Sistemas embebidos en tiempo real

PROYECTO

INFORME FINAL

Nombre	CI	email
González Juan M	4469746-7	juan4469@gmail.com
Tassano Matias	4545764-4	tassoyro@gmail.com
Biagioni Pablo	4429899-6	biagioni84@hotmail.com

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de la República

Resumen

El presente proyecto trata del diseño y desarrollo de un sistema que genera y procesa un pedido de fecha y hora a un servidor remoto que maneja el protocolo Simple Network Time Protocol (SNTP). Este pedido es iniciado por medio de una terminal serie y es transferido al servidor mediante la interconexión a una LAN Ethernet. Para lograr generar un paquete de pedido de fecha y hora al servidor remoto (paquete SNTP client) se deben utilizar diferentes protocolos de Internet, los cuales se debieron desarrollar o adaptar y posteriormente integrar en el sistema. Luego de recibida la respuesta por parte del servidor, esta debe procesarse mediante ciertos algoritmos para poder lograr la sincronización temporal.

El sistema se diseñó en base al procesador ATmega32 de la familia AVR 8 bits(RISC) de ATMEL. Para poder lograr la conexión a la red LAN fue necesario integrar un chip que resolviese la transmisión física de los datos por medio del cable Ethernet. Para este proyecto se eligió el integrado ENC28J60 de Microchip. Este se comunica con el MCU por medio de la interfaz SPI. Con respecto a la comunicación serie necesaria para el inicio del proceso de sincronización se utilizó un conversor de niveles MAX232 de Maxim.

Este proyecto se desarrolló para la asignatura Sistemas Embebidos en Tiempo Real dictado en la Facultad de Ingeniería de la Universidad de la República y está enmarcado en el proyecto de fin de carrera HAOPL llevado a cabo por los mismos participantes.

Para la compresión completa de este informe son necesarios conocimientos previos sobre redes de datos.

Palabras claves : Ethernet, NTP, SNTP, ATmega32, enc28j60, UDP, ARP, IP, ICMP, Round Robin

Tabla de contenidos

Introducción

El problema a solucionar el cual motivó el desarrollo de todo este proyecto fue la necesidad de poseer fecha y hora actualizadas en un sistema embebido que funcionará como webserver y agendará tareas indicadas por el usuario a realizar en determinado momento. Al estar el sistema integrado en una Local Area Network (LAN) se buscó la solución entre los protocolos de Internet ya establecidos y se encontró como candidato más fuerte al protocolo Simple Time Network Protocol (SNTP).

SNTP y demás protocolos en ATmega32:

El protocolo de Internet utilizado para obtener la fecha y hora fue el SNTP, el cual surge como una simplificación del Network Time Protocol (NTP). SNTP es menos preciso que NTP, pero necesita algoritmos menos complejos para la sincronización de tiempo. Estos protocolos utilizan la referencia UTC (Universal Time Coordinated, evolución de su antecesor Greenwich Mean Time, GMT).

A nivel de capas en el modelo TCP/IP, los anteriores protocolos son clientes de UDP, que a su vez es cliente de IP y este último de Ethernet. Para que el sistema pueda integrarse de forma eficiente a una red LAN, es también necesario el protocolo ARP. Es por eso que fue necesario desarrollar tanto los protocolos mencionados, así como una estructura de software que los vincule y ejecute de la forma más efectiva posible.
En la siguiente imagen puede verse un diagrama de los protocolos utilizados ordenados de forma descendiente según nivel de capa creciente.

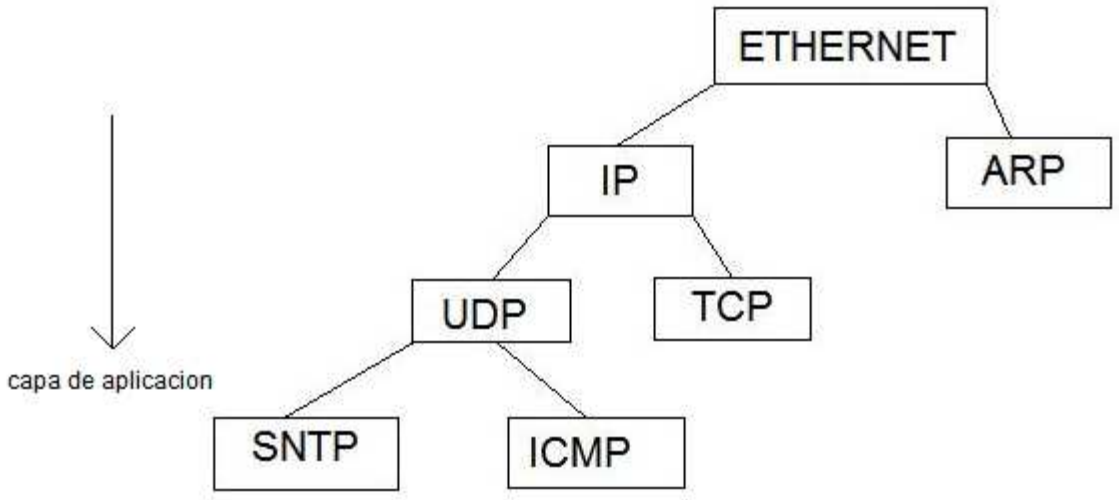


Figura 1. Protocolos usados, separados en capas.

SNTP utiliza el protocolo UDP para que este transporte sus paquetes por la red. El protocolo ICMP es necesario para poder realizar un *ping* hasta el servidor SNTP y descubrir si está activo. ARP averigua la correspondencia entre direcciones de capa de de red y capa MAC, y es fundamental ya que lo único que el sistema conoce a la hora de la inicialización es la dirección de capa de red del servidor NTP.

Antecedentes:

El proyecto esta basado en un proyecto de AVR portal llamado AVR net (http://www.avrportal.com/). En este se diseña una placa que implementa un webserver embebido para enviar datos capturados por el ADC de un ATmega32 a través de Internet. Se tomó como base para nuestro proyecto tanto la parte de hardware como de software, que fueron modificados para adaptarse a nuestras necesidades. Con respecto a la parte de hardware, se mejoró la comunicación entre el ENC28J60 y el ATmega introduciendo un adaptador de niveles de acuerdo a lo sugerido en la datasheet del ENC. Del software, se utilizaron varias funciones implementadas en el stack TCP/IP como punto de partida, y sobre todo las definiciones de constantes de los campos de los paquetes y valores típicos para los diferentes protocolos.

Objetivos

Para una realización satisfactoria del proyecto se pretendía lograr:

- Funcionamiento adecuado de la interfaz USART integrada en el ATmega32 para lograr la comunicación serie. Para lograr esto se utilizó parte del código desarrollado en los laboratorios de la asignatura.
- Adquirir el integrado ENC28J60 el cual no se encuentra en plaza y luego de esto poder comunicarse con él desde el ATmega32 mediante interfaz SPI. Los chips fueron importados desde USA.
- Integrar las implementaciones de los protocolos a utilizar conjuntamente con una plataforma de software que utilice estos de forma eficiente y que realice los procesos necesarios para generar las peticiones para luego procesarlas y almacenarlas como corresponde.
- Utilizar la menor cantidad de recursos de memoria del microcontrolador utilizado.

Alcance

Este punto fue evolucionando y reacomodándose en el transcurso del proyecto. Debido a estimaciones muy optimistas de carga de trabajo de las tareas a desarrollar, inicialmente se propuso que el proyecto tuviese un espectro mayor de alcance. Según transcurría el desarrollo del mismo, se acotó y enclareció este aspecto. Un factor determinante a la hora de tomar esta decisión fue el hecho de que la importación requirió más tiempo del esperado, retrasando el resto del desarrollo.

La situación a la cual el proyecto derivó puede resumirse con los siguientes puntos:

- El manejo de la interfaz serie es el necesario. No se realizan chequeos de errores, salvo por ciertas situaciones. El programa responde a mensajes predefinidos y es lo que se espera recibir. La posibilidad inicial de remplazar la interfaz serie por una interfaz USB quedó descartada.
- El hardware utilizado fue diseñado específicamente como prototipo en protoboard. No se pretendió construir una placa impresa que integrase todo el sistema.
- Tanto el stack TCP/IP como el driver de manejo del integrado ENC28J60 fueron adaptados de código existente de antemano y no se buscó una realización propia de estos. En cuanto a los protocolos a adaptar y probar en este proyecto, se limitó sólo a los necesarios para lograr los objetivos.
- La estructura principal del software se resume básicamente a un Round Robin con interrupciones, que fue una de las estudiadas en el curso. Se excluyó la utilización de un RTOS para el control de las tareas por dos razones: 1. la cota apretada de recursos de memoria del ATmega32, 2. la cantidad de tareas o procesos involucrados no parecían requerir un RTOS (eran pocas).

Diseño de hardware

Se armó en protoboard un sistema que posee como unidad central al microcontrolador ATmega32, al cual se le incorporó un cristal de 16MHz y para poder trabajar a esa frecuencia, que es requerida para lograr una correcta comunicación SPI con el chip Ethernet ENC28J60. Este chip trabaja con una tensión de 3.3Vdc, por lo que fue necesario agregar un conversor de niveles de tensión entre algunas señales. Se utilizó el buffer tri-estado 74ACT125 para esto. El chip Ethernet utiliza un cristal de 25MHz, que también fue importado.
Para la conexión física entre el chip ENC28J60 y el cable de Ethernet se utilizó un conector RJ45 modelo j00-0014NL de Pulse con filtros magnéticos anti EMI (Electromagnetic interference) incorporados.
Con respecto a la comunicación serie, se utilizó el conversor de niveles de tensión MAX232 de Maxim como había sido utilizado en los laboratorios de la asignatura.
Por último, para la programación y debugging del ATmega32 se utilizó la plataforma AVRDragon mediante interfaz JTAG, y a su vez alimenta al microcontrolador con tensión de 5Vdc.

El siguiente es un esquema de lo mencionado:

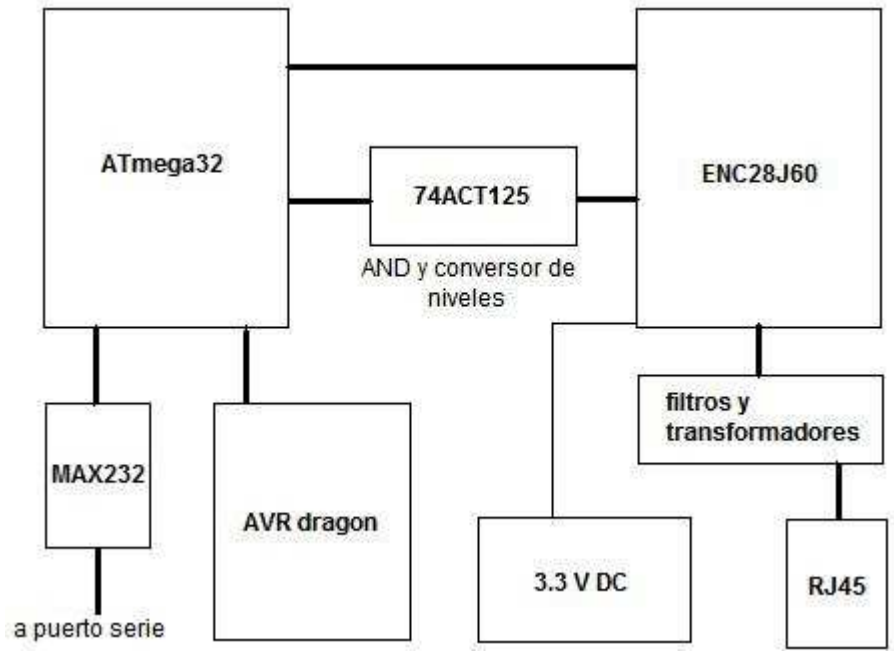


Figura 2. Diagrama del HW utilizado

Diseño de software

Se tomo como punto de partida para el diseño e implementación del software, el stack TCP/IP implementado en el proyecto AVRnet board de AVR Portal. Este se fue adaptando sucesivamente a las necesidades de nuestro proyecto.
En una primera instancia, dado que no contábamos con el HW necesario para realizar pruebas del stack base, se procedió a corroborar a mano contra las normas y datasheets tanto el funcionamiento del stack así como del driver para el ENC28J60. Pudimos ver entonces que si bien el stack estaba correctamente implementado, el mismo no satisfacía nuestras necesidades. En primer lugar, la implementación de UDP estaba hecha de una forma muy poco adaptable a nuestras necesidades y mezclaba funcionalidad propia del proyecto AVRnet con la implementación del protocolo en si. Por otra parte, el stack estaba diseñado y optimizado para utilizar round-robin con polling, cerrando la posibilidad de otras posibles implementaciones.
Lo que se hizo fue reconstruir el stack con el objetivo de generar bibliotecas de funciones que fueran lo mas independientes posibles de la implementación final para lo que se modificaron en menor medida las ya existentes (básicamente se separaron funciones que estaban integradas) y se hizo prácticamente desde cero lo referido a UDP.
De esta forma, se llego a una versión inicial conformada por una implementación y una interfaz para cada protocolo (ICMP.c, ICMP.h, IP.c, IP.h, etc.) que simplemente realizaban los chequeos y creación de paquetes correspondientes.
Dado que el sistema es en su mayoría reactivo (en el sentido de que los procesos se bloquean a la espera de eventos) y que el flujo de los mismos es predefinido e invariante (visto desde la perspectiva de comunicaciones exitosas), se decidió utilizar para el flujo del programa el mecanismo de round-robin con interrupciones, y diseñar los procesos como maquinas de estado.
Se puede ver más claramente lo anterior si analizamos el caso de una llamada ARP who is:
Primero se envía un paquete preguntando por un IP y se debe esperar la respuesta a dicho paquete. Esto implica dos estados posibles, uno activo en el que se envía el paquete, y uno en el que se espera la respuesta para procesarla.

La versión más simplificada de lo que hace el programa puede describirse así:

1. Desde una terminal en una PC se pide el tiempo.
2. Esto desencadena un ARP request.
3. Llega el ARP reply y se procesa.
4. Se hace un Echo request (función de ICMP).
5. Llega el Echo reply y se procesa.
6. Se hace un SNTP request.
7. Llega una respuesta SNTP y se procesa.
8. Luego de hacer una lectura desde una terminal se muestra el tiempo en pantalla.

Para implementar este modelo, se crearon funciones “wrappers” de los distintos protocolos, con la finalidad de dividir el flujo en la ejecución de diferentes tareas o procesos.
Se implementaron para estas, conjuntos de funciones estandarizadas de control de flujo (_start(), _init(), _error, etc.) y una estructura de banderas y campos de flujo para la intercomunicación entre procesos.

```
typedef struct _FLOW_FLAGS
{
	SNTP_FLOW SNTP;
	ARP_FLOW ARP;
	ICMP_FLOW ICMP;
	unsigned char buffer_empty:1;
} FLOW_FLAGS;
```

Como ejemplo, tomemos la siguiente:

```
typedef struct _ICMP_FLOW
{
	unsigned char on_process:1;
	unsigned char proc_step:2;
	unsigned char discover_ok:1;
	unsigned char requested_by:3;
	MAC_ADDR mac;
	IP_ADDR ip;
} ICMP_FLOW;
```

Corresponde a la estructura de banderas de flujo de ICMP. Dicha estructura es global al proyecto, y es utilizada por el loop principal para determinar si se debe ejecutar o no algún paso de la maquina de estados del proceso ICMP, y por las interrupciones y otros procesos para conocer datos como ser el estado, quien lo ejecuta, y si fue o no exitoso. Además, cada proceso tiene un campo de dirección MAC e IP.

Estructura del main

```
//inicializaciones

int main (void)
{

//mas inicializaciones

for(;;){

		if(FC.ARP.on_process) ARP_step();
		if(FC.ICMP.on_process) ICMP_step();
		if(FC.SNTP.on_process) SNTP_step();
		if (RxC_F) UART_step();

}

return 0;
}
```

En el loop principal, se chequea si los procesos están o no activos y en caso de estarlo se da un paso en la

maquina de estados. Los procesos se ponen inactivos en caso de estar a la espera de algún evento externo, y son despertados cuando este evento ocurre. El paso en que se encuentran es manejado por el handler del proceso, dentro de la función _step() o por otros procesos o interrupciones en caso de error a través de la función _error().

El main llama directa o indirectamente a las siguientes funciones:

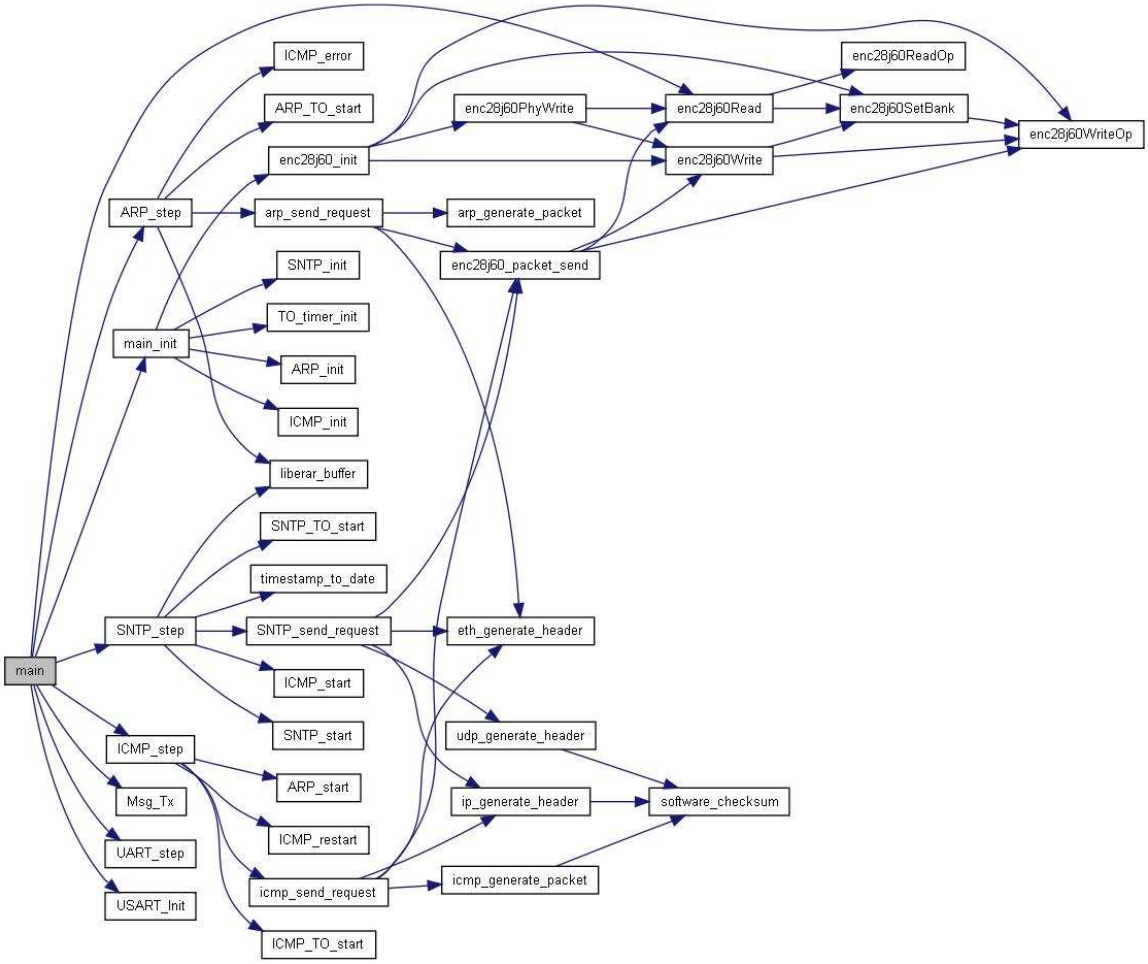


Figura 3. Diagrama de llamadas del main.

Dada la gran cantidad de archivos, interfaces publicas y funciones, se debió ser muy cuidadoso en la organización física del código. Para esto, se utiliza un archivo de inclusión global del proyecto (includes.h) el cual incluye las interfaces públicas de todos los protocolos y handlers. Por otra parte, dada la gran cantidad de estructuras utilizadas, las mismas están definidas en un único archivo (struct.h) para evitar diseminarlas por todo el código y hacer más fácil su modificación.
Un comentario aparte, es que los handlers de los protocolos también utilizan otros protocolos mediante las funciones *PROT_start* lo que justifica utilizar campos _requester para poder saber quien debe ser notificado en caso de éxito o error de un proceso.
Por ejemplo *SNTP_step* llama a *ICMP_start*, que inicia un proceso ICMP. A se vez ICMP_step llama a ARP_start que inicia un proceso ARP.

Flujo normal del programa.

Para mostrar el flujo del programa, usamos las siguientes convenciones:

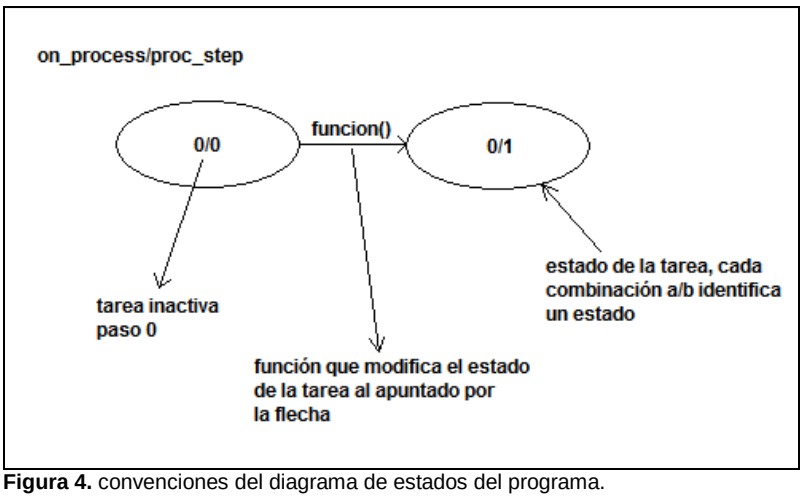


Figura 4. convenciones del diagrama de estados del programa.

El diagrama de estados del programa es el siguiente:

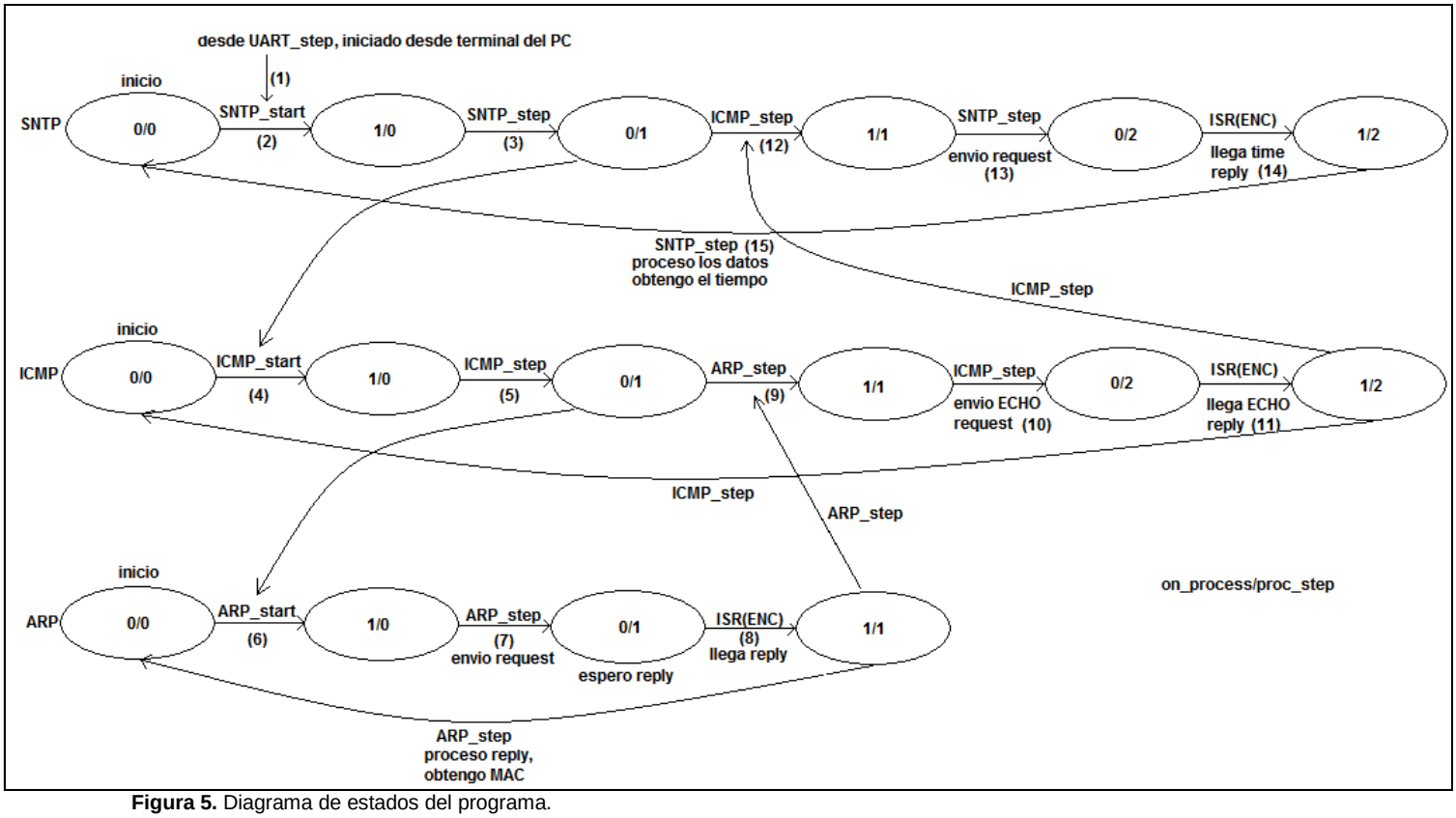


Figura 5. Diagrama de estados del programa.

Si miramos horizontalmente se aprecia la evolución de los estados de cada tarea asociada a un protocolo.

Si seguimos los números, se aprecia la secuencia de eventos que se seceden desde que se inicia el pedido de tiempo desde la USART hasta que se procesa el timestamp que devuelve el servidor SNTP. Como ya se vio, SNTP despierta a ICMP e ICMP despierta a ARP.

Manejo de la RAM

La principal restricción en el momento del diseño es que solo contamos con 2Kbytes de RAM. Dado que una trama Ethernet tiene un tamaño máximo de 1518 bytes, y debemos poder ser capaces de almacenarla en memoria para su procesamiento, se define una variable (buffer) de ese tamaño. Esto implica que sólo podremos procesar un paquete a la vez. Sin embargo, al contar con una cola FIFO implementada en RAM del ENC28J60, viendo el sistema completo tendremos una cola FIFO de unos 5.5Kbytes (4 del ENC y 1.5 del ATmega32). Esto disminuye la posibilidad de perder paquetes debido a la saturación de la cola.
La capacidad de dicha cola puede aumentarse hasta los 8Kbytes ya que podemos utilizar hasta 6.5Kbytes del espacio de memoria del ENC si disminuimos a 1.5Kbytes el espacio de memoria destinado a Tx del mismo. Esto no fue hecho así ya que, teniendo en cuenta una futura implementación de TCP, es útil tener más espacio de Tx para poder efectuar retrasmisiones en caso de ser necesarias.
Dado que solo quedan disponibles 530 bytes para variables auxiliares (de los 2 KB del ATmega) debimos tener especial cuidado en cuanto al manejo de memoria.
Entre las precauciones que se tomaron, las funciones fueron escritas utilizando para el pasaje de parámetros punteros ya que esto reduce el consumo de memoria.
Ej: icmp_send_request (buffer, (BYTE*)&FC.ICMP.mac, (BYTE*)&FC.ICMP.ip);
En algunos casos, como ser en la conversión de timestamp de SNTP a formato dd/mm/aa hh/mm/ss, se diseñaron estructuras de almacenamiento que minimizaran el consumo de memoria:

```
typedef struct {
        unsigned char anio : 6;
        unsigned char mes : 4;
        unsigned char dia : 5;
        unsigned char hora:5;
        unsigned char minutos:6;
        unsigned char segundos:6;
}date_time;
```

Además, los paquetes que se reciben y envían son procesados en la medida de lo posible dentro del buffer, es decir, se evita copiar datos del buffer a ubicaciones de procesamiento temporal para luego procesarlos. Dado que el buffer es utilizado por múltiples funciones y por la ISR de recepción de paquetes, es necesario protegerlo mediante el uso de una flag buffer_empty, que indica si esta o no disponible.

Recepción de paquetes

El ENC28J60 puede ser configurado para generar interrupciones si hay paquetes no leídos en el buffer de recepción por intermedio de una señal activa por nivel bajo y dicha señal se mantiene baja hasta que se vacíe la cola.
Por otra parte, es posible configurar el ATmega 32 para interrumpir por flancos (ascendente o descendente) o nivel (alto o bajo) si se utiliza la interrupción externa INT0 (no así si se utiliza INT2).
Se comenzó reconociendo la interrupción por flanco de bajada, lo cual nos daba la ventaja de interrumpir únicamente al principio de cada ráfaga de paquetes que pudiera llegar. Utilizada así, procesábamos el primer paquete de la ráfaga y cuando se liberaba el buffer, mirábamos si había paquetes pendientes (por medio de polling a EPKTCNT packet counter). De esta forma, evitábamos atender interrupciones de paquete pendiente cuando el buffer estaba ocupado.
Este método no funciono, en teoría porque si el flanco llegaba en medio de un bloque protegido, la misma no era reconocida. Por esto decidimos configurar INT0 por nivel bajo y, para evitar que se interrumpiera innecesariamente cuando el buffer estaba ocupado, deshabilitar la interrupción cuando se ocupaba el buffer y volver a habilitarla cuando se desocupaba. Esto, que surgió como una solución temporal hasta encontrar la causa del mal funcionamiento en modo flanco, resulto ser mucho mejor solución ya que el código se simplificaba notablemente, además de reducirse los tiempos de ejecución, por lo que se adopto como solución definitiva.

ISR de recepción de paquetes

El funcionamiento de la ISR se aparta del funcionamiento típico de una rutina de atención a interrupciones en el contexto de round-robin con interrupciones ya que el procesamiento “primario” que realiza es en muchos casos el único procesamiento que reciben los paquetes.
Cuando se desata una interrupción, se chequea que el buffer este vacío y en caso afirmativo se descarga el paquete a RAM y se realiza un procesamiento primario.
Se chequea el protocolo del paquete recibido, y en base a esto y a los estados (process_step) de los diferentes procesos se los procesa o descarta. Si un paquete es descartado (ya sea por ser de un protocolo, IP o puerto invalido o porque nadie lo esta esperando), se sale de la ISR marcando el buffer como libre y dejando la interrupción INT0 habilitada. Por otra parte, si el paquete es válido, puede requerir procesamiento inmediato y ser este el único procesamiento necesario (p.e un ICMP request necesita una respuesta inmediata) o necesitar almacenarse en el buffer para ser procesado posteriormente. En el primer caso, se procesa dentro de la ISR y se sale de la misma marcando el buffer como libre y dejando las interrupciones habilitadas. En el segundo caso, se despierta al proceso dueño del paquete y se sale de la ISR marcando el buffer como ocupado y deshabilitando la interrupción INT0.
Es por esto que todas las rutinas que necesiten escribir el buffer deben conocer si esta o no ocupado para lo que se definió en la estructura FLOW_FLAGS el campo buffer_empty.
Algo que debemos señalar es que chequear el la ISR si el buffer esta o no libre es innecesario ya que, como dijimos anteriormente, la solución adoptada de forma definitiva implica que INT0 solo estará habilitada cuando el buffer este libre.
La forma en que se chequea el protocolo en la ISR es la siguiente:

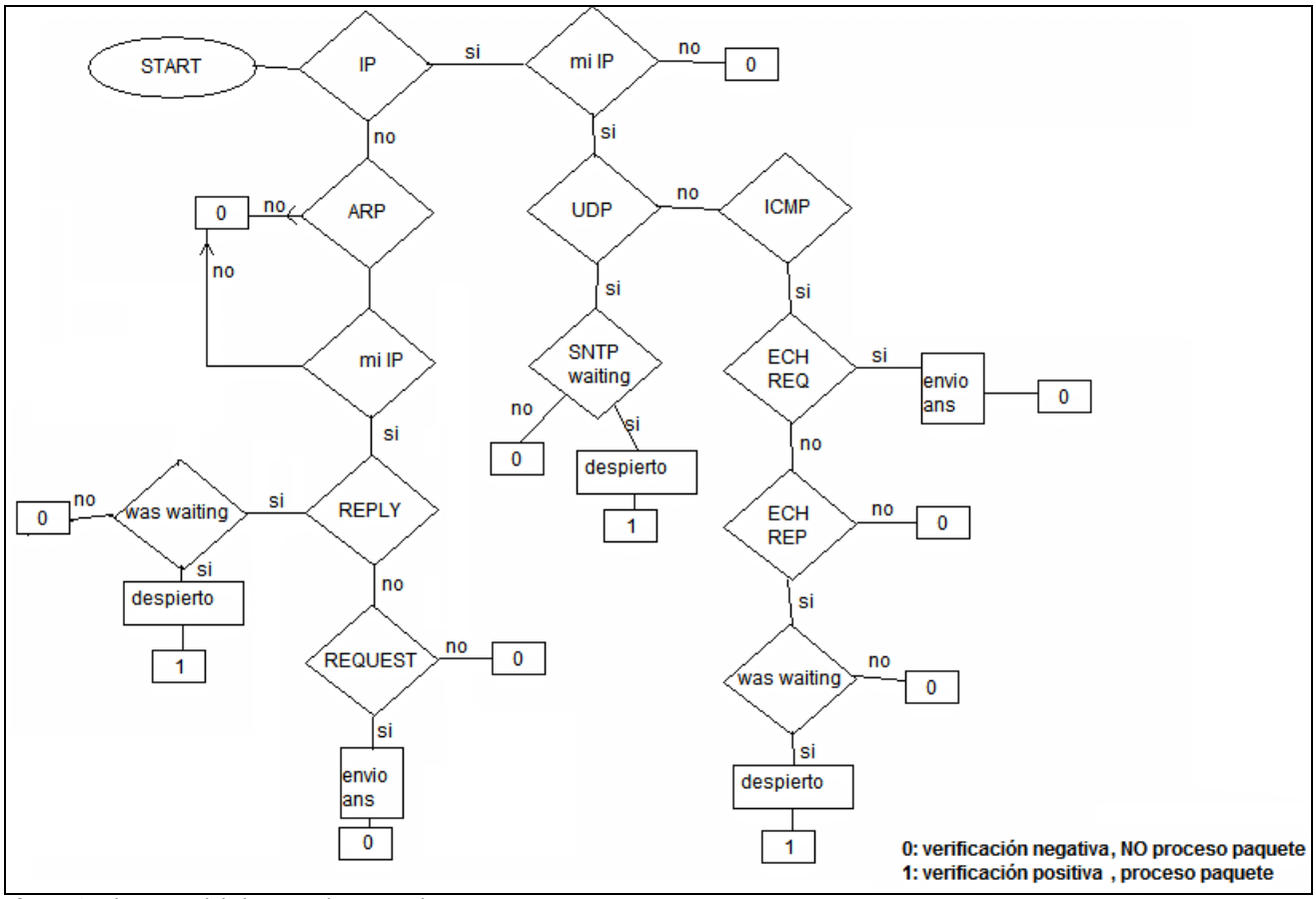


Figura 6. Diagrama del chequeo de protocolo

Time Out timers

Como nada garantiza que nuestros paquetes (o las respuestas a los mismos) no se pierdan en la red, se debió implementar un mecanismo de time out para evitar que los procesos se bloquearan si esto ocurría.
Por esto, se recurrió a la utilización de timers que llevaran una cuenta regresiva del tiempo de vida de los paquetes y que en caso de llegar a 0 llamaran a una rutina de atención a errores para el proceso dueño del paquete.
Dado que no puede configurarse un timer por protocolo, se utilizó un único timer (Timer2) configurado para interrumpir cada 5 ms. Se define la siguiente estructura para llevar el conteo de Time Out de los paquetes asociados a los distintos procesos:

```
typedef struct _TO_TIMER_FLAGS{
        int SNTP_TO_count;
        int ARP_TO_count;
        int ICMP_TO_count;
        unsigned char SNTP_TO_ON:1;
        unsigned char ARP_TO_ON:1;
        unsigned char ICMP_TO_ON:1;
} TO_TIMER_FLAGS;
```

Cuando se necesita arrancar un time out timer, se pone a 1 la bandera XXXX_ON correspondiente a dicho proceso y XXXX_TO_count se pone en un valor inicial.
La ISR por tanto, decrementará o no los contadores de acuerdo a si están o no encendidos, y al llegar a 0 ejecutaran la función de error.

```
ISR(TIMER2_COMP_vect){
        if(TF.SNTP_TO_ON)if(!--TF.SNTP_TO_count)SNTP_error();
        if(TF.ARP_TO_ON)if(!--TF.ARP_TO_count)ICMP_error();
        if(TF.ICMP_TO_ON)if(!--TF.ICMP_TO_count)ARP_error();
}
```

Los timers son encendidos inmediatamente después de enviar un paquete y son apagados al reconocerse su respuesta en la ISR de recepción de paquetes por lo cual, el valor inicial de los mismos se debe setear de acuerdo a los valores de retardos típicos en la red.

Pruebas realizadas

Dado que el sistema utiliza tanto protocolos de software y hardware estándar, las pruebas de funcionamiento no requirieron de mucha infraestructura.
Inicialmente se intentaron realizar las pruebas conectándonos directamente a la LAN del laboratorio de software pero luego de una jornada de pruebas insatisfactoria se decidió realizarlas en una red mas simple, para evitar al máximo problemas ajenos a nosotros. Esto hizo posible que pudiéramos realizar las pruebas sin tener que concurrir al laboratorio de software, haciendo de la tarea algo mas practico.
Por esto se utilizó a lo largo de todo el proceso de desarrollo una conexión por medio de cable UTP cruzado CAT 5 entre la tarjeta de red de la PC utilizada y la placa de desarrollo. Usamos como herramienta principal, el analizador de protocolos de red Wireshark, para poder capturar y visualizar el tráfico, y observar la correcta formación de los paquetes enviados. Además, se utilizó la función ping de Windows para poder verificar el funcionamiento de ARP e ICMP reply, así como para desencadenar interrupciones de paquete entrante en las primeras etapas de diseño.
Dado que el diseño se centro en el stack TCP/IP, y la funcionalidad de comunicación por intermedio de la UART y el puerto serie de la PC se agregó a último momento, el sistema se probó a lo largo del desarrollo iniciando los procesos vía código, antes de entrar al loop principal.
Para realizar dichas pruebas, no se necesita realizar muchos cambios en la configuración estándar de la PC, simplemente se debe tener cuidado de deshabilitar el firewall de Windows, o habilitar en el mismo el puerto 123 de UDP (correspondiente al servidor NTP). Además, se deben configurar los datos de IP del servidor SNTP y del gateway (grabados en EEPROM) con el IP de la PC. No es necesario instalar un servidor NTP (por lo menos en Windows XP) ya que este cuenta con uno propio aunque en otros SO podría llegar a ser necesario instalar uno.
Los protocolos se fueron probando a medida de que se escribían, primero ARP, luego ICMP y finalmente UDP.
Las pruebas se realizaron utilizando la herramienta de debug de AVR Studio, por medio de la interfaz JTAG del AVRdragon.
La función de SNTP timestamp_to_date() se probó pasándole por código un timestamp correspondiente a una fecha conocida y verificando los cálculos hechos paso a paso.
Luego de haber llegado a una versión definitiva del stack a entregar, se agregó la funcionalidad de

comunicación vía puerto serie. Se utilizo el programa “Hercules Setup”, una especie de hyperterminal para la comunicación vía puerto serie, configurándolo a 4800 bps y datos de 8 bits.
En este punto nos encontramos con algo inesperado, el programa debía enviar “Hola Mundo” por medio de la UART en la inicialización únicamente, y dicho mensaje se repetía a lo largo de la ejecución normal del programa. Luego de descartar que fuera un problema de comunicación de la UART, pusimos un breakpoint en el inicio de la rutina main lo que nos permitió ver que el programa se reseteaba aleatoriamente.
Para solucionar esto se realizaron algunas modificaciones, incluyendo la colocación de un pull-up externo en el pin de reset. Además se colocó en el código una ISR de interrupción mal atendida:
ISR(BADISR_vect){}
Y un breakpoint en la misma viéndose que una de las causas del reinicio del programa era que por error se habilitaba una interrupción (INT2) la cual no tenía una ISR asociada. Dicha interrupción a su vez, era desencadenada por la presencia de un LED puesto a tierra colocado en el pin de INT2 que también funciona como pin de I/O. Dicho LED fue utilizado en algún momento, pero luego se elimino del código y no del hardware, provocando de esta forma el error por lo cual fue removido.
Con estas dos modificaciones se solucionó el problema ya que ambas aportaban en mayor o menor medida al mismo.
En este punto, se procedió a realizar una prueba final del funcionamiento de la placa siendo esta exitosa. Se generaron por medio de la UART peticiones de fecha y horas las cuales fueron atendidas correctamente, procesándose exitosamente los paquetes SNTP y devolviendo por intermedio de la UART la fecha y hora en el formato pretendido.
A continuación se muestra una captura realizada en el Wireshark.

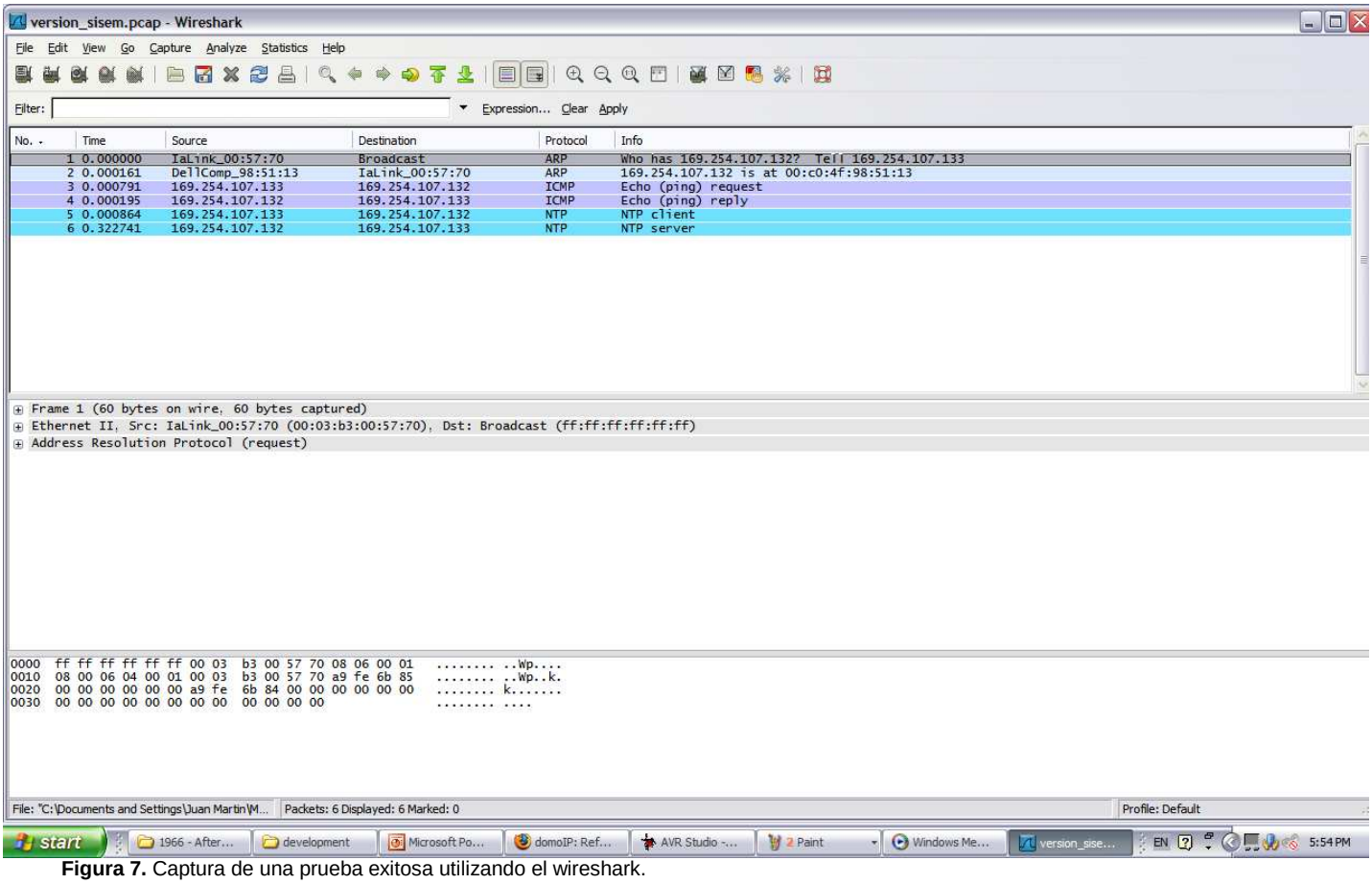


Figura 7. Captura de una prueba exitosa utilizando el wireshark.

Conclusiones

Los objetivos planteados fueron cumplidos de forma satisfactoria. Como se vió en la demo de la presentación final del curso, las peticiones de cada protocolo fueron generadas y luego procesadas de forma correcta por nuestro sistema, desplegando en pantalla la fecha y hora, como era lo esperado.
Se aplicaron técnicas de programación vistas en el curso y se ganó experiencia y soltura en el manejo de las herramientas de programación y debugging del microcontrolador. Se decidió utilizar la plataforma

AVRStudio conjuntamente con la suite de programas de código libre WinAVR para la programación y debugging del ATmega32 en vez del IAR Embedded Workbench por varias razones, pero fundamentalmente porque es la opción más utilizada en la comunidad AVR y el código que se encuentra de otros proyectos está escrito de forma compatible con el compilador AVR-GCC, incluido en el WinAVR.
La realización de este proyecto representa un hito importante para nuestro grupo, ya que vale recordar que el presente proyecto está enmarcado en un proyecto de fin de carrera en el cual se utilizará como base fundamenteal lo logrado hasta ahora.
El tamaño del código generado, aunque se trató de minimizar al máximo, representa una cantidad un poco preocupante en relación a la capacidad del ATmega32 (alrededor del 80%), en especial porque se planea agregar código extra más adelante para lograr otras funcionalidades.

Anexo.

Especificación del proyecto para Sistemas Embebidos año 2008

Descripción del problema a ser resuelto

Se diseñará un sistema de control remoto de dispositivos hogareños (electrodomésticos o similares) multiprotocolo. Dicho sistema permitirá a un usuario conectarse de forma segura a través de un PC o similar (cualquiera que maneje TCP/IP) a un dispositivo central (situado en el hogar, oficina, etc.) que se encargará de comandar, utilizando como canal de comunicación la línea eléctrica, actuadores conectados a los electrodomésticos. El sistema permitirá efectuar operaciones básicas (encendido, apagado, intensidad de luz) previéndose la posibilidad de ampliar su funcionalidad. Dicho comando será instantáneo o agendado permitiéndole al usuario la creación y modificación de eventos agendados así como la posibilidad de crear, agendar y ejecutar escenas programadas por el usuario.
El (los) protocolo(s) a utilizar en el control de los electrodomésticos se trata(n) de protocolo(s) sobre la línea eléctrica (Powerline Bus).
En definitiva el problema que este proyecto trata de resolver es el de controlar remotamente algunos aspectos de un hogar.

Antecedentes del proyecto

Los URLs que figuran a continuación son de proyectos que resuelven la comunicación con Internet usando microprocesadores de la familia AVR.

- http://www.avrportal.com/?page=avrnet.
- http://members.home.nl/bzijlstra/software/examples/enc28j60.htm

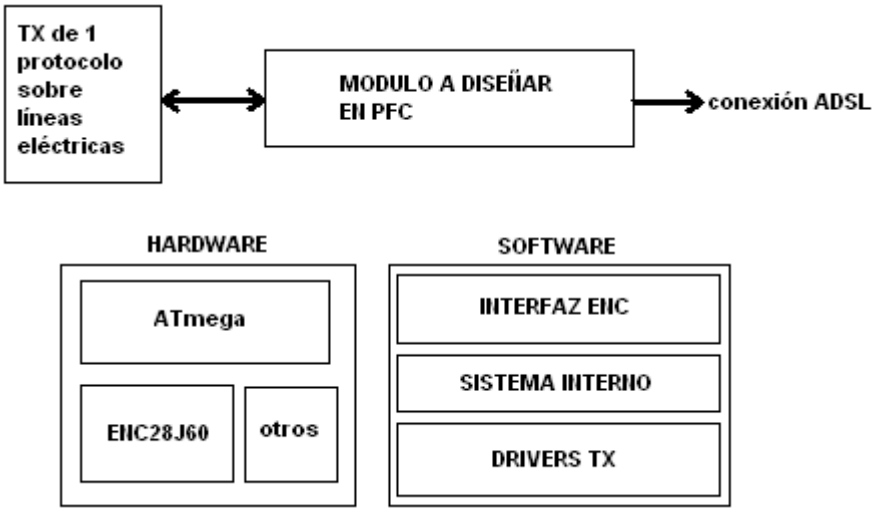
Hoy en día hasta donde sabemos no hay registros de soluciones pensadas para el protocolo PLCbus. Si existen soluciones para protocolos sobre líneas eléctricas más viejos como X10.

En cuanto a la literatura que usaremos nos remitiremos a:

- Datasheet del Atmega32
- Datasheet del ENC28J60
- Manual de uso del AVR dragon
- Tutoriales de C para software embebido
- Tutoriales del IAR y AVR
- Información disponible en la web y sobre el stack TCP/IP
- “An embedded Software Primer”.
- Alguna hoja de datos adicional como la del MAX232

Objetivos del proyecto

A continuación se muestra en un diagrama muy simplificado lo que se implementará para la materia Proyecto de Fin de Carrera, de modo que quede más claro lo que vamos a hacer como proyecto final para Sistemas Embebidos.



Se diseñaran:

1. la interfaz con el chip ENC28J60 (chip Ethernet) (software).
2. driver TX (que manejará el transmisor de un protocolo sobre líneas eléctricas).
3. hardware de desarrollo. (si bien no forma parte estrictamente hablando de la asignatura lo usaremos para estas y futuras pruebas).

En cuanto a los entregables:

Se entregarán los códigos funcionando y el HW de desarrollo.
Se verificará el correcto funcionamiento mediante las siguientes pruebas:

1. *prueba de comunicación TCP/IP.* Se enviarán y recibirán datos, usando una red Ethernet, verificando que el contenido de las variables correspondientes en memoria del micro sean los correctos.
2. *prueba de manejo del transmisor*. Para esto se programarán rutinas de prueba en el sistema interno con funciones básicas de automatización de hogares que “llamen” al driver y se observarán los actuadores para ver si hacen lo que se espera.

Alcance del proyecto y hardware adicional

El proyecto comprende el código de la interfaz con el ENC28J60 a efectos de poder transmitir y recibir datos mediante el protocolo TCP/IP y el código del driver para manejar el transmisor de automatización.

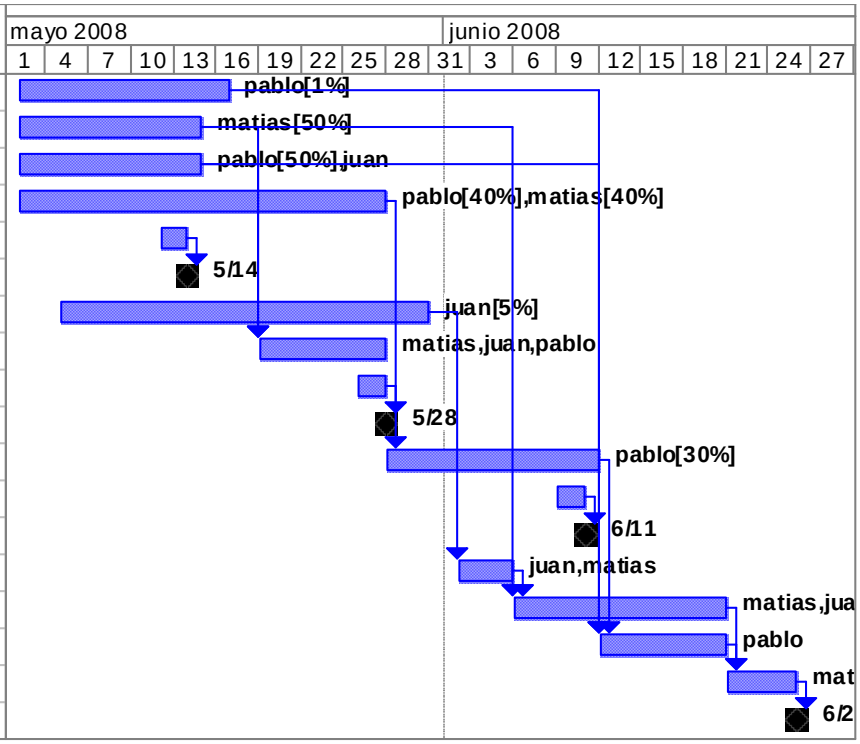
Se deberá hacer lo siguiente como pasos previos a la programación mismo:

- Estudiar el funcionamiento de la interfaz RS232 o USB de los microprocesadores ATmega.
- Estudiar el funcionamiento del chip Ethernet.
- Estudio del funcionamiento y estructura del stack TCP/IP.
- Adaptación del stack TCP/IP a las necesidades del proyecto
- Construcción del Hardware de desarrollo. Este placa deberá incluír el microprocesador, el chip Ethernet y todo lo necesario para que funcione (adaptadores de voltaje hacia el transmisor y hacia la interfaz Ethernet, conexiones, capacitores de bias, etc). Se usará conjuntamente con el AVRdragon.

Planificación

Se definieron para el proyecto las tareas que figuran a continuación, con precedencias y recursos asignados:

Id	Nombre de tarea
1	adquisición de chip ethernet
2	estudio funcionamiento int RS232/USB en Atmega
3	estudio de funcionamiento chip ethernet
4	estudio funcionamiento stack TCP/IP
5	preparación presentación 1
6	presentación 1
7	adquisición de módulos PLCbus
8	construcción del HW de desarrollo
9	preparación presentación 2
10	presentación 2
11	adaptación del stack TCP/IP
12	preparación presentación 3
13	presentación 3
14	corroboración de protocolos sobre powerlines
15	programación de divers para transmisor
16	pruebas de comunicación TCP/IP
17	preparación presentación final
18	FIN DEL PROYECTO



Los porcentajes que figuran arriba están hechos sobre el total de horas que cada miembro del equipo le dedicará por día al proyecto.

Se fijaron los siguientes 4 hitos del proyecto.

- *Presentación 1.* Está fijada para el día 14 de mayo. Esta constará de lo siguiente:
  1. explicación del funcionamiento del chip Ethernet
  2. explicación del funcionamientode las interfaces RS232/USB del ATmega.

- *Presentación 2.* Está fijada para el día 28 de mayo. Constará de lo siguiente:
  1. stack TCP/IP, qué contiene y como se implementa.
  2. hardware de desarrollo. Qué incluye y qué problemas se enfrentaron al diseñarlo.

- *Presentación 3.* Está fijada para el día 11 de junio. Constará de lo siguiente:
  1. cómo se adaptó el stack TCP/IP a nuestros objetivos.
  2. cómo se verificaron los protocolos de transmisión con los transmisores.
  3. cómo se encaró la programación de los drivers (no tiene porque estar terminada).

- *Presentación final.* Está fijada para el día 25 de junio. Constará de lo siguiente:

  Se mostrará el funcionamiento de ambas interfaces (hacia el transmisor y hacia Ethernet) y qué problemas se resolvieron para lograrlo. Se harán algunas conclusiones finales del proyecto.

Comentarios sobre planificación.

Como se aprecia arriba se planificó algo bastante distinto a lo que se hizo. En la planificación inicial, se incluía manejar dispositivos de control remoto de dispositivos, conectados en una red de domótica, sin embargo durante el transcurso del proyecto se enfocaron los esfuerzos en implementar adecuadamente la conexión con una LAN Ethernet, y los protocolos necesarios para dicho propósito; con el fin último de realizar y procesar un pedido de fecha y hora a un servidor NTP. Es decir, el objetivo mismo del proyecto cambió bastante. También se descartó la configuración de una interfaz USB por un tema de tiempos entre otras cosas. En cuanto a las horas empleadas, sin duda fueron muchas más de las previstas, y la carga fue despareja. La cantidad de horas que cada uno le dedicó al proyecto fueron aproximadamente:

Pablo: 265 horas
Matías: 80 horas
Juan: 65 horas
Total: 410 horas

El proyecto se terminó antes de lo previsto, aunque no de acuerdo a los objetivos fijados inicialmente en la planificación, sino de acuerdo a las necesidades que fueron surgiendo.