. a. K. y. H. Hariri, «Indy in Depth», 2002 - 2005, cap. 8: *Plain Text*, p. 125.

[2] Salvá, Antonio y Altamirano, Luis, «Dispositivos Chipbas8, microcontroladores HC08 programables en lenguaje BASIC», Memoria del Simposio Anual de Automatización, Electrónica e Instrumentación, Julio 2009, Madrid, España. [En línea]. Liga: http://dctrl.fi-b.unam.mx/~salva.

[3] A. Salvá Calleja, «Ambiente Integrado para Desarrollo y Aprendizaje con Microcontroladores de la Familia 68hc908 de Freescale», Archivo muaida08ve2011.pdf, [En línea]. Liga: http://dctrl.fi-b.unam.mx/~salva.

[4] Embarcadero Technologies, «RAD Studio XE8», Mayo 2015. [En línea]. Liga: http://www.embarcadero.com/es/products/rad-studio.

[5] Atozed Software, «Overview of Indy Sockets» [En línea]. Liga: http://www.indyproject.org/sockets/index.en.aspx . [Último acceso: Mayo 2015].

[6] C. Z. Hower y H. Hariri, «Indy in Depth», Atozed Software, July 25, 2005.

[7] Google, «Data Centers,» Junio 2015. [En línea]. Liga: https://goo.gl/cpYHGl.

[8] C. Z. Hower y H. Hadi, «Indy in Depth», 2002 - 2005, cap. 16: *Hundreds of threads*, p. 188.

[9] Embarcadero Technologies, «The TThread Class» [En línea]. Liga: http://docwiki.embarcadero.com/Libraries/Seattle/en/System.Classes.TThread. [Último acceso: Class TThread julio 2015].

[10] Embarcadero Technologies, «Initializing a Thread» [En línea]. Liga: http://docwiki.embarcadero.com/RADStudio/Seattle/en/Initializing_a_Thread. [Último acceso: Julio 2015].

[11] B. Benchoff, «Brian Benchoff», Hackaday, 26 agosto 2014. [En línea]. Liga: http://hackaday.com/2014/08/26/new-chip-alert-the-esp8266-wifi-module-its-5/. [Último acceso: Enero 2016].

[12] A. C. (tuxedo0801), « ESP8266 Module Family» [En línea]. Liga: http://www.esp8266.com/wiki/doku.php?id=esp8266-module-family [Último acceso: marzo 2016].

[13] CNLohr, «ESP8266 Wifi Range/Distance Tests» [En línea]. Liga: https://www.youtube.com/watch?v=7BYdZ_24yg0. [Último acceso: agosto 2015].

[14] Espressif Systems, «ESP8266 AT Instruction Set» [En línea]. Liga: http://goo.gl/KyqqPE . [Último acceso: Agosto 2015].

[15] Espressif , «ESP826 AT Release V0.60» [En línea]. Liga: http://bbs.espressif.com/viewtopic.php?f=46&p=5570#p5570. [Último acceso: Junio 2015].

[16] Espressif, «Flash Download Tool» [En línea]. Liga: http://bbs.espressif.com/viewtopic.php?f=5&t=433. [Último acceso: Agosto 2015].

[17] CUI Inc., «PBK-5 Series» [En línea]. Liga: http://www.cui.com/product/resource/pbk-5.pdf . [Último acceso: Octubre 2015].

[18] Freescale Semiconductor, «MC9S08SH32/16 Data Sheet» [En línea]. Liga: http://goo.gl/6CJzWr . [Último acceso: agosto 2015].

[19] Suntac Electronic Corp., «BT134 Series,» [En línea]. Liga: http://pdf1.alldatasheet.es/datasheet-pdf/view/257748/SUNTAC/BT134.html. [Último acceso: Junio 2015].

[20] D. Dextrel, «Isolated High Quality Mains Voltage Zero Crossing Detector» [En línea]. Liga: http://www.dextrel.net/diyzerocrosser.htm. [Último acceso: Mayo 2015].

[21] Maxim Integrated, «DS1307 Real-Time Clock» [En línea]. Liga: http://datasheets.maximintegrated.com/en/ds/DS1307.pdf. [Último acceso: Julio 2015].

[22] D. H. Summerville , «Embedded Systems Interfacing for Engineers using the Freescale HCS08 Microcontroller I: Assembly Language Programming», 2009.