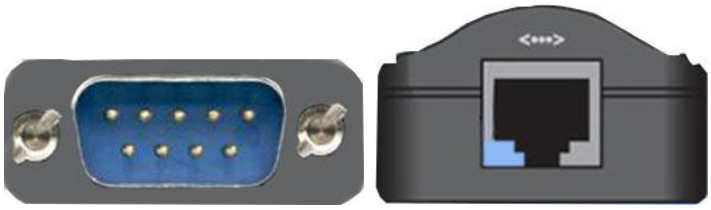
Universidad de las Ciencias Informáticas

Facultad # 5

Título: Conversor de protocolo RS-232 a Ethernet con microcontrolador STM32F107VCT.

Trabajo de Diploma para optar por el título de Ingeniero de las Ciencias Informáticas.



Autor(es): Aylín Rodríguez Reyo.

Tutor(es):

Ing. Alexander Moreno Limonte.

Ing. Leonardo Rafael Fernández Ruiz.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____

Aylín Rodríguez Reyo

Autora

_____

Ing. Alexander Moreno Limonte

Tutor

_____

Ing. Leonardo Rafael Fernández R

Tutor

Datos de contactos

Tutor: Ing. Alexander Moreno Limonte.

Síntesis del Tutor:

- Especialidad de graduación: Ingeniero en Control Automático.
- Categoría docente: Instructor.
- Categoría científica: -
- Años de experiencia en el tema: 6 años.
- Años de graduado: 6 años.
- E-mail: amlimonte@uci.cu

Tutor: Ing. Leonardo Rafael Fernández Ruiz.

Síntesis del Tutor:

- Especialidad de graduación: Ingeniero en Ciencias Informáticas.
- Categoría docente: Instructor.
- Categoría científica: -
- Años de experiencia en el tema: 4 años.
- Años de graduado: 4 años.
- E-mail: lrfernandez@uci.cu

Dedicatoria

A mis padres.

Por enseñarme que las cosas buenas de la vida siempre requieren de grandes esfuerzos, por confiar en mí en cada momento y por su sacrificio para poder estar hoy donde estoy. Por el amor incomparable que siempre me han dedicado.

A mi hermano.

Por estar siempre en mi corazón. Para demostrarte que cuando se quiere se puede y que siempre hay tiempo para cada cosa. Espero que *esto te sirva de ejemplo para que algún día tú te gradúes como universitario y entonces sea mi nombre el*
que aparezca en la dedicatoria.

A mis tías:

Mirna, Marlene, Rosa María, Marisol.

Por darme siempre mucho cariño y ser las tías más maravillosas que una persona pueda tener.

A mi tío José.

Por su esfuerzo, amor y apoyo incondicional.

Agradecimientos

A mis padres y mi hermano.

A las personas que siempre me ha brindado su apoyo incondicional en todo momento sabiendo siempre como ser los mejores padres del mundo. Por saber guiarme siempre por el buen camino hasta ser la clave de mis resultados y porque cada consejo suyo en mi niñez fue una herramienta para la vida. Por estar presente cada vez que los necesito.

A mi familia

A las personas que han estado conmigo dándome su confianza, amor y apoyo constante.

A mi tutor.

A la persona que ha sabido guiarnos por el camino de la ciencia como todo un profesional. Siempre muy atento y cuya fuerza de voluntad y entusiasmo son casi incomparables. Por ser también un buen amigo.

A mis amigos.

A ese grupo entusiasta de amigos por pasar tantos momentos buenos y malos juntos, por contribuir a que la mejor etapa de mi vida sea más agradable y divertida. En fin, por ayudar a que todo sea más fácil.

A Frank.

Por haberme ayudado incondicionalmente en todos estos años de la carrera de una forma u otra, por brindarme su apoyo y amor en cada momento.

Todos.

A todos los que de una forma u otra contribuyeron a mí formación y a este logro.

Resumen

La presente investigación recoge el diseño e implementación del *firmware* para un conversor de protocolos de RS-232 a *ethernet*. La solución propuesta se implementa sobre un microcontrolador STM32F107VCT, del fabricante *STmicroelectronic*. La arquitectura está sustentada sobre el sistema operativo FreeRTOS y se hace uso de la API[1] que brinda la biblioteca LwIP[2] para el trabajo con la interfaz *ethernet*. Para el manejo del núcleo y de los periféricos se utiliza la biblioteca estándar STM32F10x, así como los manejadores de los periféricos utilizados (USART[3], *ethernet*). Por último se exponen los resultados de las pruebas de validación realizadas en cada iteración con la herramienta "Test Soft", desarrollada por el equipo de pruebas del Centro de Informática Industrial (CEDIN), que permiten comprobar el desempeño de los prototipos funcionales, así como el funcionamiento del producto en su totalidad.

Palabras claves: CEDIN, conversor de protocolo, ethernet, firmware, FreeRTOS, LwIP, sistema embebido, USART.

[1] API: Interfaz de Programación de Aplicaciones (*Application Programming Interface*).
[2] LwIP: Biblioteca ligera TCP/IP (*Light Weight TCP/IP*).
[3] USART: Transmisor Receptor Universal Sincrónico y Asincrónico (*Universal Synchronous / Asynchronous Receiver Transmitter*).

Índice

Introducción

El mundo moderno está regido cada vez más por automatismos que controlan la mayoría de los sistemas con los que interactuamos a diario, desde una lavadora hasta una fábrica, cuentan con elementos de cómputo que realizan tareas y supervisan la realización de determinados procedimientos. Esto es posible gracias a una compleja red de sensores, actuadores y elementos de control programables, conocidos como PLC[4], que entrelazados, a través de un sistema de comunicación conocido comúnmente como bus de campo, permiten interactuar con la planta para lograr cumplir con el objeto de control, que puede ser desde que la ropa salga limpia, hasta que la planta logre mantener un nivel de producción estable con una calidad determinada (Verhappen y Byres, 2006).

Con la introducción de la computadora como elemento de control y el rápido desarrollo de la informática han aparecido otros niveles en la pirámide de control, que permiten no solo controlar los procesos, sino también la supervisión de los mismos por sistemas SCADA[5] (Research, 2000). Estos sistemas son capaces de comunicarse con los PLC, adquirir la información de los sensores y representar a través de mímicos, el estado general o particular de una planta o proceso, permitiendo al usuario la interacción y la toma de decisiones.

La interacción de los SCADA con la planta se realiza a través de los sistemas de comunicación, haciendo uso de los buses de campo y de los distintos protocolos industriales que existen. Estos buses de campo pueden tener una capa física que varía desde interfaces seriales hasta inalámbricas haciendo uso de las normas industriales para estos fines. Desde el punto de vista de los ordenadores donde radican los sistemas SCADA, una de las más usadas hoy en día son las redes *ethernet,* debido a las grandes facilidades que brindan, además de la gran gama de protocolos que permiten manejar y sobre todo su naturaleza de carácter distribuida (Stallings, 2000). Esto ha provocado, que los fabricantes hayan dejado de incluir otro tipo de interfaces en la arquitectura de los ordenadores modernos como por ejemplo el serial RS-232. En el mundo de los automatismos, debido a la naturaleza de las señales que se transmiten y lo hostil del medio en que se realizan estas comunicaciones, se mantienen este tipo de

[4] PLC: Controlador Lógico Programable (*Programing Logic Controllers*).
[5] SCADA: Sistema de Supervisión, Control y Adquisición de Datos (*Supervisory Control And Data Acquisition*).

interfaces seriales con variantes industriales que permiten una mayor eficiencia como son las normas eléctricas RS-485, RS-422.

En casos como estos donde las capas de transporte entre el sistema supervisorio y el controlador son de carácter distinto, para poder monitorear toda la información proveniente de este último, se necesita convertir de un medio de transmisión a otro. Estos entes hoy en día son conocidos como los conversores de protocolos y son muy usados en las soluciones de automatización industrial.

Dentro de los fabricantes más destacados de este tipo de dispositivos encontramos Moxa con sus series de productos CN2610/2650, NPort_6450, BRAINBOXES con la serie ES-246 y ES-357, además de diversos fabricantes chinos con sede en Shanghai, Shenzhen y Jinan con disímiles dispositivos de conversión.

Las redes de comunicación industrial en Cuba tienen la misma topología de estas redes de comunicación a nivel mundial, por lo que se necesita contar con este tipo de conversores. En nuestro país, al no contar con producciones locales de este tipo de dispositivos, es necesario realizar la adquisición de los mismos en el mercado mundial, muchas veces a través de terceros países, lo que incrementa su costo.

Tratando de darle solución a este problema se han hecho algunos intentos de fabricación que han dado resultado, ejemplo de ello se ve reflejado en la empresa de Servicios Técnicos de Computación, Comunicación y Electrónica del Níquel (SERCONI) en Holguín, que produjo un conversor de protocolo de RS-232 a RS-485. Este dispositivo a pesar de sus buenos resultados en el desempeño a nivel industrial no satisface la demanda creciente en cuanto a la necesidad de interconexión de redes series con redes *ethernet*.

Un ejemplo donde se evidencia este tipo de problema es en el sector turístico, específicamente en el monitoreo de las instalaciones, las habitaciones y las cámaras de frío. Estos procesos en la mayoría de los casos son supervisados por tarjetas de control, que solo cuentan con una interfaz serie para la comunicación, necesitando así, un conversor de este tipo por cada elemento a supervisar en el sistema. Añadir a esto que el costo promedio de cada conversor en el mercado oscila entre los $200 y los $600 USD.

Otro ejemplo se ve reflejado en el sector del Ministerio del Interior (MININT), específicamente en las unidades de guarda fronteras que atienden los sistemas de radares. La supervisión de este sistema se realiza a través de una aplicación que monitorea las embarcaciones que necesitan comunicarse con el sistema de radares,

que en este caso cuenta con una interfaz serie, implicando que se necesite de estos dispositivos de conversión.

A demás del costo, está el tema de la seguridad que implica obtener estos dispositivos fabricados por compañías extranjeras que pueden introducirle puertas traseras para sabotear los sistemas o piratear información. Ejemplo de esto es el sabotaje a la empresa Petróleos de Venezuela SA (PDVSA), en el 2002, donde penetraron en la redes de comunicaciones industriales generando pérdidas de millones de dólares.

En la Universidad de la Ciencias Informáticas (UCI), específicamente en el Centro de Informática Industrial (CEDIN), dentro de la línea de Sistemas Empotrados, se creó el proyecto de conversores de datos para tratar de resolver este tema y ayudar a lograr la soberanía tecnológica.

Por lo antes planteado surge el siguiente **problema a resolver**: ¿Cómo lograr la comunicación entre dispositivos sobre un bus serie y sistemas sobre redes *ethernet*?

Se define como **objeto de estudio** de la investigación, las Redes de Comunicación Industrial. Teniendo como **objetivo general**, desarrollar un conversor de protocolo RS-232 a *ethernet* soportado por el microcontrolador STM32F107VCT. Siendo el **campo de acción** los Conversores de Datos.

Para dar solución a lo antes planteado se definen las siguientes **tareas de investigación:**

1. Elaboración del marco teórico a partir del estado del arte existente de las normas, los protocolos y procedimientos estandarizados por los fabricantes de este tipo de dispositivos para realizar un correcto diseño de la solución.
2. Caracterización del microcontrolador STM32F107VCT para poder definir las herramientas a utilizar en el desarrollo de la solución.
3. Investigación de las herramientas y tecnologías a utilizar en el desarrollo que brinda el fabricante para seleccionar las más idóneas para nuestro trabajo.
4. Implementación de las capas de transporte serie y *ethernet* que serán embebidas en el microcontrolador STM32F107VCT para gestionar la comunicación.
5. Implementación de la lógica de conversión entre las dos interfaces de comunicación para lograr la conversión de una interfaz serie a una TCP/IP[6].

[6] TCP/IP: Protocolo de Control de Transmisión/Protocolo de Internet (*Transmission Control Protocol/Internet Protocol*).

6. Integración de la aplicación de configuración con el firmware de los conversores para lograr un producto terminado.
7. Realización de pruebas de validación de la aplicación de configuración para determinar el correcto funcionamiento de la misma.

Con el cumplimiento de los elementos antes planteados se espera obtener como **posibles resultados** el Firmware del conversor de protocolo RS-232 a *ethernet* que ejecutará el microcontrolador STM32F107VCT.

Para la realización de la investigación y elaboración del presente trabajo se emplearon varios **métodos científicos de investigación**, entre los cuales se pueden mencionar los siguientes:

Métodos teóricos que sustentan el trabajo de investigación:

Histórico – Lógico: Este método es utilizado para realizar el estudio de las tendencias históricas y actuales en cuanto al desarrollo de conversores de protocolos de comunicación.

Analítico – Sintético: Se utiliza para el estudio de cada una de las características y el desempeño de los distintos módulos que integrarán el *firmware*, así como la interrelación de estos para soportar las funcionalidades que deben ser implementadas.

Modelación: Es empleado en la comprensión profunda sobre cómo se comporta el hardware del microcontrolador STM32, estableciendo ese conocimiento como punto de partida para el desarrollo del *firmware.*

Métodos empíricos:

Consulta de fuentes de información: Es utilizado en la consulta, comparación e identificación de fuentes bibliográficas especializadas y de actualidad que brinden claridad y aportes durante la investigación.

Experimento: Es utilizado para comprobar el resultado mediante pruebas de aceptación.

La estructura del documento está definida de la siguiente manera:

Capítulo 1. Fundamentación teórica: Se exponen los conceptos fundamentales relacionados con el tema de investigación y se presentan las bases teóricas fundamentales relacionadas con los conversores de protocolo RS-232 a *ethernet*. Se abunda en los conceptos de Comunicación serie, TCP/IP, UDP y *ethernet*. Además se hace un estudio sobre los principales fabricantes de conversores, los tipos de comunicaciones industriales y las funcionalidades que brindan los mismos.

Capítulo 2. Herramientas y metodologías utilizadas para la solución: Se mencionan las herramientas de desarrollo y la metodología empleada en la realización de este trabajo. Además se realiza una caracterización del microcontrolador STM32F107VCT, realizando un estudio de los drivers de bajo nivel brindados por el fabricante para el desarrollo de las aplicaciones. Se aborda también las características del sistema operativo en tiempo real FreeRTOS y su vinculación con la biblioteca TCP/IP, LwIP, para el manejo de las comunicaciones sobre *ethernet*.

Capítulo 3. Discusión y resultado de la solución propuesta: En este capítulo queda registrado el diseño de la solución propuesta, la arquitectura, así como los artefactos generados a partir de la aplicación de la metodología XP. Se exponen los casos de pruebas realizados al sistema.

Capítulo1 Fundamentación teórica

Este capítulo hace alusión a los conceptos fundamentales relacionados con el tema de investigación y se presentan las bases teóricas relacionadas con los conversores de protocolo RS-232 a *ethernet*. Se expone el estudio realizado a la familia de conversores de protocolos, NPort 6000, del fabricante Moxa. Finalmente se abordan temas relacionados con los microcontroladores, sistemas embebidos y sistemas operativos en tiempo real.

1.1 Características de las comunicaciones industriales.

La comunicación se ha hecho imprescindible en la industria moderna. Muchos sistemas están conformados por equipos de diferentes fabricantes y funcionan en diferentes niveles de automatización. Pese a que pueden estar distanciados entre sí, a menudo se desea que trabajen de forma coordinada para un resultado satisfactorio del proceso. El objetivo principal es la comunicación totalmente integrada en el sistema. Esto reporta la máxima flexibilidad y permite integrar sin problemas productos de otros fabricantes a través de las interfaces estandarizadas. Esta integración total se conoce como CIM[7] (Hassan y Bright, 2012).

Las comunicaciones industriales están estructuradas a nivel de campo en forma de canales de comunicaciones que cumplen ciertas y determinadas normas que deben su origen a la creación de los buses de campo como las redes digitales bidireccionales, multipunto, montadas sobre un bus serie que conectan dispositivos de campo como transductores, actuadores, sensores, controladores de velocidad, terminales de operador con los sistemas de control campo (Verhappen y Byres, 2006).

En los últimos años las aplicaciones industriales basadas en comunicación digital se han incrementado haciendo posible la conexión de sensores actuadores y equipos de control. De esta manera, la comunicación entre la sala de control y los instrumentos de campos se han convertido en realidad (Qutaiba y Basil, 2011). La comunicación digital debe integrar la información provista por los elementos de campos en el sistema de control de procesos. De ahí a que sea tan importante el proceso de captación de información proveniente de los dispositivos de campos que implementan una interfaz

[7] CIM: Manufactura Integrada por Computador (*Computer Integrated Manufacturing).*

serial con los ordenadores en la sala de control para su monitorización. Para que este proceso de captación de datos sea posible, debe existir un dispositivo clave dentro de las redes de comunicación industrial capaz de comunicar un dispositivo con interfaz serie, con los ordenadores con interfaz *ethernet*.

1.2 Protocolos de Comunicación.

Un protocolo es un conjunto de reglas y procedimientos que establece métodos que regulan las comunicaciones y cómo será el intercambio de datos. Especifica y describe cómo dos dispositivos intercambiarán datos y órdenes durante la comunicación (Briceño, 2005).

Un protocolo determina:

- Cómo el dispositivo que envía debe indicar que ha enviado un mensaje.
- Cómo indicará el dispositivo que recibe, que ha recibido un mensaje.
- Qué se utilizará para comprobar errores.
- Cuál será el método de compresión de los datos, si es que los hay (Rodriguez, 2013).

Funcionamiento de los Protocolos

Los protocolos se llevan a cabo en el siguiente orden:

1. El mensaje se divide en secciones más pequeñas denominadas *paquetes*.
2. Se añaden los paquetes de información sobre la dirección, de forma que el equipo destino puede determinar si el paquete le pertenece.
3. Se preparan los datos para transmitir y enviarlos a través de la red.

Existen protocolos que están soportados por sobre normas eléctricas series que permiten el uso de conversores para enviar sus tramas a través de redes de *ethernet*. Los clientes de protocolos SCADA que implementan estos, permiten que se use el protocolo serial a través de un conversor serie a *ethernet*.

1.3 Protocolos TCP/IP. Características fundamentales.

El protocolo TCP/IP es uno de los más importantes en Internet (Jhonson, 2008). TCP se encarga de controlar, ordenar y manejar los datos, además de asegurarse de que lleguen a su destino sin que se pierdan paquetes. Al mismo tiempo actúa tanto como emisor y receptor de paquetes. El TCP emisor divide la información a transferir en

paquetes y envía cada uno por separado, retransmitiéndolos si alguno no se entrega correctamente. Si es necesario el receptor TCP tendrá que reordenar los paquetes secuencialmente (cada paquete tiene un número de secuencia de 32 bits, que es un campo de la cabecera) para poder obtener la información.

TCP permite que varias aplicaciones en una misma máquina puedan comunicarse simultáneamente. Para este propósito el protocolo TCP incorpora los denominados puertos que son usados para identificar la conexión entre dos dispositivos o aplicaciones. Cada puerto tiene asociado un entero de 16 bit (1 - 65535) para identificarlo. Dado que el número de puertos es único dentro de una máquina, el destino último para el tráfico TCP queda perfectamente determinado por la dirección IP del "host" y el número de puerto asignado a la conexión (Augé, 2000).

Por otra parte, el protocolo IP tiene como misión, hacer llegar los fragmentos de información a su destino correcto. TCP toma los datos y los fragmenta en paquetes de no más de 64KB. IP transporta esos datagramas, fragmentándolos en trozos más pequeños, a su destino, donde el protocolo TCP los reordena y ensambla para volver a formar los datos originales (Stallings, 2000). Los datagramas IP tienen una cabecera, con una parte fija de 20 bytes y una parte variable que contiene la dirección IP destino y origen entre otros campos.

La identificación de un *host* bajo el direccionamiento del IPv4 puede estar conformada por los siguientes campos:

- **Dirección IP:** Es la identificación única que el protocolo IP hace de una máquina y de la red a la que pertenece. Es un número de 32 bits, representado por 4 bloques de 8 bits separados por puntos.
- **Puerta de enlace:** Es la dirección de IP de un dispositivo que permite dotar a las máquinas de una red local (LAN[8]) conectadas a él de un acceso hacia una red exterior.
- **Máscara de red:** Es una dirección IP cuyos bits activos (1) son los empleados para identificar la red y cuyos bits no activos (0) forman parte de la identificación de cada ordenador dentro de esa red.

TCP está contenido dentro de las capas del Modelo OSI.

[8] LAN: Red de Área Local (*Local Area Network*).

1.3.1 Modelo OSI.

El modelo de interconexión de sistemas abiertos, también llamado OSI[9] es el modelo de red descriptivo, que fue creado por la ISO[10], en el año 1984.

Es un marco de referencia para la definición de arquitecturas en la interconexión de los sistemas de comunicaciones. Es decir, se trata de una normativa estandarizada útil debido a la existencia de muchas tecnologías, fabricantes y compañías dentro del mundo de las comunicaciones, y al estar en continua expansión, se tuvo que crear un método para que todos pudieran entenderse de algún modo, incluso cuando las tecnologías no coincidieran (Stallings, 2000).

El modelo OSI, elaborado para describir protocolos para una sola red, no contiene un nivel específico para el ruteo en el enlace de redes, como sucede con el protocolo TCP/IP.

1.4 Comunicación Serie.

La comunicación serie, es el proceso de envío de datos de un bit por vez, secuencialmente, sobre un canal de comunicación. La comunicación serial es un protocolo muy común entre dispositivos que se incluye de manera estándar en prácticamente cualquier computadora (Instruments, 2013).

El puerto serial envía y recibe bytes de información un bit a la vez. Siendo más lento que la comunicación en paralelo, que permite la transmisión de un byte completo por vez, este método de comunicación es más sencillo y puede alcanzar mayores distancias.

Para realizar la comunicación se utilizan 3 líneas de transmisión:

- Tierra (o referencia).
- Transmitir.
- Recibir.

Debido a que la transmisión es asincrónica, es posible enviar datos por una línea, mientras se reciben datos por otra.

[9] OSI: Interconexión de sistemas abiertos (*Open System Interconnection*).
[10] ISO: Organización Internacional de Normalización (*International Standarization Organization*).

Ejemplos:

- RS-232: Norma eléctrica que especifica las características de la comunicación serie de baja velocidad.
- RS-485/422: Norma eléctrica que especifica las características de la comunicación serie de baja velocidad, implementado en las comunicaciones industriales para lograr una mayor reducción del ruido en la transmisión.

Ambas normas eléctricas especifican una serie de parámetros que deben ser definidos de igual forma en los dispositivos que se van a comunicar. Puesto que las normas no permiten indicar en qué modo se está trabajando, es el usuario quien tiene que decidirlo y configurar ambas partes. Dichos parámetros son:

- **Velocidad de transmisión (baud rate):** Indica el número de bits por segundo que se transfieren, y se mide en baudios (bauds). Las velocidades de transmisión más comunes para las líneas telefónicas son de 14400, 28800, y 33600.
- **Bits de datos:** Se refiere a la cantidad de bits en la transmisión. Cuando la computadora envía un paquete de información, el tamaño de ese paquete no necesariamente será de 8bits. Las cantidades más comunes de bits por paquete son 5, 7 y 8 bits. El número de bits que se envía depende del tipo de información que se transfiere.
- **Bits de parada:** Usado para indicar el fin de la comunicación de un solo paquete. Los valores típicos son 1, 1.5 o 2 bits. Dan un margen de tolerancia para la diferencia en los relojes. Mientras más bits de parada se usen, mayor será la tolerancia a la sincronía de los relojes, sin embargo la transmisión será más lenta.
- **Bit de Paridad:** Con este bit se pueden descubrir errores en la transmisión. Se puede dar paridad par o impar. En la paridad par, por ejemplo, la palabra de datos a transmitir se completa con el bit de paridad de manera que el número de bits 1 enviados es par.

1.5 Conversores de Serie a Ethernet. Características principales.

Los conversores de serie a *ethernet* permiten comunicar equipos que se encuentran sobre una red serie con equipo que están sobre una red *ethernet*. Para ello cuentan con un elemento inteligente capaz de manejar las conexiones TCP/IP asociadas a un puerto serie. Su funcionamiento básico consiste en desensamblar la trama TCP/IP, recibida a

través de la conexión, obteniendo los datos para enviarlos por el puerto serie correspondiente. Los datos obtenidos a través del puerto serie son enviados a ethernet por la conexión TCP/IP asignada (Moxa, 2011).

Desde el punto de vista de la arquitectura se pueden considerar los conversores de datos como sistemas embebidos, puesto que en la mayoría de los casos la gestión de la lógica de conversión se realiza mediante un microcontrolador, que cuenta con un firmware en el que se integra una biblioteca TCP/IP, un sistema operativo en tiempo real, así como los manejadores de los periféricos con que cuenta el microcontrolador.

1.6 Principales fabricantes de conversores. Características de su línea de productos.

En el mundo existen disímiles fabricantes de conversores de protocolo, entre ellos se encuentra Moxa en la Tabla I-I se muestran las características de algunas de sus líneas de productos (Moxa, 2013).

Tabla I-I. Fabricantes y características.

Series	CN2610/2650	NPort 6250	NPort 6450
Number of Ports	8 or 16-port RS-232/422/485 terminal servers with LAN redundancy.	2-port RS-232/422/485 secure terminal servers.	4-port RS-232/422/485 secure terminal server.
Serial Communication Parameters	Data Bits 5, 6, 7, 8 Stop Bits 1, 1.5, 2	Data Bits 5, 6, 7, 8 Stop Bits 1, 1.5, 2	Data Bits 5, 6, 7, 8 Stop Bits 1, 1.5, 2
Software	Network Protocols ICMP, IP, TCP, UDP, DHCP, BOOTP, Telnet, DNS, SNMP V1/V2c/V3, HTTP, SMTP, ARP, PPPoE, DDNS.	Network Protocols ICMP, IP, TCP, UDP, DHCP, BOOTP, Telnet, DNS, SNMP V1/V2c/V3, HTTP, SMTP, ARP, PPPoE.	Network Protocols ICMP, IP, TCP, UDP, DHCP, BOOTP, Telnet, DNS, SNMP V1/V2c/V3, HTTP, SMTP, ARP, PPPoE.

Operation Modes	Standard Real COM, TCP Server, TCP Client, UDP, RFC2217, Terminal, Reverse Telnet, PPP, DRDAS, Redundant COM, Disabled.	Standard Real COM, TCP Server, TCP Client, UDP, Pair Connection, RFC2217, Terminal, Reverse Telnet, ethernet Modem, Printer, PPP, Disabled.	Standard Real COM, TCP Server, TCP Client, UDP, Pair Connection, RFC2217, Terminal, Reverse Telnet, ethernet Modem, Printer, PPP, Disabled.

A continuación se muestran las características generales de los conversores pertenecientes a la serie Nport 6000 del fabricante Moxa.

1.6.1 Serie NPort 6000.

La serie NPort 6000 actualmente posee diecisiete modelos que se diferencian entre sí mediante el número de puertos y el tipo de conexión de red empleada. Se puede utilizar para conectar cualquier dispositivo serial a una red *ethernet*, y es compatible con diferentes modos de operación, además permite el uso de transmisión no estandarizado. Todos los dispositivos pertenecientes a esta serie, cuentan con una amplia gama de parámetros configurables que permiten adaptar el dispositivo a los requerimientos de una solución específica. El fabricante permite realizar la configuración de dichos parámetros a través de varios métodos bien sea por una consola web, una aplicación de escritorio mediante una interfaz serie o mediante Telnet. Dados los objetivos de este trabajo se realizó el estudio de la consola web (Moxa, 2011).

Una vez seleccionada en el menú los parámetros básicos de la red se despliega una página que permite establecer características del dispositivo tales como: el nombre del servidor, ubicación del servidor, zona horaria, tiempo local y tiempo del servidor. Estos parámetros permiten un mejor mantenimiento así como una trazabilidad en cuanto al tiempo de los eventos de error que puedan ocurrir.

Cada puerto serie puede configurarse independientemente. Para realizar la configuración de los modos así como de los parámetros del puerto serie, se despliega el

menú con la opción de “Parámetros del puerto serie” y se selecciona el puerto que se desea configurar.

El primer parámetro que se debe configurar es el modo de funcionamiento que se desea establecer para el puerto seleccionado. Esta gama de dispositivos provee las siguientes posibilidades:

- Modo COM real.
- Modo COM real inverso.
- Modo RFC2217.
- Modo TCP servidor.
- Modo TCP cliente.
- Modo UDP.
- Modo SSH.
- Modo ASCII.
- Modo binario.

De todos estos modos se analizan a continuación el modo TCP servidor y el modo TCP cliente puesto que son los de interés para el producto que se desarrolla.

Para establecer el modo TCP servidor se deben configurar los siguientes parámetros:

- **Tiempo de chequeo del estado de la conexión:** Este campo, especifica el tiempo que se esperará una respuesta ante el envío de un paquete (“keep alive”). El NPort 6000 comprueba el estado de la conexión mediante el envío periódico de paquetes (“keep alive”). Si el host remoto no responde al paquete en el plazo especificado se forzará al cierre de la conexión TCP existente.
- **Tiempo de inactividad:** Este campo especifica el tiempo que el NPort 6000 esperará la entrada y salida de datos a través del puerto serie antes de cerrar la conexión TCP. La conexión TCP se cierra si no hay datos entrantes o salientes a través del puerto serial pasado el tiempo de inactividad especificado. Si este campo se establece en 0, la conexión TCP se mantiene activa hasta que una solicitud de cierre de conexión se recibe.
- **Tiempo de respuesta:** Este campo especifica el tiempo que el NPort 6000 esperará a que los datos de respuesta a través del puerto serie antes de enviar el siguiente comando. El NPort 6000 envía el comando siguiente si hay respuesta a través del puerto serie durante el tiempo especificado por el tiempo de espera de respuesta. Si este campo se establece en 0, el tiempo de espera de respuesta es esencialmente infinito, y el NPort 6000 esperará hasta que la

solicitud del comando tenga una respuesta y se reciba para enviar el siguiente comando.

- **Puerto TCP:** Es el número del puerto TCP asignado a la conexión del puerto serie del NPort 6000. Es el número del puerto que el puerto serie utiliza para escuchar por una conexión y que los otros dispositivos usan para conectarse con el puerto serie. Para evitar conflictos con los valores conocidos de puertos TCP se inicializa a 4001.

El modo TCP cliente mantiene los parámetros del modo TCP servidor y se le agrega el siguiente parámetro:

- **Dirección IP destino:** Se debe especificar una dirección IP, que permite al NPort 6000 conectarse activamente con el host remoto. Al menos un destino debe ser proporcionado.

Otra de las propiedades que se deben establecer a los puertos series son los parámetros de comunicación en el NPort 6000. Estos parámetros deben coincidir con los parámetros utilizados por el dispositivo que será conectado al NPort 6000. Dichas propiedades son:

- Velocidad de transmisión
- Bits de datos
- Bits de parada
- Paridad
- Control de flujo

1.7 Microcontroladores. Aplicaciones y Características principales.

En la actualidad, existen computadores alrededor de nosotros haciendo cálculos silenciosamente sin interactuar con ningún humano. Estas computadoras pueden encontrarse en su auto, en un juguete e incluso en el mando del televisor de su casa. A estos dispositivos los llamamos Microcontroladores. Micro porque son unidades pequeñas, y controlador porque controlan máquinas o incluso otros controladores (Wilmshurt, 2007).

1.7.1 Definición.

La generalidad de las definiciones de “microcontrolador” planteadas por los expertos sobrepasa el ámbito electrónico y presentan a los microcontroladores como computadoras miniaturizadas. Las siguientes definiciones aclaran lo mencionado anteriormente:

- Es un sistema de control completo basado en un microprocesador pero construido en un solo chip (Crisp, 2004).

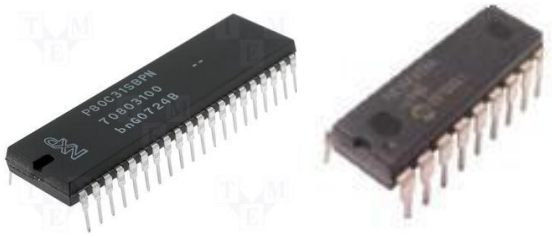


Figura I-I. Microcontroladores a) NXP b) PIC

En resumen, un microcontrolador es un circuito integrado que contiene todos los componentes de un computador. Se emplea para controlar el funcionamiento de una tarea determinada y debido a su reducido tamaño, suele ir incorporado en el propio dispositivo al que gobierna. Esta última característica es la que le confiere la denominación de «controlador incrustado» *(embedded controller*). Se dice que es “la solución en un chip” porque su reducido tamaño minimiza el número de componentes y el costo (Velázquez y Guerrero, 2009).

En su interior incluye los tres bloques fundamentales de una computadora: CPU, Memoria y Unidades Entrada/Salida. En su memoria sólo reside un programa destinado a gobernar una aplicación determinada; sus líneas de entrada/salida soportan la conexión de sensores y actuadores del dispositivo a controlar. Una vez programado y configurado el microcontrolador solamente sirve para gobernar la tarea asignada (Wilmshurt, 2007).

1.7.2 Características.

Los microcontroladores son diseñados para reducir el costo económico y el consumo de energía de un sistema en particular, sus componentes son elegidos con el objetivo de reducir su tamaño y producirlos tan baratos como sea posible. Es válido mencionar su alto grado de integración, su simplificación de periféricos adicionales y su cada vez

mayor capacidad y velocidades de ejecución, así como su mínimo consumo de potencia a un precio razonable.

Entre sus principales características podemos encontrar que son:

- **Empotrados:** Se encuentran "empotrados" en el interior de dispositivos para controlar las características o acciones de este (Calcutt et al., 2004).
- **Dedicados:** Están programados, generalmente, para realizar una sola tarea y ejecutar un programa en específico, el que se almacena en la memoria no volátil y normalmente no varía (Zurrell, 2000).
- **Bajo consumo de energía:** Un microcontrolador que funcione con una batería consume aproximadamente 50 miliwatts (Calcutt, Cowan y Parchizadeh, 2004). Que sean bajo consumidores de energía garantiza un funcionamiento a largo plazo en ambientes extremos.

Al microcontrolador se le diseña de tal manera que tenga todos los componentes integrados en el mismo chip. No necesita de otros componentes especializados para su aplicación, porque todos los circuitos necesarios, que de otra manera correspondan a los periféricos, ya se encuentran incorporados. Así se ahorra tiempo y espacio necesario para construir un dispositivo.

1.7.3 Arquitectura de los microcontroladores.

En cuanto al modo de ejecución estas arquitecturas están clasificadas en Harvard que propone memorias separadas para datos y programas, por otra parte, Von Neumann que establece una sola memoria para su uso indistinto de programas y datos, ver Figura I-II. Ambas se diferencian en la forma de conexión de la memoria al procesador y en los buses que cada una necesita por lo que en esta clasificación es importante conocer como está dispuesto el bus de direcciones y el bus de datos (Gridling y Weiss, 2007).

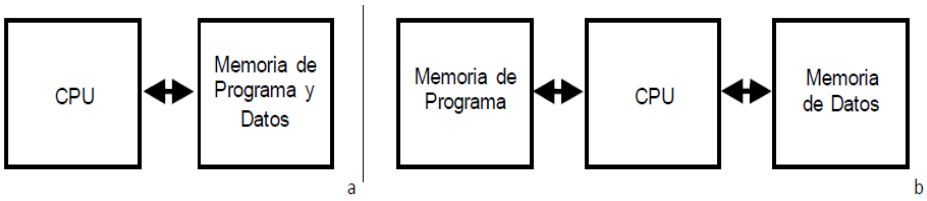


Figura I-II. Arquitectura de Microcontroladores: a) Von Neumann b) Harvard

Harvard conocida como arquitectura de ejecución paralela, posee varias unidades de ejecución. Divide los procesos, está orientada a sistemas multitareas y los buses de dirección y de datos están separados (Jasio, 2007).

La ventaja fundamental de esta arquitectura es que permite adecuar el tamaño de los buses a las características de cada tipo de memoria; además, el procesador puede acceder a cada una de ellas de forma simultánea, lo que se traduce en un aumento significativo de la velocidad de procesamiento. Típicamente los sistemas con esta arquitectura pueden ser dos veces más rápidos que sistemas similares con arquitectura Von Neuman (Wilmshurt, 2007).

1.7.4 Componentes y periféricos.

Existe una variada serie de componentes de los microcontroladores, a continuación se hace referencia a algunos de ellos.

Puertos Entrada/Salida: Los puertos de E/S permiten al microcontrolador establecer una comunicación y controlar, a través de interfaces u otros dispositivos, a relés, interruptores, teclados, pantallas LCD[11], sensores, etc. Pueden ser unidireccionales o bidireccionales además de que estos tienen que ser mapeados y deben configurarse a través de un registro específico. Estos puertos son la principal utilidad de los pines de un microprocesador. Dentro de ellos se encuentran los puertos de comunicación:

- **Puerto serie:** Este periférico existe en los microcontroladores como UART [12] o USART dependiendo de sí permiten o no el modo sincrónico de comunicación. El objetivo de este periférico es la comunicación con otro microcontrolador o con una computadora. Se implementa a través de las interfaces RS-232, RS-485, RS-422.
- **Ethernet:** La interfaz de comunicación *ethernet* se utiliza mayormente para la comunicación del microcontrolador con otro microcontrolador o con una computadora. Es muy útil en la integración de los sistemas de control basado en microcontroladores para realizar tareas de monitoreo y control remoto de los sistemas a través de HMI (siglas en inglés de Human – Machine Interface, interfaz hombre - máquina).
- **Universal Serial Bus (USB):** Mientras que los periféricos sintetizados anteriormente se usan indistintamente para la comunicación entre microcontroladores, microcontroladores con otros periféricos o con una computadora, la interfaz USB normalmente se utiliza para la comunicación del microcontrolador con una computadora y con la finalidad de acceder a memoria no volátil en los casos que esto sea técnicamente posible.

[11] LCD: Monitor de Cristal Líquido (*Liquid Crystal Display*).
[12] UART: Receptor Transmisor Universal Asincrónico (*Universal Asynchronou Receiver Transmitter*).

- **Memoria no volátil**: La memoria del microcontrolador es un lugar donde se almacenan las instrucciones del programa y los datos que manipula. Anteriormente se mencionó que en los microcontroladores el programa normalmente se almacena en una memoria no volátil (ROM [13]), conocida generalmente como la memoria de programas pues es donde están contenidas las instrucciones a ejecutar por el CPU. Para soportar esta función existen varios tipos de memorias no volátiles. Las más utilizadas actualmente para la programación de microcontroladores son:
  - ✓ **EEPROM:** La EEPROM (*Electrically Erasable Programmable Read-Only Memory).* Constituye una memoria de sólo lectura, programable y borrable eléctricamente. Tanto el grabado como el borrado, se realizan eléctricamente desde una computadora a través de una interfaz. El número de veces que pueden realizarse ambas operaciones en una memoria EEPROM es finito, por lo que no es aconsejable una reprogramación continua.
  - ✓ **Flash:** Se trata de una memoria no volátil de bajo consumo energético, que se puede escribir y borrar. Funciona como una ROM y una RAM[14], pero consume menos energía y es más pequeña. Es más rápida y de mayor densidad que la EEPROM. La alternativa FLASH está recomendada frente a la EEPROM cuando se precisa gran cantidad de memoria de programa no volátil. Es más veloz y tolera más ciclos de escritura/borrado. Las memorias EEPROM y FLASH son muy útiles al permitir que los microcontroladores que las incorporan puedan ser reprogramados en circuito, es decir, sin tener que sacar el circuito integrado de la tarjeta.
- **Memoria volátil**: El almacenamiento de datos y de variables se realiza en la memoria volátil (RAM). La RAM memoria de lectura y escritura pierde la información almacenada cuando falta la energía que alimenta la memoria.

1.8 Sistemas Empotrados.

Son dispositivos usados para controlar equipos, operación de maquinarias o plantas industriales completas. El término “embebido” (también se lo conoce como “incrustado” o “embutido”) está caracterizando que esos circuitos integrados, son una parte integral del sistema en que se encuentran.

[13] ROM: Memoria de sólo Lectura (*Read Only Memory*).
[14] RAM: Memoria de Acceso Aleatorio (*Random Access Memory*).

Los microcontroladores comúnmente vienen alojados en dispositivos o equipos formando parte de sistemas empotrados:

> *Los sistemas empotrados son sistemas informáticos con el hardware fuertemente acoplado y con integración de software, que están diseñados para llevar a cabo una función específica. La palabra empotrado refleja el hecho de que estos sistemas son generalmente una parte integral de un sistema más grande, conocido como el sistema de empotramiento (Qing, 2003).*

Y aporta los siguientes elementos:

> *Están construidos para controlar una función o serie de funciones y no están diseñados para ser programados por el usuario final de la misma manera en que lo es una PC[15] (Heath, 2003).*

En resumen un sistema empotrado no es más que aquella aplicación o software que se diseña e implementa para ejecutarse en dispositivos (generalmente microcontroladores) y que se enfrenta a fuertes restricciones de hardware. Combina interfaces o periféricos de entrada un procesador interno, software de ejecución y periféricos de salida, además poseen una cantidad de recursos físicos, como memorias, discos duros, procesadores etc., están diseñados para realizar variadas funciones, como por ejemplo funciones repetitivas (Wilmshurt, 2007).

1.9 Sistemas operativos en tiempo real.

Los Sistemas Operativos de Tiempo Real (RTOS) son diseñados para determinar la respuesta a sucesos o eventos, de forma tal, que se respeten las restricciones de tiempo de respuesta, los cuales, en el caso de ser críticos para el sistema, requieren de atención inmediata. En todo caso el RTOS son deterministas, en consecuencia permite que el usuario prediga los tiempos de respuesta que el RTOS proporcionara al sistema informático que administra. Estos eventos, normalmente se producen en el entorno del sistema; llegan como interrupciones al procesador, ya sea por software o hardware. Los RTOS deben atender ráfagas de miles de interrupciones que activan diferentes procesos, Para no perder ningún suceso recibido, el RTOS debe trabajar con la técnica multiproceso, que ejecuta los procesos independientes unos de otros, pero en ciertas aplicaciones se deben tener estrategias para asignar prioridad a los procesos y

[15] PC: Computadora Personal (*Personal Computer*).

ejecutarlos según su importancia, la cual se logra con una planificación basada en prioridades (Rubio, 2005).

La funcionalidad principal y requerida para un sistema operativo de tiempo real es proveer de un nivel de servicio adecuado a las aplicaciones que requieran una respuesta en un intervalo de tiempo determinado.

1.9.1 Características de un Sistema Operativo en Tiempo Real.

La característica principal de un sistema operativo de tiempo real es la respuesta ante eventos internos o externos, tales como interrupciones hardware externas, interrupciones software internas o interrupciones de reloj internas, es decir los requerimientos temporales.

- Las características que los RTOS deben ofrecer son:
- Soporte para la planificación de procesos en tiempo real.
- Planificación por prioridad.
- Garantía de respuesta ante interrupciones.
- Comunicación interprocesos.
- Adquisición de datos a alta velocidad.
- Soporte de E/S.
- Control, por parte del usuario, de los recursos del sistema.

Una de las medidas de rendimiento de un Sistema Operativo de Tiempo Real es la latencia, o tiempo desde que ocurre el evento hasta que este es tratado. La otra medida es el *jitter*, o variaciones en el período normal de ocurrencia de eventos periódicos. Todos los sistemas operativos tienden a tener una baja latencia y un bajo *jitter*, pero los sistemas operativos de tiempo real requieren que esos valores estén determinados y que no dependan de la carga del sistema (Zamarrón, 2004).

Un sistema operativo es determinista si realiza las operaciones en instantes fijos y predeterminados o en intervalos de tiempos predeterminados. Cuando compiten varios procesos por los recursos y por el tiempo del procesador, depende, en primer lugar, de la velocidad con la que pueda responder a las interrupciones y en segundo lugar, de si el sistema posee suficiente capacidad para gestionar todas las peticiones en el tiempo requerido. Para operar de forma determinista el retardo máximo que se produce desde la llegada de la interrupción de un dispositivo de alta prioridad hasta que comienza el servicio del mismo, debe ser minimizado y conocido.

Otra de las características de este tipo de sistemas es la sensibilidad. El determinismo hace referencia a cuánto tiempo consume un sistema operativo en reconocer una interrupción. La sensibilidad se refiere a cuánto tiempo consume un sistema operativo en dar servicio a la interrupción después de reconocerla. Las características de la sensibilidad son, entre otras:

- La cantidad de tiempo necesario para iniciar la gestión de la interrupción y comenzar la ejecución de su rutina de tratamiento (ISR, *Interrupt Service Routine*).
- La cantidad de tiempo necesario para ejecutar la ISR. Generalmente, depende de la plataforma del hardware.
- El efecto del tratamiento de interrupciones. El servicio se retrasará si una ISR puede ser interrumpida por la llegada de otra interrupción.

El determinismo y la sensibilidad forman conjuntamente el tiempo de respuesta a sucesos externos. Los requisitos en tiempo de respuesta son críticos ya que cada sistema debe cumplir los requisitos de tiempo impuesto por los individuos, dispositivos y flujos de datos externos al sistema.

Otra de las distinciones de este tipo de sistemas en tiempo real es que el control de los usuarios es generalmente mucho mayor que en un sistema operativo ordinario. En un sistema operativo típico que no sea en tiempo real, el usuario no tiene control sobre la función de planificación del sistema operativo. En un sistema en tiempo real resulta esencial permitir al usuario un control preciso sobre la prioridad de las tareas. El usuario debe poder distinguir entre tareas rígidas y flexibles y especificar prioridades relativas dentro de cada clase. Un sistema en tiempo real también permitirá al usuario especificar características como por ejemplo, que procesos deben estar siempre residente en la memoria principal por mencionar alguno.

La tolerancia a los fallos es otra de las características que poseen estos sistemas y hace referencia a la propiedad de conservar la máxima capacidad y los máximos datos posibles en caso de fallos por ejemplo, un sistema UNIX clásico, cuando detecta datos corruptos en el núcleo, genera un mensaje de error en la consola del sistema, vuelca el contenido de la memoria en el disco para un análisis posterior y finaliza la ejecución del sistema. Un sistema en tiempo real intentará corregir el problema, o minimizar su efecto, mientras continúa la ejecución.

Para cumplir los requisitos anteriores los sistemas operativos actuales en tiempo real incluyen normalmente las siguientes características:

- Cambios rápidos de procesos o hilos.
- Pequeño tamaño (con una mínima funcionalidad asociada).
- Capacidad de responder rápidamente a interrupciones externas.
- Multitarea con herramientas de comunicación entre procesos, como semáforos, señales y sucesos.
- Uso de archivos secuenciales especiales que puedan acumular datos a alta velocidad.
- Planificación preferente basadas en prioridades.
- Reducción de intervalos en los que están inhabilitadas las interrupciones.
- Primitivas para demorar tareas durante un tiempo fijo y para detenerlas y reanudarlas.
- Alarmas especiales y temporizadores.

El corazón de un sistema en tiempo real es el planificador de tareas a corto plazo. Las aplicaciones de tiempo real normalmente necesitan tiempos de respuesta deterministas en un rango de varios milisegundos, las aplicaciones al límite, como los simuladores de aviones militares, por ejemplo presentan a menudo restricciones en un rango de diez a cien microsegundos (Zamarrón, 2004).

1.9.2 Aplicaciones de los sistemas operativos en tiempo real.

Dada las características de los sistemas operativos en tiempo real sus principales aplicaciones están en el área industrial.

- Dominio Industrial en la fabricación de sistemas de control, tarjetas de adquisición de datos, sensores inteligentes entre otros.
- En el área militar son muy utilizados también en el guiado de misiles, sistemas de navegación, procesamiento digital de imágenes así como en sistemas de comunicación.
- En el sector de la aviación es muy utilizado en todos los aspectos del control de vuelo entre otros.

Todos estos usos son frecuentemente como parte de un sistema embebido.

1.10 Conclusiones del Capítulo.

En este capítulo fueron analizados los conceptos y definiciones asociados al problema de la investigación. Se determinó que los conversores de datos son sistemas empotrados que cuentan con disímiles interfaces de comunicación, que son manejadas mediante un elemento inteligente conocido como microcontrolador. En este elemento radica una aplicación capaz de gestionar la conversión de datos entre las distintas interfaces soportadas por el sistema.

Los conversores de serie/*ethernet* cuentan con una capa MAC[16] gestionada a través de una biblioteca TCP/IP que se encarga del manejo de las conexiones. Dentro de los modos de trabajo soportados por este tipo de elemento están el modo cliente TCP y el modo servidor TCP. La asignación de los puertos serie se realiza asociando una conexión TCP al mismo.

Se determinaron las características principales de las series NPort 6000 del fabricante Moxa que constituye una referencia representativa de este tipo de conversores a nivel mundial. Con este estudio se determinaron las funcionalidades necesarias para el diseño de un conversor serie/*ethernet* en los modos de trabajo cliente o servidor TCP.

[16] MAC: Control de Acceso Medio (*Media Access Control*).

Capítulo2 Herramientas y metodologías utilizadas para la solución.

En este capítulo se seleccionan las herramientas y la metodología empleada para el desarrollo de la solución. Estas tecnologías, además de poseer características que se ajustan a las necesidades de la presente investigación, son ampliamente utilizadas en la línea Sistemas Empotrados, perteneciente al CEDIN.

2.1 Metodologías de Desarrollo de Software.

La Programación Extrema (XP) es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. Esta metodología se basa en la idea de que existen cuatro variables que guían el desarrollo de sistemas: Costo, Tiempo, Calidad y Alcance. La manera de encarar los desarrollos avalados por este modelo de desarrollo es permitir a las fuerzas externas (gerencia, clientes) manejar hasta tres de estas variables, quedando el control de la restante en manos del equipo de desarrollo (Krishna et al., 2011).

Es imposible prever todo antes de comenzar a programar; muchas veces se gasta demasiado tiempo y recursos en cambiar la documentación de la planificación para que se parezca al código. Para evitar esto, XP intenta implementar una forma de trabajo donde se adapte fácilmente a las circunstancias. Básicamente consiste en trabajar estrechamente con el cliente, haciendo pequeñas iteraciones (mini-entregas), cada dos semanas, donde no existe más documentación que el código en sí; cada versión contiene las modificaciones necesarias según el cliente vaya retroalimentando el sistema (por eso es necesaria la disponibilidad del cliente durante todo el desarrollo). Para suplir la falta de requisitos, casos de uso, y demás herramientas; XP utiliza historias de usuarios, la historia de usuario es una frase corta que representa alguna función que realizará el sistema. Cada historia de usuario no puede demorar en desarrollarse más de una semana, si así lo requiriera, debe segmentarse ver Figura II-I.

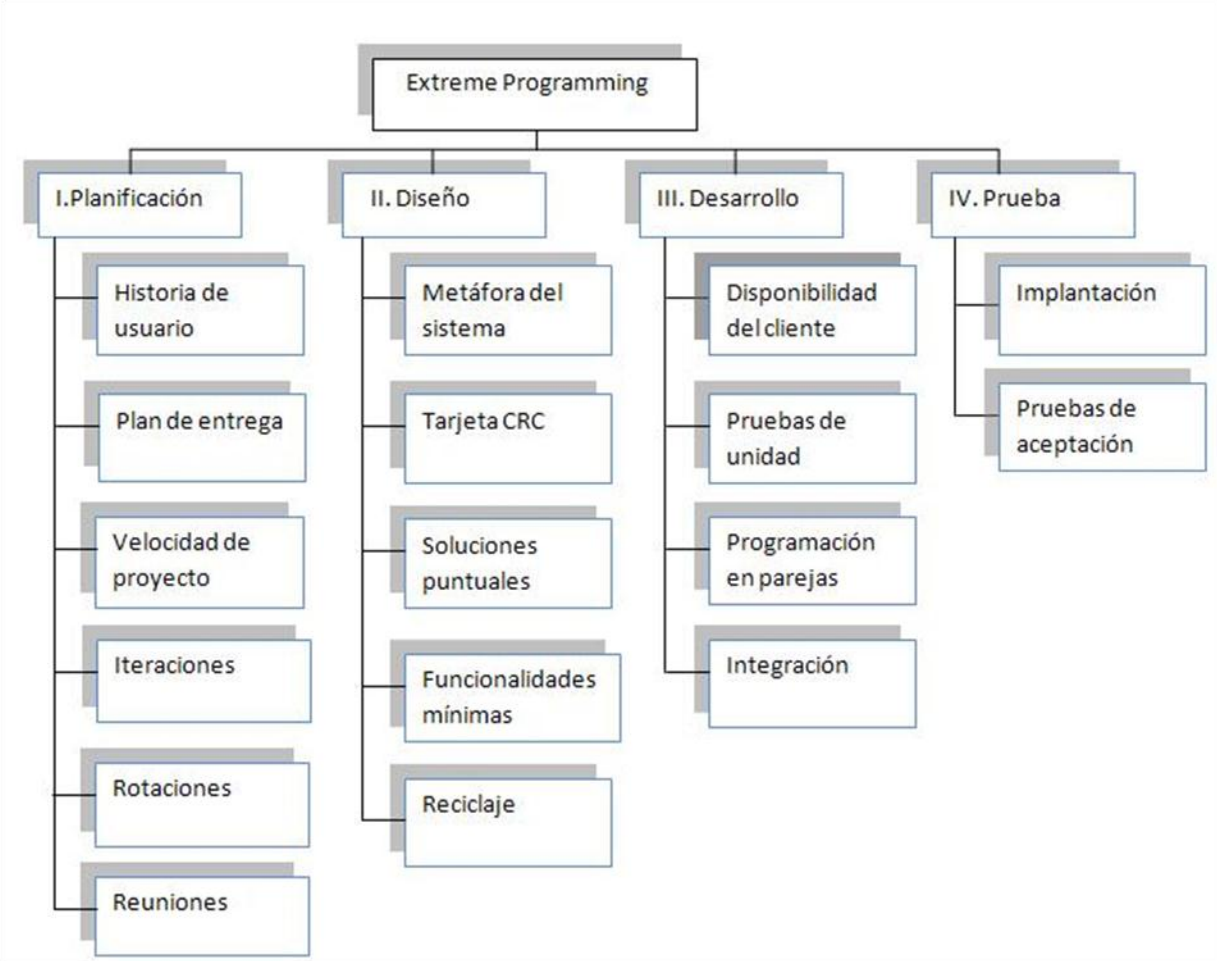


Figura II-I. Actividades propuestas por cada XP en cada una de las fases de desarrollo.

Es requisito para XP definir un estándar en el tipo de codificación, esto hace que los programadores tengan definido ya el estilo de programación y no que cada uno programe a su estilo. Las pruebas en cada iteración son más que importante; de eso se trata este paradigma de programación, corregir mientras se programa. De esta forma se van cubriendo todos los baches que cada versión padezca. El código no es de nadie, todo el equipo puede manipular el código que existe, de esta forma cada pareja puede mejorar cada sección de código que utiliza, esto requiere de una prueba del mismo y la re-implementación en el sistema general. Cada dos semanas se entrega una versión al cliente, que lo verifica, realiza la retroalimentación y se continúa el desarrollo; este ciclo continúa hasta que el sistema cumpla con las expectativas del cliente, acto que concluirá el proyecto (Informáticos, 2013).

La aplicación de esta metodología al desarrollo de sistemas embebidos se debe a que este proceso tiene entre sus características principales que la implementación de

software y la del *hardware* se realizan de forma paralela. Esto implica que muchas veces no se cuente con la solución de *hard* donde se debe empotrar el *soft* impidiéndole al equipo de programación comenzar el desarrollo desde el inicio del proyecto. En este período, se propone realizar de conjunto con el cliente, un levantamiento detallado de la solución, enmarcando este en pequeñas tareas que al empezar el desarrollo sean de fácil implementación, realizando pequeñas entregas que tributen a la solución final (Pierce, 2004). Al aplicar este mecanismo se obtiene como resultado un ahorro del tiempo empleado en la conceptualización de la solución (Ridi y Paulus Insap, 2012). Además de que los cambios introducidos por el cliente son de fácil resolución debido a que la implementación se realiza sobre la base de pequeñas historias de usuario (Drobka et al., 2004), (Ganssle, 2001), (Ionel NĂ y Ivona, 2009).

2.2 Hardware a utilizar.

El microcontrolador STM32F107VC pertenece a la familia ARM Cortex-M3 que es una de las generaciones de procesadores ARM más recientemente desarrollados para sistemas empotrados. Proporciona una plataforma de bajo costo que satisface las necesidades de implementación, con un número de pines reducido y bajo consumo de energía, al tiempo que ofrece excelente rendimiento computacional y un avanzado sistema de respuesta a las interrupciones. La arquitectura RISC del procesador ARM Cortex-M3 de 32 bits ofrece una excepcional eficiencia de código, proporcionando un alto rendimiento del núcleo ARM en el uso del tamaño de la memoria. Con el núcleo ARM integrado, el microcontrolador STM32F107VC, orientado a la conexión, es compatible con todas las herramientas y software de esta arquitectura. La amplia gama de periféricos incorporado en su diseño, hacen de este micro una herramienta poderosa que permite darle solución a disímiles aplicaciones logrando el grado de conectividad con otros sistemas deseados. La Figura II-II muestra el diagrama de bloques de este microcontrolador.

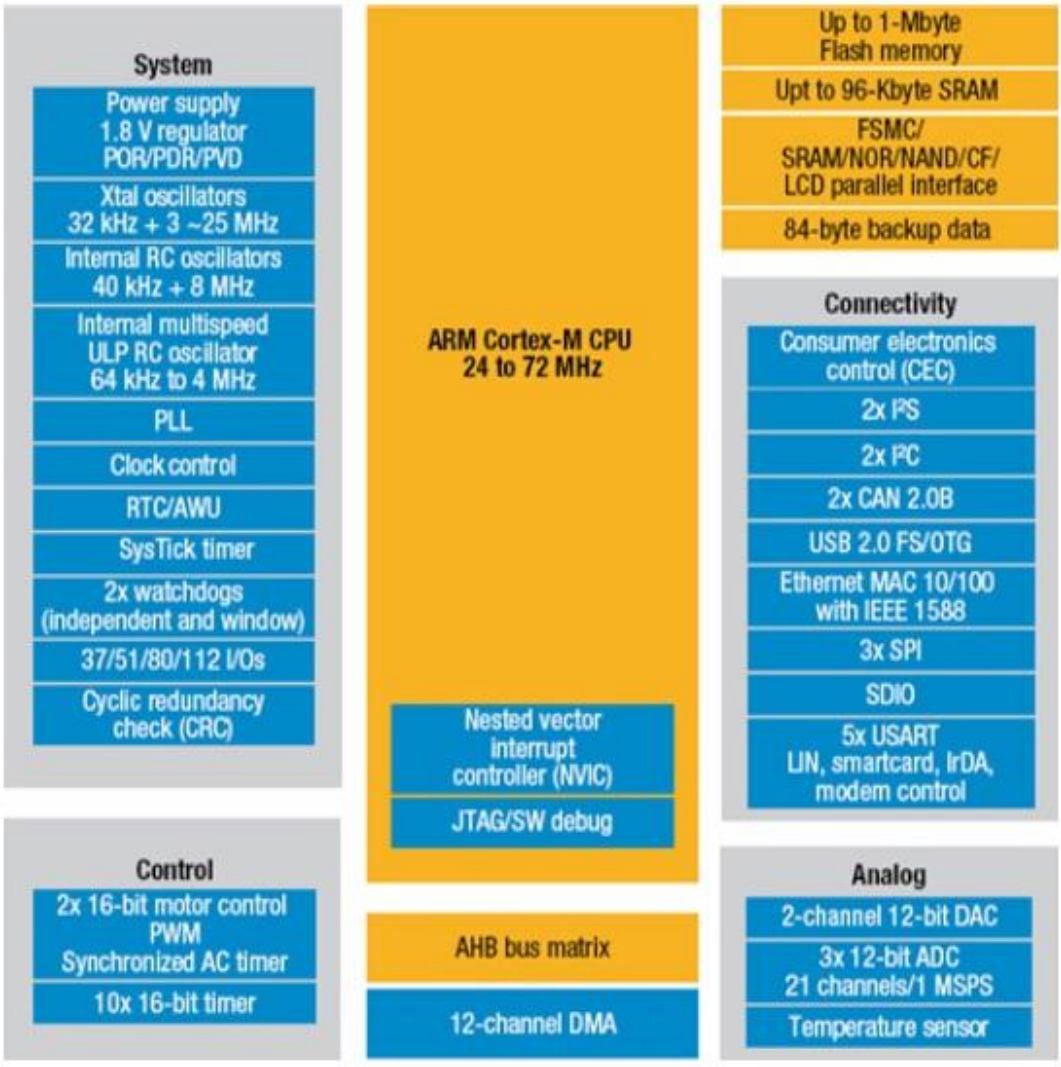


Figura II-II. Diagrama de bloques del microcontrolador.

2.2.1 Capa MAC del microcontrolador STM32F107VC.

El periférico *ethernet* habilita al micro STM32F107VC para transmitir y recibir datos a través *ethernet* de acuerdo con el estándar IEEE 802.3-2002. Proporciona un periférico configurable y flexible para satisfacer las necesidades de las distintas aplicaciones y clientes. Soporta dos interfaces estándar con la capa física externa (PHY), la definida por defecto en el estándar IEEE 802.3, MII (*Media-Independent Interface*) y la RMII (*Reduced Media-Independent Interface*). Se puede utilizar en un gran número de aplicaciones tales como *switch*, tarjetas de interfaz de red, etc. Es compatible con los siguientes estándares:

- IEEE 802.3-2002 para *ethernet* MAC.
- Estándar IEEE 1588-2002 para la sincronización de reloj de precisión en red.
- AMBA 2.0 para puertos AHB Maestro / Esclavo.
- Especificaciones RMII del consorcio RMII.

Dentro de las principales características y funcionalidades que permite este periférico se pueden encontrar las siguientes:

- Soporta tasas de transferencias de datos de 10/100 Mb a través de una interfaz física externa (PHY).
- Interfaz MII compatible con el estándar IEEE 802.3 para la comunicación con una capa física externa que soporte Fast *ethernet*.
- Soporta los modos de operación Half-Duplex y Full-Duplex.
- Soporte de Bak-pressure para la operación en modo Half-Duplex.
- Soporte del protocolo CSMA/CD en el modo de operación Half-Duplex.
- Soporte del estándar IEE 802.3x para el control de flujo en el modo de operación Full-Duplex.
- Generación automática de relleno y CRC.
- Opciones para la extracción automática del relleno o el CRC en las tramas de recepción.
- Tamaño de tramas programables para soportar las tramas estándar con tamaño superior a los 16 KB. Tiempo de espera entre tramas programables (de 40 a 96 bit en pasos de 8).
- Soporta una gran variedad de modos de dirección y filtrado.
- Retorno por separado de 32 bit de estado para la transmisión y la recepción de paquetes.
- Soporte del estándar IEEE 802.1Q para la detención de etiquetas VLAN en la recepción de tramas.
- Interfaces de transmisión, recepción y control separadas de la aplicación.
- Interfaz MDIO para la configuración y gestión del dispositivo de la capa física.
- Detección de tramas *wakeup* de redes LAN y de tramas de paquetes mágicos AMD.
- Capacidad de recepción mejorada para el chequeo de la suma de comprobación de las cabeceras y el encapsulado de la suma de comprobación en los datagramas de TCP, UDP o ICMP en las versiones IPv4 e IPv6.

- Descarta los marcos en colisión tardía, las colisiones excesivas, el aplazamiento excesivo y condiciones de empotramiento.
- Maneja retransmisión automática de las tramas de colisión para la transmisión.

La capa MAC es compatible con el estándar *IEEE 802.3*, y posee un controlador DMA (*Direct Memory Access*) dedicado. Soporta los estándares MII y RMII los que son seleccionados a través de un bit en el registro *AFIO-MAPR*. Las interfaces del controlador DMA con el núcleo y la memoria se hacen a través de las interfaces esclavo y amo del bus AHB. La interfaz amo controla la transferencia de datos, mientras que el esclavo controla los accesos a los registros de estado y control (CSR). La cola de transmisión (FIFO Tx) almacena los datos leídos de la memoria del sistema a través del canal DMA antes que se realice la transmisión por la capa MAC. De manera similar, la cola de recepción (RX FIFO) almacena las tramas recibidas desde la línea hasta que se transfieren a la memoria del sistema a través del canal DMA. El periférico *ethernet* también incluye una interfaz SMI (*Station Management Interface*) para comunicarse con la capa física externa PHY. Esta interfaz a través de un conjunto de registros de configuración permite al usuario seleccionar el modo deseado y las características de la MAC y el controlador de DMA. En la Figura II-III se puede apreciar el diagrama de bloques de este periférico.

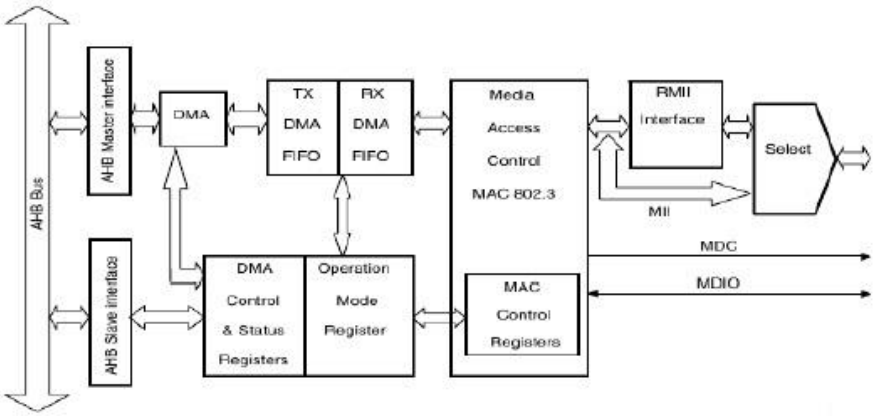


Figura II-III. Diagramas del periférico ethernet.

2.2.2 Transmisor Receptor Universal Sincrónico Asincrónico (USART).

El microcontrolador cuenta con 3 puertos USART y dos UART que permiten el intercambio de datos con equipos externos que requieran de un estándar de comunicación serie full–dúplex. Estos periféricos ofrecen un amplio rango de

velocidades de transmisión a través del uso de un generador de velocidad de transmisión fraccional. Soportan las transmisiones en los modos *full-dúplex* y *half-dúplex* y poseen el control de flujo por hardware (CTS/RTS). Dentro de sus principales características se encuentra:

- Comunicaciones asincrónicas en modo full dúplex.
- Generador de velocidad de transmisión fraccionario.
- Bit de datos programables (8 o 9 bits).
- Bit de paradas configurables (1 o 2 bits).
- Posee 10 fuentes de informaciones.

2.3 Librerías de soporte para el desarrollo con microcontrolador STM32.

Debido a la amplia gama de periféricos y funcionalidades con que cuentan estos microcontroladores, el fabricante STMicroeletronics brinda un Estándar de Interfaz de Software para los Microcontroladores Cortex (CMSIS por sus siglas en inglés) que posee una serie de bibliotecas de bajo nivel que forman la base de la estructura de un proyecto (STmicroelectrinics, 2012b). Dentro de estas librerías se encuentran:

- Capa de acceso a los periféricos del núcleo: Contiene la definición de nombres, direcciones, y funciones que ayudan a acceder a los registros de los periféricos. Además define una interfaz independiente para los núcleos de tiempo real que incluye los canales de traceo. Esta capa de software es expandida por los socios del proyecto Silicon con:
  - ✓ Capa de acceso a los periféricos del dispositivo: Provee las definiciones para todos los periféricos.
  - ✓ Funciones de acceso a los periféricos: Proveen funciones adicionales para el manejo de los periféricos.

CMSIS define para los sistemas de microcontroladores Cortex–M una forma común de acceso a los registros de los periféricos y la definición de vectores de excepción. También define los nombres de los registros de los periféricos del núcleo, así como los vectores de excepción del núcleo y brinda una interfaz independiente para los núcleos de tiempo real, incluyendo un canal de traceo. Usando estos componentes de software el usuario puede reutilizar el código además de que este estándar habilita que se

pueda usar la combinación de múltiples componentes de software (STmicroelectrinics, 2010). Los ficheros genéricos que se encuentran dentro del estándar son:

- device.h: Es el fichero central y es provisto por el vendedor, el mismo contiene:
  - ✓ La definición de los números de interrupciones: Provee el número de interrupción para todos los núcleos así como las excepciones y las interrupciones específicas de los dispositivos.
  - ✓ Configuración del núcleo Cortex-M3: Contiene la configuración actual del procesador Cortex-M que es parte del dispositivo y provee el acceso a los registros del procesador y al núcleo de los periféricos
  - ✓ Funciones de acceso a los periféricos: Son funciones de acceso que pueden ser incluidas como funciones en línea o a través de librerías externas de un dispositivo proporcionadas por el vendedor.
- startup_device.h: Este fichero es provisto por ARM para cada compilador soportado. Es adoptado por los vendedores para incluir los vectores de interrupción para todos los manipuladores de interrupción. Cada manipulador de interrupción es definido como una función.
  - system_device.h: Es provisto por ARM y adaptado por los vendedores para que machee con el dispositivo actual. Provee una función de configuración del sistema y variables globales que contienen la frecuencia de reloj.

Para el caso del microcontrolador stm32f10vc se deben incluir los siguientes ficheros:

- **stm32f10x.h:** Este fichero contiene las definiciones de los registros periféricos así como la definición de los bits y el mapeo de memoria STM32F10X para línea de dispositivos orientados a la conexión STM32F10x.
- **startup_stm32f10x.h:** Provee los vectores de interrupción para todos los manipuladores de interrupción.
- **system_stm32f10x.h:** Contiene la capa de acceso al dispositivo Cortex-M3.

Con estos paquetes incluidos, más los manejadores de los periféricos, de los cuales se va a hacer uso en el proyecto, se cuenta con una arquitectura como la mostrada en la Figura II-IV (STmicroelectrinics, 2012b).

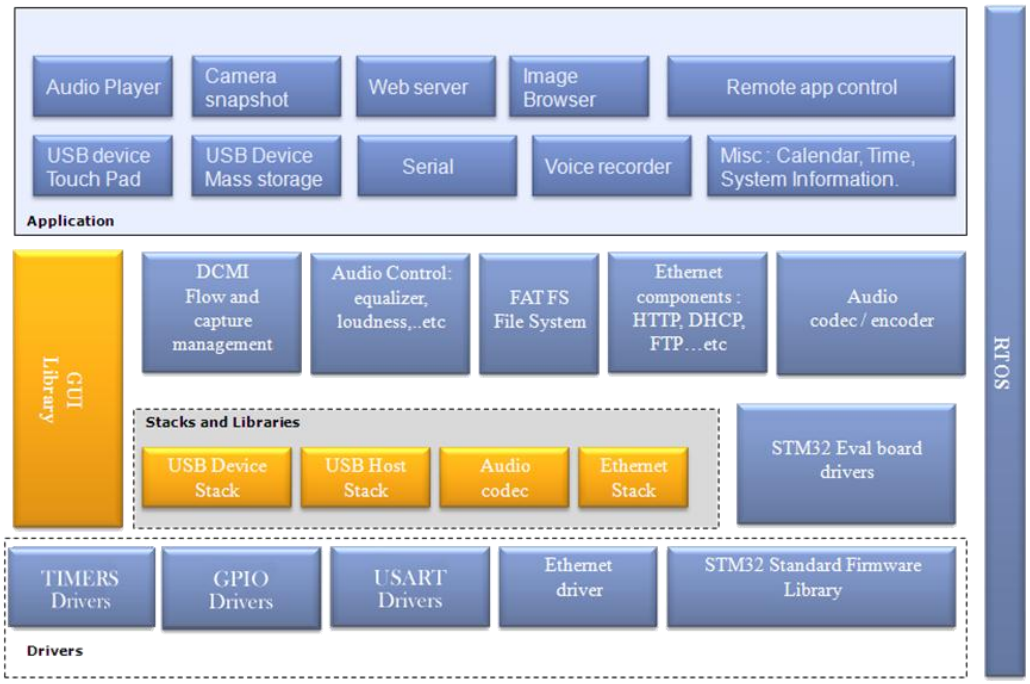


Figura II-IV. Diagramas del periférico ethernet.

Como se aprecia en la arquitectura es posible integrar en la misma el uso de un sistema operativo de tiempo real. A continuación se explica el porqué de la selección y las características de FreeRTOS.

2.4 FreeRTOS.

El fabricante del microcontrolador STM32F107VCT en el reporte (STmicroelectrinics, 2012a) brinda un detallado informe sobre las características de los *kernel* de tiempo real soportados por sus microcontroladores en la Tabla I-I.

Se puede observar que existen solo tres *kernels* libres, de ellos dos son soportados por la serie F1 a la cual pertenece el microcontrolador STM32F, en el caso de ChibiOS/RT se cuenta con una licencia de tipos GPL3 y del mismo no se dispone de una amplia documentación. En el caso del micro uCLinux no es soportado por la serie F1 puesto que necesita más recursos de memoria dinámica, quedando solo el caso de FreeRTOS que es soportado por la serie F1, es distribuido bajo una licencia GPL libre de costo, además de contar con una amplia documentación y soporte.

Tabla I-I. Kernel de tiempo Real para núcleo STM32.

Proveedor	Solución	Modelo	Costo	Disponibilidad						
				F0	F1	F2	F3	F4	L1	W
AVIX-RT	AVIX	Binaries	License	N	Y	Y	Y	Y	Y	N
CMX	CMX-RTX	Source	License	N	Y	Y	Y	Y	Y	N
CMX	CMX-Tiny	Source	License	Y	Y	Y	Y	Y	Y	N
Chibios	ChibiOS/RT	Open source (GPL3) or Source	Free or License	Y	Y	Y	Y	Y	Y	N
eCosCentric	eCosPro	Source[1]	License	N	Y	Y	Y	Y	Y	N
eForce	µC3	Source	License	Y	Y	Y	Y	Y	Y	N
Emcraft Systems	uCLinux	Open Source (GPL)[2]	Free[2]	N	N	Y	Y	Y	N	N
EUROS	EUROSPlus	Binaries	License	N	Y	Y	Y	Y	Y	N
Express Logic	ThreadX	Source	License	Y	Y	Y	Y	Y	Y	N
FreeRTOS	FreeRTOS	Open source (modified GPL)	Free	Y	Y	Y	Y	Y	Y	N
Green Hills	µ-velOSity	Source	License	Y	Y	Y	Y	Y	Y	N
Keil/ARM	MDK-ARM	Source	License	Y	Y	Y	Y	Y	Y	N

2.4.1 Características de FreeRTOS.

FreeRTOS está diseñado para determinar la respuesta a sucesos o eventos, de forma tal, que se respeten los rangos de tiempos de respuesta, los cuales, en el caso de ser críticos para el sistema, requieren de atención inmediata. Es un sistema determinista, en consecuencia permite que el usuario prediga los tiempos de respuesta que FreeRTOS proporcionará a los eventos que ocurran en el sistema informático que administra. Estos eventos, que normalmente se producen en el entorno del sistema; llegan como interrupciones al procesador, ya sea por software o hardware. FreeRTOS es capaz de atender ráfagas de interrupciones que activan diferentes procesos. Para no perder ningún suceso recibido, FreeRTOS debe trabajar con la técnica multiproceso, la cual ejecuta los procesos independientes unos de otros. Además de que permite estrategias para asignar prioridad a los procesos y ejecutarlos según su importancia. Esto se logra con una planificación basada en prioridades (Rubio, 2005).

Ha sido portado a 31 arquitecturas de hardware desde pequeños micro controladores de 8 byte a procesadores completos de 32 byte, incluyendo ARM7, ARM9, Cortex-M3, MSP430, AVR y PIC. Su versatilidad se debe a varios factores:

- El código base es pequeño. Está compuesto por un total de tres archivos para el núcleo y un archivo adicional llamado “puerto” que brinda la compatibilidad entre el dispositivo y el *kernel*.

- Está en su mayoría escrito en C estándar. Solo unas pocas líneas de código ensamblador son necesarias para adaptarlo a una plataforma en particular.
- Posee una extensa documentación en el código fuente así como en el sitio web oficial, con *benchmark* a nivel de aplicación.
- Es de código abierto. Está licenciado bajo GPL modificada y puede ser usado en aplicaciones comerciales bajo la misma. El código está disponible de manera gratuita en su sitio web, lo que hace al *kernel* fácil de estudiar y entender.

2.4.2 Tareas.

El sistema operativo maneja el concepto de tareas que son implementadas como funciones C. En cada momento solo una tarea podrá ser ejecutada por el procesador, esto implica que las tareas pueden estar básicamente en dos estados existentes: corriendo o detenida.

El estado detenido contiene una serie de sub-estados, ver Figura II-V, que serán explicados más adelante. Cuando la tarea se encuentra corriendo el procesador está procesando realmente su código, no siendo así cuando se encuentra detenida pues pasa a estar inactiva salvando su contexto para que sea recuperado una vez que el planificador decida que esta debe pasar a estado corriendo. El planificador FreeRTOS es la única entidad que puede cambiar el estado de las tareas.

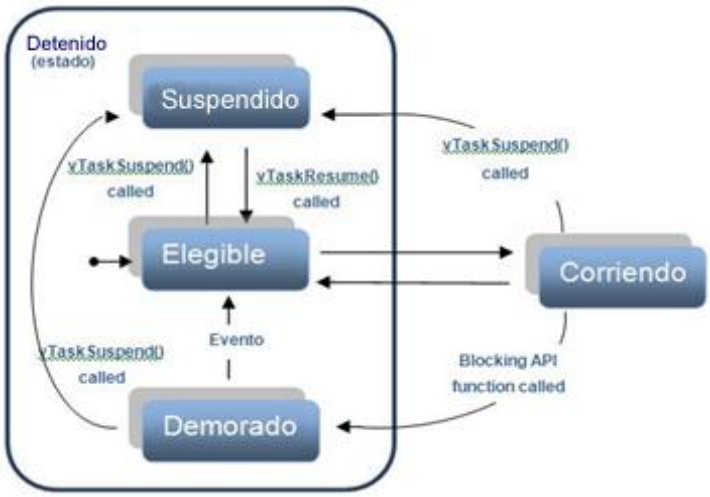


Figura II-V. Diagrama del estado de tareas.

Como se observa en la Figura II-V las tareas que se ejecutan por el *kernel* de FreeRTOS pueden transitar por varios estados (Barry., 2009) siendo estos los siguientes:

- **Demorado:** Se dice que una tarea está en estado demorado cuando está a la espera de un evento. Estos eventos pueden ser de dos tipos: eventos de temporización o eventos de sincronización.
- **Suspendido:** En este estado la tarea no está disponible para el planificador. Una tarea solo puede entrar en este estado a través de una llamada a la función vTaskSuspend() y solo saldrá del mismo mediante una llamada a la función vTaskResume() o vTaskResumeFromISR().
- **Elegible:** Una tarea que no se encuentre ni en el estado Demorado ni en el estado Suspendido se dice que está en el estado Elegible puesto que la misma entrará en ejecución una vez que su prioridad lo permita.

2.4.3 Implementación de tareas.

Las tareas se crean utilizando la función *xTaskCreate*, tal como se observa en la Figura II-VI.

```
#include "FreeRTOS.h"
#include "task.h"

portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
                           const signed portCHAR * const pcName,
                           unsigned portSHORT usStackDepth,
                           void *pvParameters,
                           unsigned portBASE_TYPE uxPriority,
                           xTaskHandle *pvCreatedTask
                         );
```

Figura II-VI. Función xTaskCreate.

Dicha función se encuentra dentro de la API de FreeRTOS y a la misma se le pasan una serie de parámetros que se definen en la Tabla II-II.

Tabla II-II. Parámetros de la función xTaskCreate.

Parámetro	Descripción
pvTaskCode	Es un puntero a la función que realizara la tarea. Es justo el nombre de la tarea en cuestión.
pcName	Nombre descriptivo para la tarea. No es usado por FreeRTOS, solo es un elemento descriptivo que se utiliza para el traceo del programa pues es mucho más fácil de hacer esto a través de un nombre que con el manipulador que crea el sistema para la tarea
UsStackDepth	Cada tarea al ser creada por el sistema posee una serie de parámetros que indican su propio estado y que son asignados por el núcleo de FreeRTOS a la tarea creada. Estos parámetros siempre que la tarea cambie del estado corriendo al estado detenida, deben de ser guardados

	en una pila para poder retornar al estado inicial una vez que la tarea entre nuevamente en ejecución. Este valor le dice al núcleo que tan grande puede ser la pila asociada a esta tarea.	
pvParameters	Acepta el paso de parámetros de un puntero a *void*. El valor asignado al *pvParameters* cuando la tarea es creada es el valor pasado a la función.	
uxPriority	Define la prioridad en que se ejecutará la tarea. Las prioridades se pueden asignar desde 0, que es la prioridad más baja, hasta (configMAX_PRIORITIES -1) que es la más alta prioridad. El parámetro configMAX_PRIORITIES es definido en el fichero FreeRTOSConfig.h.	
pcCreatedTask	Puede ser utilizado para pasar un manejador a la tarea que está siendo creada. Dicho manejador puede ser utilizado para hacer referencias a las tareas sin utilizar las llamadas a la API, por ejemplo cambiar la prioridad de la tarea o eliminarla.	
Returned value	hay dos posibles valores devueltos:	
	pdTRUE: esto indica que la tarea se ha creado con éxito.	errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY: cuando ocurre un error a la hora de crear la tarea.

2.4.4 Colas.

Las aplicaciones que usa FreeRTOS están estructuradas como un conjunto de tareas independientes, cada tarea es efectivamente un mini programa en su propio contexto. Es probable que este conjunto de tareas autónomas tengan que comunicarse entre sí con el fin de que, en conjunto pueden proporcionar la funcionalidad de un sistema útil. Las colas son el primitivo subyacente utilizado por todo tipo de comunicación FreeRTOS y el mecanismo de sincronización. Estas pueden contener un número fijo de elementos.

Para crear una cola se utiliza la función *xQueuesCreates* como se muestra en la Figura II-VII.

```
#include "FreeRTOS.h"
#include "queue.h"

xQueueHandle xQueueCreate( unsigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize );
```

Figura II-VII. Función xQueuesCreates.

Esta función devuelve un xQueueHandle para hacer referencia a las colas que se crean. Se le definen una serie de parámetros tales como:

- **uxQueusLength:** El número máximo de elementos que la cola que se está creando puede contener a la vez.

- **uxItemSize:** El tamaño en bytes de cada elemento de datos que pueden ser almacenados en la cola.
- La función *xqueueSend* se utiliza para enviar datos a la parte posterior de la cola. Los parámetros que requiere son:
  - ✓ **xQueue:** Identificador de la cola a la que se están enviando los datos (escrito).
  - ✓ **pvItemToQueue:** Un puntero a los datos que se copiarán en la cola.
  - ✓ **xTicksToWait:** Cantidad máxima de tiempo que la tarea debe permanecer en el estado bloqueado para esperar a que el espacio esté disponible en la cola, la cola debe estar llena.

La función *xQueueReceive* se utiliza para recibir (leer) elementos de una cola. El material que se recibe se elimina de la cola. Los parámetros de esta función son:

- **xQueue:** Identificador de la cola de la que los datos están siendo recibidos (leídos).
- **pvBuffer:** Un puntero a la memoria en la que los datos recibidos se copiarán.
- **xTicksToWait:** Cantidad máxima de tiempo que la tarea debe permanecer en el estado bloqueado a esperar que los datos estén disponibles en la cola, la cola debe estar vacía.

La cola puede tener múltiples lectores por ello es posible que una sola cola tenga más de una tarea bloqueada en la espera de datos. Cuando este sea el caso sólo una tarea se desbloquea cuando se disponga de datos. La tarea que se desbloquea será siempre la más alta prioridad que se espera de datos.

Al igual que al leer de una cola, se puede especificar un tiempo para escribir en la cola. Este es el tiempo máximo que la tarea debe estar en el estado bloqueada en espera a que el espacio esté disponible en la cola.

2.4.5 Mecanismos para la sincronización.

FreeRTOS ofrece una serie de mecanismos de sincronización de tareas que permiten la utilización de recursos compartidos que pueden ser parte de una tarea y de una rutina de servicio de interrupción ISR. Estos mecanismos permiten mantener las ISR lo más pequeñas posible de forma tal que estas no ocupen el procesador por un tiempo relativamente largo (Barry., 2009). Estos mecanismos son:

- **Semáforos Binarios:** Permiten desbloquear una tarea cada vez que una interrupción ocurra de forma tal que la tarea quede sincronizada con el evento en específico.
- **Semáforos de conteo:** Este mecanismo es utilizado para sincronizar una tarea con un evento de interrupción que ocurra con una frecuencia muy elevada. A diferencia del semáforo binario que solo puede manejar un evento a la vez, este tipo de semáforo almacena el número de veces que se genera el evento, permitiendo que la ISR se ejecute por cada vez que se genere la interrupción.
- **Colas:** Las colas de FreeRTOS también pueden ser usadas para sincronizar eventos. A diferencia de los semáforos estas estructuras también permiten la comunicación de datos.

2.5 LwIP.

Como se observa en la Figura II-IV para el desarrollo del sistema se puede contar con bibliotecas para el manejo de la lógica de *ethernet*. El fabricante *STmicroelectronic* en el documento (STmicroelectrinics, 2012a) muestra una lista de las bibliotecas *TCP/IP* soportados por los microcontroladores de la familia STM32 la cual se muestra en la Tabla II-III.

Tabla II-III. Lista de bibliotecas TCP/IP compatibles con la familia STM32.

Proveedor	Solución	Modelo	Costo	Disponibilidad			
				F107	F2	F4	W
CMX	CMX-TCP/IP, CMX-MicroNet, CMX-INet	Source	License	Y	Y	Y	N
EUROS	TCP/IP stack	Binaries	License	Y	Y	Y	Y
Express Logic	NetX and NetX Duo IPv4/IPv6	Source	License	Y	Y	Y	N
eCosCentric	SecureSockets, SecureShell	Source	License	Y	Y	Y	N
eForce	µNet3	Source	License	Y	Y	Y	N
GreenHills	µ-velOSity TCP/IP v4/v6	Source	License	Y	Y	N[1]	N
HCC	MISRA HCC-TCP/IP v4/v6	Source	License	Y	Y	Y	N
Interniche	NicheLite	Source	Free	Y	Y	Y	N
Interniche	NicheStack	Source	License	Y	Y	Y	N
Interniche	embTCP v4/v6	Binaries	License	N	Y	Y	N
Keil/ARM	MDK-ARM TCPNET	Source	License	Y	Y	Y	N
SICS	LwIP	Open source (BSD)	Free	Y[2]	Y[2]	Y[2]	N
Mentor Embedded	Nucleus Network	Source	License	Y	Y	Y	N

De todos los que se encuentran listados solo dos son de libre distribución: NicheLite y LwIP. En el primer caso no existe la documentación necesaria de como portar esta biblioteca con el sistema operativo FreeRTOS. En el caso del proyecto de LwIP existe

una amplia documentación además de que es una biblioteca que su uso no depende del sistema operativo y que su programación está realizada en su totalidad en C estándar.

El diseño de esta biblioteca se enfocó en mantener el consumo de memoria bajo, al igual que el tamaño del código, con el fin de que se pudiera utilizar en sistemas embebidos, para lograr este objetivo el código de LwIP evita al máximo la copia de información entre buffers y utiliza en su lugar intercambio de apuntadores. Con el fin de hacer esta biblioteca portable para distintas arquitecturas, no se realizan llamadas a funciones del sistema operativo ni se usan estructuras de datos en la implementación. En cambio, cuando las funciones del sistema son necesarias se utiliza una capa que realiza la emulación del sistema operativo.

La capa de emulación proporciona una interfaz uniforme para los servicios del sistema, tales como temporizadores, sincronización de procesos y mecanismos de paso de mensajes. En principio, al portar LwIP a otros sistemas operativos, sólo es necesaria una implementación de la capa de emulación para ese sistema operativo en particular.

2.5.1 API LwIP.

La API LwIP fue diseñada a partir del conocimiento de la estructura interna de LwIP para lograr una mayor eficacia en el desarrollo de las aplicaciones. La implementación de la API se divide en dos partes, debido al modelo de proceso de TCP/IP. Una parte de la API se implementa como una biblioteca enlazada al programa de aplicación y la otra es implementada en el proceso TCP/IP en cuestión. Las dos partes se comunican mediante la comunicación entre procesos (IPC) que son los mecanismos previstos por la capa de emulación del sistema operativo. La implementación actual usa los siguientes tres mecanismos IPC:

- La memoria compartida.
- Paso de mensajes.
- Semáforos.

El principio general de diseño utilizado es dejar que se realice tanto trabajo como sea posible dentro del proceso de aplicación en lugar de en el proceso de TCP/IP. Este código puede ser compartido entre los procesos, incluso si las bibliotecas compartidas no están soportados por el sistema operativo, el código se almacena en la ROM. Esto constituye una ventaja en el desarrollo de sistemas embebidos pues los microcontroladores que se usan para este fin, usualmente llevan cantidades bastante grandes de ROM, no siendo así la capacidad de su memoria de datos.

El manejo de información se realiza mediante los buffers para los cuales existe una serie de funciones que permiten la gestión de memoria. Los buffers utilizan la referencia a memoria, en vez de memoria dinámica, también se utiliza la memoria compartida. Es por ello que para que esto funcione, tiene que ser posible compartir la referencia a memoria entre los procesos. Los sistemas operativos utilizados en sistemas embebidos para los que LwIP se destina por lo general no aplican ningún tipo de protección de la memoria, por lo que esto no será un problema. Las funciones que manejan las conexiones de red se implementan en la parte de la implementación de la API, que reside en el proceso de TCP/IP. A continuación se hace una referencia a las funciones de la API LwIP.

Esta biblioteca maneja dos tipos de datos que se utilizan para la API LwIP. Estos son:

- **Netbuf:** La abstracción de buffer de red.
- **Netconn:** La abstracción de una conexión de red.

Cada tipo de datos se representa como un puntero a una estructura C. El conocimiento de la estructura interna de la estructura no se debe utilizar en los programas de aplicación. En cambio, la API proporciona funciones para modificar y extraer campos necesarios.

2.5.2 Manejo de la memoria en la API LwIP.

Las estructuras netbufs son buffer que se utilizan para enviar y recibir datos. Internamente, un netbuf se asocia con un pbuf como se muestra en la Tabla II-IV. La memoria asignada es la memoria RAM que está destinada de forma explícita para la retención de datos de la red, mientras que la referencia a memoria, podría referirse a ambos (referencia a RAM o referencia a ROM). La referencia a la memoria es útil para enviar datos que no son modificados, como las páginas web estáticas o imágenes.

Los datos contenidos en un netbuf pueden ser fragmentados en bloques de tamaño diferentes. Esto significa que una aplicación debe estar preparada para aceptar datos fragmentados. El contenido de un netbufs que ha sido recibido de la red, también contiene la dirección IP y el número de puerto del host origen. Las funciones para la extracción de esos valores se relacionan en la Tabla II-IV.

Tabla II-IV. Funciones para el manejo de memoria.

Funciones	Sinopsis	Descripción
netbuf_new()	struct netbuf * netbuf new(void)	Asigna una estructura netbuf.
netbuf_delete()	void netbuf delete(struct netbuf *)	Elimina la estructura netbuf.
netbuf_alloc()	void * netbuf alloc(struct netbuf *buf, int size)	Asigna memoria de un tamaño especificado en byte al buffer.
netbuf_free()	int netbuf free(struct netbuf *buf)	Libera la memoria asignada.
netbuf_ref()	int netbuf ref(struct netbuf *buf, void *data, int size)	Asocia al buffer un puntero a la memoria externa con un tamaño.
netbuf_len()	int netbuf len(struct netbuf *buf)	Retorna la longitud total de los datos en el buffer.
netbuf_data()	int netbuf data(struct netbuf *buf, void **data, int *len)	Esta función se utiliza para obtener un puntero y la longitud de un bloque de datos en el netbuf.
netbuf_next()	int netbuf next(struct netbuf *buf)	Esta función actualiza el puntero interno del buffer para que apunte al siguiente fragmento.
netbuf_first()	void netbuf first(struct netbuf *buf)	Restablece el puntero del buffer para que apunte al primer fragmento.
netbuf_copy()	void netbuf copy(struct netbuf *buf, void *data, int len)	Copia todos los datos del buffer en la memoria apuntada por el puntero a memoria externa y devuelve la longitud.
netbuf_chain()	void netbuf chain(struct netbuf *head, struct netbuf *tail)	Intercambia la cabecera y la cola del buffer.
netbuf_fromaddr()	struct ip addr * netbuf fromaddr(struct netbuf *buf)	Retorna la dirección IP del cual se recibió los datos.
netbuf_fromport()	unsigned short netbuf fromport(struct netbuf *buf)	Retorna el número del Puerto TCP del cual se recibió la información.

2.5.3 Manejo de las conexiones en la API LwIP.

Las funciones que permiten el manejo de las conexiones están asociadas al tipo de dato *netconn*. Estas funciones permiten la creación, de una conexión ya sea del tipo TCP/IP o UDP, permiten crear conexiones de tipo servidor que esperan a que un cliente se conecte o de tipo cliente para el caso que se desee conectarse a un servidor. Además permiten realizar la recepción y envío de datos a través del objeto conexión creada, así como monitorear el estado de las conexiones. Estas funciones son las mostradas en la Tabla II-V.

Tabla II-V. Funciones de la API.

Funciones de la API	Sinopsis	Descripción
netconn_new()	struct netconn * netconn new(enum netconn type type)	Crea una nueva conexión.
netconn_delete()	void netconn delete(struct netconn *conn)	Elimina una conexión existente.
netconn_bind()	int netconn bind(struct netconn *conn, struct ip addr *addr, unsigned short port)	Enlaza una conexión a una dirección IP local y el puerto.
netconn_connect()	int netconn connect(struct netconn *conn, struct ip addr *remote addr, unsigned short remote port)	Se conecta a una dirección IP remota.
netconn_recv()	struct netbuf * netconn recv(struct netconn *conn)	Recibe datos de una conexión.
netconn_listen()	int netconn listen(struct netconn *conn)	Establece una conexión TCP en un modo de escucha.
netconn_accept()	struct netconn * netconn_accept(struct netconn *conn)	Acepta una conexión entrante en una conexión TCP de escucha.
netconn_write()	int netconn write(struct netconn *conn, void *data, int len, unsigned int flags)	Envía los datos de una conexión TCP conectado
netbuf_data()	int netbuf data(struct netbuf *buf, void **data, int *len)	Esta función se utiliza para obtener un puntero a la longitud de un bloque de datos en el buffer.
netconn_close()	int netconn close(struct netconn *conn)	Cierra la conexión

2.6 Lenguaje de Programación.

En 1978, Dennis Ritchie y Brian Kernighan presentaron el libro "*El lenguaje de programación C*" en el cual describían al lenguaje con la siguiente frase:

> *C es un lenguaje de programación de propósito general que ofrece la economía de expresión, el flujo de control moderno, estructuras de datos y un rico conjunto de operadores. C no es un lenguaje de muy alto nivel, ni uno grande, y no está especializado en cualquier área particular. Sin embargo, la ausencia de restricciones y su generalidad que lo hacen más conveniente y eficaz para muchas tareas que supuestamente otros lenguajes poderosos (Kernighan y M, 1991).*

C carece de muchas características que poseen los lenguajes de programación modernos. No es orientado a objetos, no cuenta con un apoyo firme en la tipificación y

asignación de tareas, no soporta el manejo de excepciones, la protección de punteros, la comprobación de los límites en las variables y los tipos de matriz dinámica de cadenas (Zurrell, 2000). A pesar de sus limitaciones, “C se ha convertido en el lenguaje de los programadores de sistemas empotrados”. El lenguaje de programación C tiene una larga lista de ventajas. Solo se mencionarán algunas que lo relacionan directamente con los sistemas empotrados (Barry., 2009):

- Es pequeño y bastante sencillo de aprender, los compiladores están disponibles para casi todos los procesadores en uso hoy en día, lo que garantiza portabilidad.
- Tiene la ventaja de ser independiente del procesador, lo que permite a los programadores concentrarse en algoritmos y aplicaciones en lugar de los detalles particulares de la arquitectura del procesador.
- Posibilita la producción de código compacto, eficaz para casi todos los procesadores.

Tal vez la mayor fortaleza de C, y lo que lo diferencia de lenguajes como Pascal y FORTRAN, es que es un lenguaje de no muy alto nivel, es más bien pequeño, sencillo y no está especializado en ningún tipo de aplicación. C proporciona a los programadores de sistemas empotrados un grado extraordinario de control directo de hardware sin sacrificar los beneficios de un lenguaje de alto nivel.

2.7 Entorno de Desarrollo integrado.

La plataforma Eclipse consiste en un Entorno de Desarrollo Integrado (IDE[17]), abierto y extensible. Un IDE es un programa compuesto por un conjunto de herramientas útiles para un desarrollador de software. Como elementos básicos, un IDE cuenta con en un editor de código, un compilador/intérprete y un depurador. Eclipse sirve como IDE Java y cuenta con numerosas herramientas de desarrollo de software. También da soporte a otros lenguajes de programación, como son C/C++, Cobol, Fortran, PHP o Python. A la plataforma base de Eclipse se le pueden añadir extensiones (plugins) para extender la funcionalidad. Permite la integración con las herramientas para compilar, tracear y descargar los programas en la memoria *flash* del dispositivo (Melgarejo, 2009).

[17] IDE: Entorno de Desarrollo Integrado (*Integrated Development Environment*).

Características generales:

- Muy completo.
- Configurable mediante plugins.
- Pensado para Java pero adaptable a otros lenguajes.
- Versiones preparadas para C/C++.
- Plugins para Subversión y Doxygen.

2.8 Conclusiones del Capítulo.

El componente de la investigación que se detalla en este capítulo posibilitó la selección de métodos, técnicas y herramientas que permitirán lograr un ambiente estable y satisfactorio para la implementación del sistema. A continuación se detalla la elección:

- Se decidió que la metodología XP guiará el proceso de desarrollo del *firmware,* debido a las facilidades que esta brinda para la gestión de sistemas embebidos.
- A partir del estado del arte expuesto acerca de los Sistemas Operativos Empotrados de Tiempo Real, se decidió emplear en la solución a **FreeRTOS**. La decisión se basa principalmente en que es de código abierto y por los escasos recursos de hardware que requiere, esto unido a que, como se mencionó anteriormente, es pequeño, simple y fácil de usar.
- En cuanto a lenguaje de programación la decisión fue utilizar **C** por su portabilidad y sus características de lenguaje de bajo nivel, además de ser el lenguaje de preferencia universal para el desarrollo de sistemas empotrados.
- Como IDE se seleccionó **Eclipse CDT** que, junto a otras herramientas, conforman un entorno de desarrollo basado en tecnologías libres y/o de código abierto.

Capítulo3 Discusión y resultado de la solución propuesta.

En este capítulo se exponen los elementos que forman parte del modelo de desarrollo como son: la propuesta del sistema, el modelo arquitectónico propuesto, una visión general del diseño y la implementación de la solución. Se detallan y explican los artefactos generados por la metodología XP. Se exponen los resultados de las pruebas realizadas para comprobar las funcionalidades obtenidas en cada iteración.

3.1. Propuesta del Sistema.

El sistema a desarrollar contará con 2 interfaces de comunicación, uno para la capa de transporte TCP y otro para la capa de transporte serie. La capa de transporte TCP estará encapsulada dentro de una tarea que será manejada por el sistema operativo FreeRTOS. Dicha tarea se le pasarán una serie de parámetros como son: la dirección IP del dispositivo, la máscara de la red, la puerta de enlace, el modo de la conexión (Cliente/Servidor) y la dirección IP del servidor en caso de ser cliente. Dicha tarea mediante las funciones de la API LwIP creará una conexión en el modo pasado por parámetros, una vez creada la conexión se estará un tiempo tratando de recibir datos de *ethernet* y otro tiempo tratando de escribir datos en *ethernet*. Esto se debe a que el objeto de conexión creado por LwIP es bloqueante lo que impide separar en dos tareas la función de escritura y lectura. Para el manejo de los datos se crearán dos colas de 512 byte que serán pasados por parámetros a la tarea, las mismas serán usadas por las funciones de escritura y lectura. Los datos recibidos por *ethernet* serán colocados en la cola de recepción mientras que los datos a transmitir se extraerán de la cola de transmisión. Esta tarea chequeará en todo momento los posibles errores que puedan suceder en la conexión eliminando la misma en caso de error y volviéndola a crear para recuperarse ante el fallo.

La capa de transporte serie será implementada mediante los drivers de bajo nivel suministrados por el fabricante. Para ello se cuenta con las funciones de envío y recepción de un carácter por el puerto serie, así como el envío y recepción de cadenas de caracteres por el puerto serie, dichas funciones permiten establecer un tiempo máximo para el cumplimiento de la acción, de no poder realizarse retornan un error. La tarea de lectura de la capa de transporte serie recibirá por parámetros la dirección de la

cola de recepción en la que en todo momento insertará los datos leídos por el puerto serie. Esta tarea recibe por parámetros la cola en la que ubicará los datos. Esta función estará escuchando en todo momento si llegan datos por el puerto serie, de obtenerlo lo coloca en la cola y vuelve a escuchar el puerto. La tarea de escritura recibe por parámetros la cola de transmisión y preguntará si hay datos en la misma. De ser cierto, se envían los mismos por el puerto serie y se vuelve a preguntar si quedan datos en la cola.

La lógica de conversión radica en enlazar la conexión TCP creada con el puerto serie, básicamente consiste en pasarle por parámetros a la tarea de *ethernet* las mismas colas que se le asignan a las tareas de escritura y lectura de serie, pero en orden inverso. De tal manera que la función de escritura por *ethernet* envíe los datos que coloca la tarea de lectura por puerto serie en la cola asignada y la función de lectura por *ethernet* coloque los datos en la cola de la cual hace uso la tarea de escritura por puerto serie.

Las colas son utilizadas por dos tareas a la misma vez por lo que deben ser protegidas de alguna forma. La solución a este problema la brinda el sistema operativo FreeRTOS que cuenta con una API para el manejo de las colas, las cuales pueden ser utilizadas para realizar la comunicación de datos entre tareas. El funcionamiento general de la aplicación es mostrado en el siguiente la Figura III-I.

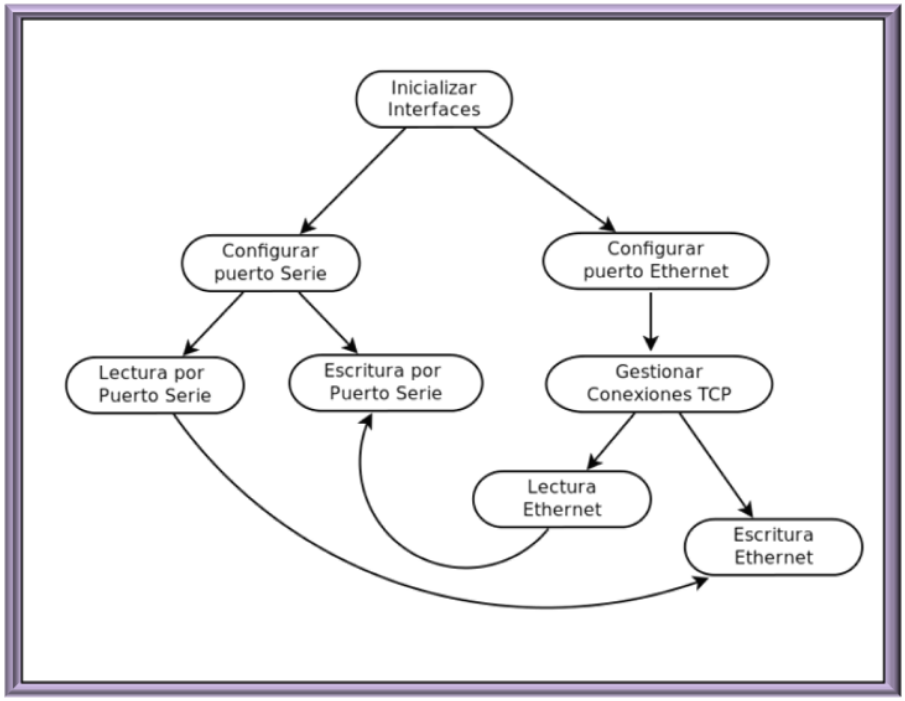


Figura III-I. Diagrama de Estado.

Ambas capas de transporte deberán obtener los parámetros para su configuración a través de una tarea que se encargará de inicializar las interfaces con los valores almacenados en la memoria de programa, a través de la aplicación de configuración.
En la Figura III-II se muestra el diagrama de despliegue de la solución propuesta.

Figura III-II. Propuesta del sistema.

3.2 Arquitectura del sistema.

Como arquitectura de la solución se propone la observada en la Figura III-III. Como se puede observar está compuesta por cuatro capas dentro de las que se encuentra la capa de manejadores, que posee la biblioteca estándar STM32, los manejadores para el puerto serie, *ethernet* y GPIO. Estas bibliotecas permiten el acceso a los periféricos así como las propiedades del núcleo del microcontrolador. Dentro de la capa de bibliotecas se encuentra la biblioteca LwIP que permitirá la gestión de la capa de transporte *ethernet* a través de las funciones que posee. En la capa superior se encontrarán las funcionalidades del protocolo TCP. Por último y haciendo uso de las capas antes mencionadas se encuentra la capa de aplicación en donde radica la lógica de conversión serie/*ethernet*, *ethernet*/serie. El enlace entre estas capas será gestionado por el planificador del sistema operativo FreeRTOS.

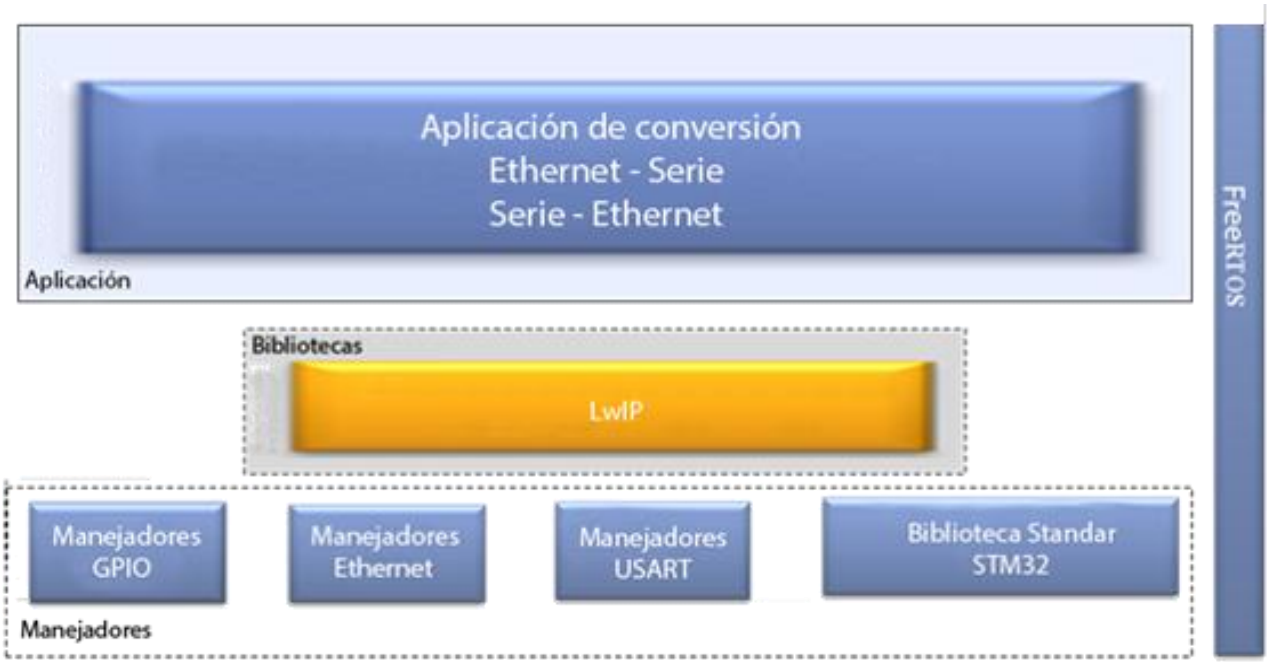


Figura III-III. Arquitectura de la aplicación de configuración.

3.3 Historias de usuarios y tareas.

Entre los artefactos que define la metodología seleccionada se encuentran las historias de usuario que presentan una breve descripción del comportamiento del sistema así como las tareas de cada una de estas historias.

Se redactaron historias de usuario que recogen las principales funcionalidades con las que debe contar el sistema a partir de las cuales se derivan un grupo de tareas de ingeniería que responden a los diferentes pasos que se deben ejecutar para compensar lo descrito en cada historia de usuario.

Tabla III-I. Historia de usuario Nº1.

Historia de Usuario	
Número: 1	**Nombre historia:** Implementación de la capa de transporte serie.
Usuario: Línea de Sistemas Empotrados.	**Iteración asignada:** 1
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto

Descripción: La capa de transporte serie debe permitir la transmisión y recepción de datos usando el mecanismo de las colas que brinda FreeRTOS para el flujo de información.
Observaciones: Se crearán dos tareas. Una para la recepción y otra para la transmisión de datos por cada puerto serie usado en el dispositivo.

Tabla III-II. Historia de usuario Nº2.

Historia de Usuario	
Número: 1	**Nombre historia:** Implementación de la capa de transporte *ethernet*.
Usuario: Línea de Sistemas Empotrados.	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Descripción: La capa de transporte *ethernet* debe permitir crear conexiones de tipo cliente o servidor según se desee, así como recibir y transmitir datos utilizando el mecanismo de las colas brindadas por FreeRTOS para el flujo de información.	
Observaciones: Se creará una tarea para la gestión de la capa de *ethernet*.	

Tabla III-III. Historia de usuario Nº3.

Historia de Usuario	
Número: 1	**Nombre historia:** Implementación de la lógica de conversión entre las interfaces series y *ethernet*.
Usuario: Línea de Sistemas Empotrados.	**Iteración asignada:** 3
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Descripción: La lógica de conversión debe permitir asociar una conexión TCP a un puerto serie determinado de forma tal que los datos recibidos por *ethernet* se envíen a través del puerto serie y viceversa.	
Observaciones: Se deben utilizar dos colas de FreeRTOS para la comunicación entre la tarea de *ethernet* y las tareas de escritura - lectura serie.	

Tabla III-IV. Historia de usuario Nº4.

Historia de Usuario	
Número: 1	**Nombre historia:** Integración de la lógica de conversión con la aplicación de configuración.
Usuario: Línea de Sistemas Empotrados.	**Iteración asignada:** 4
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Descripción: Se debe obtener los parámetros de configuración de las interfaces serie y *ethernet* a través de una tarea de configuración que permita inicializar las interfaces de *ethernet* y serie como lo desee el usuario.	
Observaciones: Se creará una tarea que permita inicializar las interfaces con los parámetros de configuración.	

3.4 Estimación de esfuerzo.

La siguiente tabla contiene la estimación de esfuerzo por historia de usuario (HU) según el orden a realizar. Cada estimación incluye el esfuerzo asociado a la implementación de la HU, la misma se expresa utilizando como medida una puntuación, que es directamente proporcional al esfuerzo. Un punto se considera como una semana ideal de trabajo, donde se trabaje el tiempo planeado sin ningún tipo de interrupción.

Tabla III-V. Estimación de esfuerzo por HU.

Historia de usuario	Puntos estimados
Implementación de la capa de Transporte serie.	2
Implementación de la capa de Transporte *ethernet*.	3
Implementación de la lógica de conversión entre las interfaces series y *ethernet*.	1
Integración de la lógica de conversión con la aplicación de configuración.	1

3.5 Plan de entregas.

El plan de entregas especifica exactamente que historias de usuario serán implementadas en cada entrega del sistema y sus prioridades, de modo que también permita conocer con exactitud qué historias de usuario serán implementadas en la

próxima liberación. Debe ser negociado y elaborado en forma conjunta entre el cliente y el equipo desarrollador durante las reuniones de planificación de entregas, la idea es hacer entregas frecuentes para obtener una mayor retroalimentación.

Tabla III-VI. Plan de entregas.

Plan de Entregas	
Conversor serie/*ethernet*	
Fecha de Reunión de Planificación:	20/01/13
Nombre de Documentador:	Aylín Rodríguez Reyo
Entrega Nº:	3
Historias de usuario a implementar en la Entrega	

No. HU	**Título**	**Prioridad**	**Fecha en la que entregará**	**Liberación en la que se incluirá**
1	Implementación de la capa de Transporte serie.	3	20/02/13	1
2	Implementación de la capa de Transporte *ethernet*.	3	25/03/13	2
3	Implementación de la lógica de conversión entre las interfaces series y *ethernet*.	3	15/04/13	3
4	Integración de la lógica de conversión con la aplicación de configuración.	3	30/04/13	4

3.6 Tareas de ingeniería.

Se redactaron un conjunto de tareas de ingeniería que constituyen los pasos a seguir para dar cumplimiento a las especificidades descritas por las historias de usuario, tal como se muestra a continuación.

Tabla III-VII. Tarea Nº1/ Historia de usuario Nº1.

Tarea	
Número tarea: 1	**Número historia:** 1
Nombre tarea: Implementación de la funcionalidad de escritura de la capa de Transporte serie.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 5-2-2013	**Fecha fin:** 10-2-2013
Descripción: Debe ser capaz de transmitir los datos existentes en la cola de transmisión a través del puerto serie.	

Tabla III-VIII. Tarea Nº2/ Historia de usuario Nº1.

Tarea	
Número tarea: 2	**Número historia:** 1
Nombre tarea: Implementación de la funcionalidad de lectura de la capa de transporte serie.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 10-2-2013	**Fecha fin:** 15-2-2013
Descripción: Debe ser capaz de recibir los datos del puerto serie colocándolos en la cola asignada para la recepción.	

Tabla III-IX. Tarea Nº1/ Historia de usuario Nº2.

Tarea	
Número tarea: 1	**Número historia:** 2
Nombre tarea: Implementación del modo de conexión cliente de la capa de *ethernet*.	

Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 5-3-2013	**Fecha fin:** 10-3-2013
Descripción: Debe ser capaz de crear una conexión de tipo cliente que se conecte a un servidor dado un ip y un puerto para la conexión.	

Tabla III-X. Tarea Nº2/ Historia de usuario Nº2.

	Tarea
Número tarea: 2	**Número historia:** 2
Nombre tarea: Implementación del modo de conexión servidor de la capa de *ethernet*.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 10-3-2013	**Fecha fin:** 15-3-2013
Descripción: Debe ser capaz de crear una conexión de tipo servidor que permita establecer la conexión de un cliente a través de un puerto TCP.	

Tabla III-XI. Tarea Nº3/ Historia de usuario Nº2.

	Tarea
Número tarea: 3	**Número historia:** 2
Nombre tarea: Implementación de las funcionalidades de lectura y escritura de la capa de transporte *ethernet*.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 15-3-2013	**Fecha fin:** 20-3-2013
Descripción: Debe ser capaz de recibir y enviar información a través de una conexión creada, ya sea en modo servidor o en modo cliente.	

Tabla III-XII. Tarea Nº1/ Historia de usuario Nº3.

Tarea	
Número tarea: 1	**Número historia:** 3
Nombre tarea: Implementación de la lógica de conversión entre las interfaces *ethernet* y serie.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 5-4-2013	**Fecha fin:** 10-4-2013
Descripción: Debe ser capaz de enviar a través de la conexión TCP creada los datos recibidos por el puerto serie asociado a dicha conexión y viceversa.	

Tabla III-XIII. Tarea Nº1/ Historia de usuario Nº4.

Tarea	
Número tarea: 1	**Número historia:** 4
Nombre tarea: Integración de la lógica de conversión con la aplicación de configuración.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 1
Fecha inicio: 20-4-2013	**Fecha fin:** 25-4-2013
Descripción: El dispositivo debe ser capaz de configurarse en el modo de trabajo obtenido mediante la aplicación de configuración.	

3.7 Requisitos no funcionales RNF.

Dentro de los requisitos no funcionales se encuentran: rendimiento, disponibilidad, costo, portabilidad entre otros. Para esta aplicación se determinó como requisitos no funcionales los mostrados en la Tabla III-XIV y Tabla III-XV.

Tabla III-XIV. Requisitos no funcionales de Portabilidad.

Historia de Usuario	
Número: 1	**Usuario**
Nombre Historia: Requisitos no funcional de Portabilidad.	
Descripción: El firmware debe empotrarse con una tarjeta basada en microcontrolador que posea como mínimo las siguientes prestaciones. Memoria RAM (64 Kb). Memoria de Programa (256Kb). Puertos USART (Para comunicación serial). 1 Capa MAC (Para la comunicación *ethernet*).	

Tabla III-XV. Requisitos no funcionales de Escalabilidad.

Historia de Usuario	
Número: 1	**Usuario**
Nombre Historia: Requisitos no funcional de Escalabilidad.	
Descripción: El *firmware* debe permitir que se le agreguen nuevas funcionalidades de configuración al sistema de una forma simple sin tener que realizar grandes cambios en el diseño de la aplicación.	

3.8 Iteraciones.

El desarrollo del sistema se realizó en cuatro iteraciones en las que se realizó la implementación de las tareas de las historias de usuarios que contienen las funcionalidades de la aplicación. A continuación se exponen los resultados de las pruebas realizadas a los prototipos obtenidos en cada iteración:

3.8.1 Primera iteración.

En esta iteración se implementa la funcionalidad recogida en la Historia de Usuario Nº1. Esta implementación se realiza a través de las tareas planificadas para la HU en cuestión. Al finalizar la iteración se debe obtener el prototipo funcional de la capa de transporte serie que permita el flujo de información a través de esta interfaz.

Tabla III-XVI. Caso de prueba de aceptación 1.

Caso de prueba de aceptación 1
Historia de usuario: Implementación de la capa de transporte serie.
Nombre: Chequear el funcionamiento de las funciones de escritura y lectura de la capa de transporte serie.
Responsable: Aylín Rodríguez Reyo.
Descripción: Esta prueba se realiza con el objetivo de comprobar que las tareas de escritura y lectura de la capa de transporte serie funcionen correctamente.
Condiciones de ejecución: PC que contenga dos puertos series con una aplicación serial instalada.
Entradas/pasos de ejecución: Puertos series uno y dos del dispositivo conectados a los puertos series de la PC. Dos aplicaciones DOCKLIGHT corriendo en la PC.
Resultado esperado: Lo que se envía hacia el puerto uno del dispositivo es transmitido por el puerto dos y viceversa.
Evaluación de la prueba: Prueba satisfactoria.

3.8.2 Segunda iteración.

En esta iteración se implementa la funcionalidad recogida en la HU Nº2. Dicha implementación se realiza a través de la solución a las tareas planificadas para la HU. Al concluir la iteración se debe obtener un prototipo funcional de la capa de transporte *ethernet* que permita el envío y recepción de datos a través de conexiones en modo cliente y servidor. Los resultados de esta prueba de muestran en el Anexo I.

Tabla III-XVII. Caso de prueba de aceptación 2.

Caso de prueba de aceptación 2
Historia de usuario: Implementación de la capa de transporte *ethernet*.
Nombre: Comprobar funcionamiento de la capa de transporte TCP.
Responsable: Aylín Rodríguez Reyo.
Descripción: Esta prueba se realiza con el objetivo de comprobar que la capa de transporte TCP permite la creación de las conexiones en modo servidor o cliente, así como el envío y la recepción de datos.

Condiciones de ejecución: PC que contenga aplicaciones TCP cliente y TCP servidores conectados a la red. El dispositivo conectado a la misma subred que la PC. Puerto serie número cuatro del dispositivo conectado al puerto serie de una PC que contenga una aplicación serial.
Entradas/pasos de ejecución: Se ejecuta la aplicación TCP cliente o TCP servidor según sea el caso y se procede a conectarse al dispositivo.
Resultado esperado: Se obtiene en la aplicación serial los datos enviados a través de la aplicación TCP cliente o TCP servidor. Los datos enviados a través de la aplicación serial se obtienen en la aplicación TCP cliente o TCP servidor.
Evaluación de la prueba: Prueba satisfactoria.

3.8.3 Tercera iteración.

En esta iteración se implementa la funcionalidad recogida en la HU Nº3. Dicha implementación se realiza a través de la solución a las tareas planificadas para la HU. Al concluir la iteración se debe obtener el prototipo funcional de la lógica de conversión entre las interfaces *ethernet* y serie.

Tabla III-XVIII. Caso de prueba de aceptación 3.

Caso de prueba de aceptación 3
Historia de usuario: Implementación de la lógica de conversión entre las interfaces series y ***ethernet***.
Nombre: Comprobar que la lógica de conversión *ethernet* y serie funcionen correctamente.
Responsable: Aylín Rodríguez Reyo.
Descripción: Esta prueba se realiza con el objetivo de comprobar que los datos recibidos a través de la conexión TCP sean enviados por el puerto serie asociados a la conexión y que los datos recibidos por el puerto serie sean enviados a través de la conexión TCP.
Condiciones de ejecución: PC que contenga aplicaciones TCP cliente y TCP servidores conectados a la red. El dispositivo conectado a la misma subred que la PC. Puerto serie número dos del dispositivo conectado al puerto serie de una PC que contenga

una aplicación serial.
Entradas/pasos de ejecución: Dos PC con la aplicación Test Software instalada. Dispositivo conectado a la misma subred que las PC y puerto serie dos del dispositivo conectado a una PC. Envío de un archivo de datos desde una PC hacia la otra a través del dispositivo.
Resultado esperado: El archivo enviado es recibido correctamente.
Evaluación de la prueba: Prueba satisfactoria.

3.8.4 Cuarta iteración.

En esta iteración se implementa la funcionalidad recogida en la HU Nº4. Dicha implementación se realiza a través de la solución a las tareas planificadas para la HU. Al concluir la iteración se debe obtener un prototipo funcional de la lógica de conversión integrado con la aplicación de configuración. Con esta iteración culminó la fase de prueba, los resultados obtenidos en el modo de funcionamiento, TCP cliente, se encuentran en el Anexo 1 y los del modo TCP Servidor, en el Anexo 2.

Tabla III-XIX. Caso de prueba de aceptación 4.

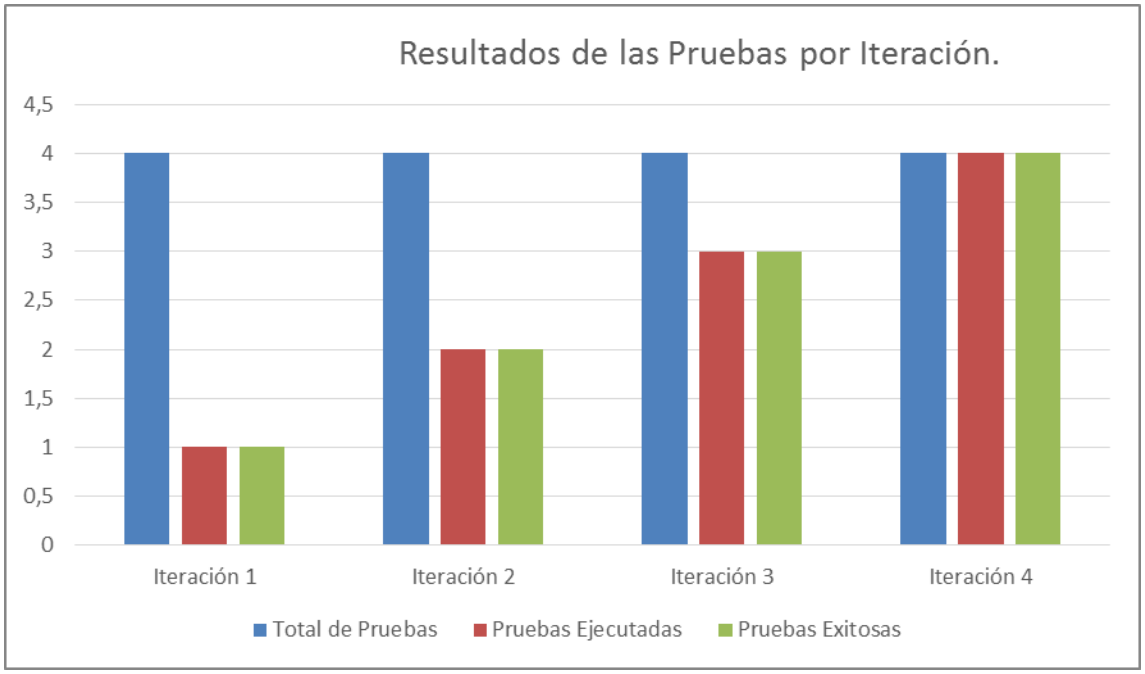
Caso de prueba de aceptación 4
Historia de usuario: Integración de la lógica de conversión con la aplicación de configuración.
Nombre: Comprobar la integración de la aplicación de configuración con el firmware.
Responsable: Aylín Rodríguez Reyo.
Descripción: Esta prueba se realiza con el objetivo de comprobar que el firmware es capaz de tomar los valores de configuración almacenados en la memoria flash y comportarse de la manera deseada por el cliente.
Condiciones de ejecución: PC que contenga un navegador web conectada a la red. El dispositivo conectado a la misma subred que la PC. Dos PC con la aplicación Test Software instalada. Dispositivo conectado a la misma subred que las PC y puerto serie dos del dispositivo conectado a una PC. Envío de un archivo de datos desde una PC hacia la otra a través del dispositivo.
Entradas/pasos de ejecución:

Se establecen varias configuraciones de modos de trabajos distintos en el dispositivo. Se comprueba con las aplicaciones Test Software el funcionamiento del dispositivo.
Resultado esperado: El dispositivo se comporta de manera requerida en la configuración.
Evaluación de la prueba: Prueba satisfactoria.

3.9 Resumen del resultado de las iteraciones.

Cada iteración concluyó con la realización de un conjunto de pruebas de aceptación para verificar el cumplimiento de los requisitos. Dichas pruebas permitieron al cliente comprobar el grado de satisfacción con el resultado. Las no conformidades detectadas fueron mitigadas antes de dar paso a una nueva iteración.

El gráfico que se muestra a continuación resume los datos arrojados por cada iteración:



3.10 Conclusiones Parciales.

En este capítulo fue definido el diseño de la solución que está basada en la arquitectura mostrada en la Figura III-III. La implementación de la solución se recoge en cuatro historias de usuario, para las que se identificaron un conjunto de tareas que generaron cuatro iteraciones en las que se obtuvieron prototipos funcionales que se fueron integrando hasta llegar a la solución final. El sistema operativo FreeRTOS permitió la generación de una tarea encargada de integración de la lógica de conversión con la

aplicación de configuración. El manejo de las conexiones se realizó a través de API LwIP. A lo largo del proceso de desarrollo y en cada iteración se realizó un conjunto de pruebas que permitieron comprobar el desempeño del sistema demostrando que se había cumplido con los requerimientos especificados por el cliente.

Conclusiones

En el presente trabajo se realizó el estudio de las aplicaciones de configuración brindadas por el fabricante Moxa en una de sus líneas de productos. Se determinaron las funcionalidades necesarias para el diseño de un conversor serie/*ethernet* en los modos de trabajo Cliente o Servidor TCP.

Se identificaron los parámetros de configuración necesarios para los modos de trabajo del conversor desarrollados en la línea de Sistemas Empotrados. Para el desarrollo de la aplicación se aplicó la metodología XP, que ha sido utilizada con éxito en proyectos de desarrollo de sistemas embebidos. Dicha metodología permitió definir junto al cliente los requisitos funcionales y no funcionales con los que debía cumplir la aplicación de configuración determinándose la arquitectura y las herramientas para dar solución al problema planteado. El desarrollo fue guiado a través de cuatro historias de usuarios que contienen un conjunto de tareas que tributaron a obtener prototipos funcionales en cada iteración que fueron probados y aceptados por el cliente, hasta obtener la solución final.

A través de las pruebas realizadas durante el proceso de desarrollo se validó la solución propuesta obteniéndose como resultado un conversor de datos para la línea de sistemas empotrados del CEDIN.

Recomendaciones:

Migrar el sistema a una versión más avanzada del sistema operativo FreeRTOS y a una versión más avanzada de la biblioteca LwIP.

Que se implemente alguna funcionalidad para manejar parámetros como los *time out* de conexión, el tiempo de escritura y lectura.

Bibliografía

AUGÉ, A. C. *Contribución al estudio de mejora de prestaciones del protocolo TCP en diferentes entornos*. Tesis para optar por el título de: Doctor en Ciencias de la Computación Universidad Politécnica de Catalunya, 2000.

BARRY., R. *FreeRTOS reference manual: API function an configuration options.* Editado. Edtion ed. Publicado en: Real Time Engineers Ltd., 2009. 121 p.

BRICEÑO, J. *Transmision de datos*. Editado. Edtion ed. Publicado en: Venezuela Taller de Publicaciones de la Facultad de Ingenieria., 2005.

CALCUTT, D., F. COWAN AND H. PARCHIZADEH *8051 Microcontroller an Applications based introduction.* Editado. Edtion ed. Publicado en: Oxford : Newnes, 2004. ISBN 0-7506-5759-6.

CRISP, J. *Introduction to Microprocessors and Microcontrollers*. Editado. Edtion ed. Publicado en: Oxford: Newnes, 2004. ISBN 0-7506-5989-0.

DROBKA, J., D. NOFTZ AND R. LAGHU *Piloting XP on Four Mission-Critical Projects*. IEEE Software, 2004, 21(6), 70-75.

GANSSLE, J. G. *eXtreme Embedded*. Embedded Systems Programming, 2001, 14(12), 75.

GRIDLING, G. AND B. WEISS*. Introduction to Microcontrollers*. Tesis para optar por el título de: Ing. En Informática Vienna University of Technology, 2007.

HASSAN, N. AND G. BRIGHT *A HYBRID RECONFIGURABLE COMPUTER-INTEGRATED MANUFACTURING CELL FOR THE PRODUCTION OF MASS CUSTOMISED PARTS*. South African Journal of Industrial Engineering, 2012, 23(1).

HEATH, S. *Embedded Systems Design*. Editado. Edtion ed. Publicado en: Oxford: Newnes, 2003. ISBN 0-7506-5546-1.

INFORMÁTICOS, S. *Dpto. de Sistemas Informáticos y Computación: UPV.*2013. Publicado por: DSIC. Accedido el 22 de enero del año 2013. Referenciado el 5 de febrero del 2013. Disponible en http://www.upv.es/entidades/DSIC/index.html.

INSTRUMENTS, N. *Comunicacion Serial: Conceptos Generales.*2013. Publicado por: National Intruments. Accedido el 10 de diciembre del año 2012. Referenciado el 15 de enero del 2013. Disponible en http://digital.ni.com/public.nsf/allkb.

IONEL NĂ, F. AND O. IVONA *CRITICAL ANALYSIS OF THE EXTREME PROGRAMMING (XP) PROJECT MANAGEMENT METHODOLOGY IN THE INFORMATION TECHNOLOGY FIELD*. Annals of the Stefan cel Mare University of Suceava : Fascicle of the Faculty of Economics and Public Administration, 2009, 9(Special Number), 14-14.

JASIO, L. D. *Programing 16 - Bit PIC Microcontrollers in C. Learning to Fly the PIC24.* Editado. Edtion ed. Publicado en: Elsevier, 2007. 401 p. ISBN 10:0-7506-8292-2.

JHONSON, R. G. *Capa de acceso a datos síncrona y asíncrona para modbus TCP*. Tesis para optar por el título de: Ing. En Ciencias Informáticas Universidad de las Ciencias Informaticas, 2008.

KERNIGHAN, B. W. AND D. M *El lenguaje de programación C*. Editado. Edtion ed. Publicado en: Pearson Educación, 1991. ISBN 9688802050.

KRISHNA, T. S. R., C. P. K. C. V, P. K. T AND V. V. KRISHNA *Survey on Extreme Programming in Software Engineering*. International Journal of Computer Trends and Technology, 2011, 2(2), 21-24.

MELGAREJO, Y. F. *Servicios de Integración con Terceros para el intercambio de Alarmas y Eventos que ocurran en el proceso y el sistema SCADA Guardián del ALBA*. Tesis para optar por el título de: Ing. En Ciencias Infomáticas Universidad de las Ciencias Informáticas 2009.

MOXA. *Nport 6000 Series User's Manual*. Publicado en: Estados Unidos de América, por Moxa Inc, 2011. Edición # 6

MOXA. *Moxa Products.*2013. Publicado por: Moxa Inc. Accedido el 2 de febrero del año 2013. Referenciado el 20 de febrero del 2013. Disponible en http://www.moxa.com/product/product.aspx.

PIERCE, D. *Extreme Programming Without Fear*. Embedded Systems Programming, 2004, 17(3), 30-35.

QING, L. *Real-time concepts for embedded systems*. Editado por: LAWRENCE. Edtion ed. Publicado en: CMP Books, 2003. ISBN 1-57820-124-1.

QUTAIBA, A. AND M. BASIL *Analysis and Design of a Guaranteed Real Time Performance Enhanced Industrial Ethernet*. International Arab Journal of e-Technology, 2011, 2(1), 18-28.

RESEARCH, E. O. F. N. *What is SCADA?*2000. Publicado por: Accedido el 15 de febrero del año 2013. Referenciado el 20 de marzo del 2013. Disponible en http://ref.cern.ch/CERN/CNL/2000/003/scada/.

RIDI, F. AND S. PAULUS INSAP *Improving Mobility in eXtreme Programming Methods through Computer Support Cooperative Work*. International Journal of Computer Science and Information Security, 2012, 10(2), 17-22.

RODRIGUEZ, M. *Tipos de Red y Protocolos de Comunicación.*2013. Publicado por: Ideatic. Accedido el 5 de febrero del año 2013. Referenciado el 15 de febrero del 2013. Disponible en http://www.estudioteca.net/universidad/telecomunicaciones.

RUBIO, D. M. F. *Arquitectura de Sistemas Multiagente para Micro controladores.* . Tesis para optar por el título de: Ing. En Electrónica. Pontifica Universidad Javeriana, 2005.

STALLINGS, W. *Local and metropolitan area networks (5th ed.)*. Editado. Edtion ed. Publicado en: Prentice-Hall, Inc., 2000. 605 p. ISBN 0-13-190737-9.

STMICROELECTRINICS. *Release Notes for STM32F10X Standard Peripherals Library (StdPeriph_Lib).*2010. Publicado por: STmicroelectrinics Inc. Accedido el 10 de enero del año 2013. Referenciado el 20 de febrero del 2013. Disponible en http://www.st.com.

STMICROELECTRINICS. *Embedded software solutions:STM32, STM8*. Publicado en: Estados Unidos de América, por STmicroelectrinics Inc., 2012a. Edición # 1.

STMICROELECTRINICS. *User Manual STM32 Demostration Builder developer guide.* Publicado en: Estados Unidos de América, por STmicroelectrinics Inc., 2012b. Edición # 1.

VELÁZQUEZ, E. C. AND C. F. S. GUERRERO. *Comunicacion entre microcontroladores bajo el protocolo zigbee.* Tesis para optar por el título de: Ing. En Electrónica Instituto Politécnico Nacional, 2009.

VERHAPPEN, I. AND E. BYRES *Industrial network integrity*. Intech, 2006, 53(10), 38-46.

WILMSHURT, T. *Designing Embedded Systems with PIC Microcontrollers. Principles and applications.* Editado. Edtion ed. Publicado en: Elsevier, 2007. 583 p. ISBN 10:0-7506-6755-9

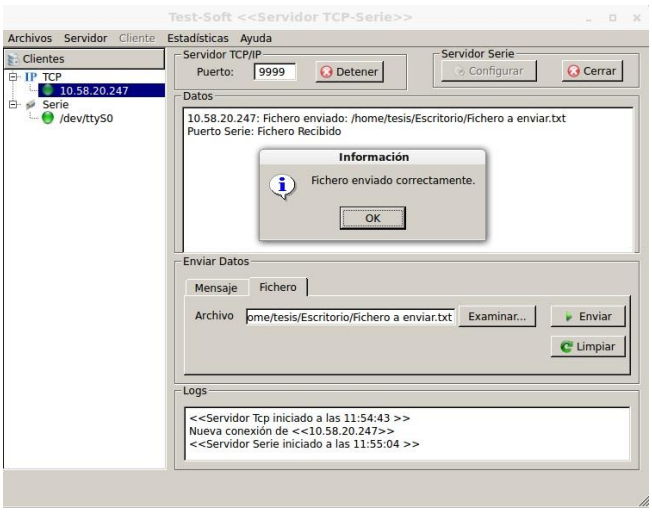
ZAMARRÓN, D. L. *Análisis de Sistemas Operativos de Tiempo Real Libres*. Publicado en: España, por Universidad Politécnica de Madrid, 2004. Edición # 1.

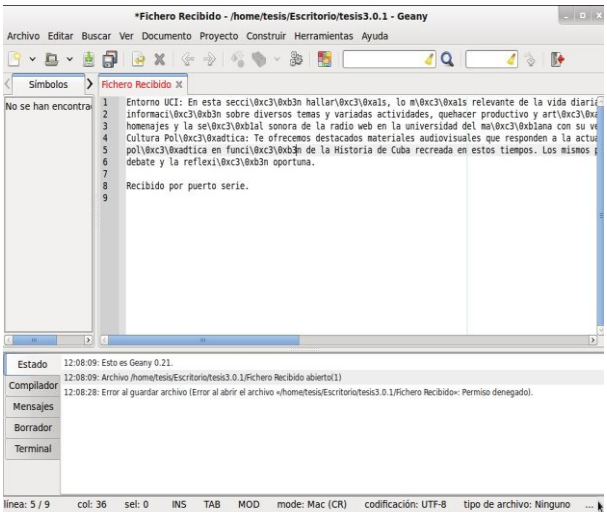
ZURRELL, K. *C Programming for Embedded Systems. Lawrence: R&D Books.* . Editado. Edtion ed. Publicado en: CMP Books, 2000. 173 p. ISBN 1929629044.

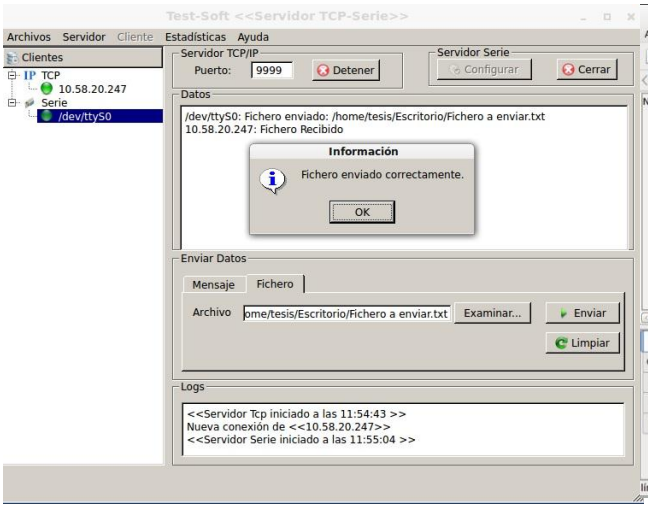
Anexos

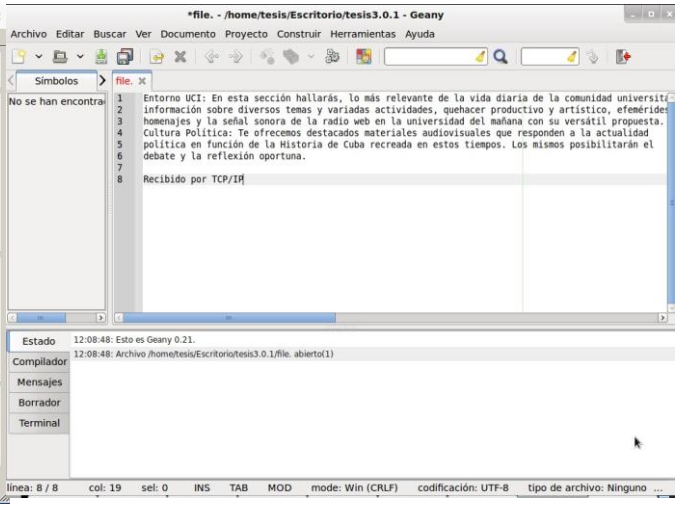
Resultado de las pruebas:

Anexo 1. Modo TCP Cliente









Anexo 2. Modo TCP Servidor