Apéndice A
Funciones de usuario

En la Tabla A-1 se describen brevemente las funciones de usuario creadas para facilitar a los desarrolladores la interacción con los diferentes módulos del microcontrolador MC9S12NE64.

Tabla A-1. Funciones de usuario del sistema de desarrollo

Funciones generales	
void RTI_init(void)	Inicializa la interrupción de tiempo real
void reset_code(void)	Reinicializa ejecución de código
void setPLL(unsigned char, unsigned char)	Configura parámetros del PLL (*Phase Lock Loop*) y lo enciende
funciones SCI	
void sci_init(void);	Inicializa SCI (comunicación asíncrona SCI1, velocidad de bus, etc)
void write_char(char)	Envía un carácter
void write_int(int)	Envía un entero
void write_string(char *)	Envía una cadena de caracteres
void write_hx(unsigned char)	Envía un dato hexadecimal
Funciones LCD	
void lcd_init(void)	Inicializa el LCD conectado al puerto H
void lcd_comand(unsigned char)	Envía un comando al LCD
void lcd_write(unsigned char)	Escribe un carácter al LCD
void lcd_position(unsigned char)	Configura la posición del cursor del LCD
void lcd_write_string(unsigned char *)	Escribe una cadena de caracteres al LCD
void lcd_write_int(int)	Escribe un dato de 16 bits al LCD
void lcd_write_long(unsigned long)	Escribe un dato de 32 bits al LCD
void lcd_write_hx(unsigned char)	Escribe un dato en formato hexadecimal al LCD
void delay_ms(unsigned int)	Retraso de n cantidad de milisegundos
void delay_us(void)	Retraso de 1 microsegundos
void delay_10us(void)	Retraso de 10 microsegundos
void delay_100us(void)	Retraso de 100 microsegundos
Funciones ADC	
void adc_init(void)	Inicializa el módulo ADC del MC9S12NE64 (configura el tipo y tiempo de muestreo)
void adc_powerup(void)	Enciende el módulo ADC
void adc_powerdown(void)	Apaga el módulo ADC
unsigned int adc_convert(unsigned char)	Configura el canal indicado (dato char) y regresa la lectura de dicho canal
void adc_toint(unsigned int)	Imprime en el LCD el valor leído en el ADC con la instrucción anterior.
Funciones Teclado	
void init_teclado(void)	Inicializa el teclado, configurando el puerto G como entradas(G4-G7) y salidas (G0-G3),
unsigned char leer_tecla(void)	Resuelve cual de las 16 teclas ha sido presionada.

Apéndice B
Monitor Serie para microcontroladores HCS12

Descripción y propósito del monitor serie

El *Monitor Serie* para microcontorladores de la familia HSC12 es un programa con un tamaño aproximado de 2 KB, el cual soporta 23 comandos básicos que permiten programar y depurar código en el módulo FLASH/EEPROM con un cable serial RS-232 conectado al microcontrolador en la interfaz SCI y a una computadora (con un ambiente de desarrollo Integrado IDE) en el puerto RS-232. El monitor serie también soporta comandos para reiniciar el microcontorlador, leer y modificar localidades de memoria (incluidas la FLASH/EEPROM), leer y modificar registros del microcontrolador.

Dado que el funcionamiento del Monitor Serie utiliza la interfaz SCI0 ésta debe estar dedicada al Monitor Serie exclusivamente y se debe descartar el funcionamiento de las aplicaciones de usuario a través esta interfaz.

El Monitor Serie usa el COP watchdog para la función de reinicio (reset) en frio, debido a esto el código de usuario no debe habilitar o deshabilitar la función del COP, es decir, no debe escribir valores en el registro COPCTL. En caso de que el código de usuario tome control a partir del reset y una interrupción del SCI0 o del SWI intente acceder al Monitor Serie, éste podría no funcionar porque los registros de inicialización del SCI, el PLL y la memoria no estarían configurados como deberían para un reset.

Cómo obtener el Monitor Serie

Este programa se puede obtener de la página de freescale *www.freescale.com*, sólo se debe buscar la nota de aplicación del monitor serie AN2548 y descargar los archivos .zip, que contienen el monitor serie.

Localización en la memoria del microcontrolador

El Monitor Serie se guarda en un bloque de memoria flash de 2KB (de la dirección $F800 a la $FFFF) protegido contra escritura para prevenir que se cambie o borre accidentalmente. A este bloque de memoria protegido sólo se puede acceder mediante un Programador y Depurador del BDM (PYDBDM) con el cual el Monitor Serie se carga al microcontrolador. En las aplicaciones que usan el monitor serie con la interfaz SCI del microcontrolador conectado a una computadora (al puerto serial) no hay forma de borrar accidentalmente el programa del Monitor Serie.

El sistema de la memoria FLASH ejecuta los comandos de programación y de borrado, los tiempos que utilizan estos comandos se determinan por la velocidad del reloj en el módulo FLASH, el cual debe estar entre los 150 y 200 KHz para funcionar apropiadamente. El Monitor Serie establece la velocidad de reloj del módulo FLASH en 200 KHz basado en la frecuencia del oscilador.

Configuración para su funcionamiento con el MC9S12NE64

En algún momento dado, el microcontrolador opera ya sea en modo Monitor o modo Aplicación. El *modo Monitor* es el modo en el que el microcontrolador está ejecutando el código del Monitor Serie y mantiene el control del microcontrolador esperando algún comando a través de la interfaz serial; el *modo Aplicación* es el modo en el que el microcontrolador está ejecutando el programa de aplicación del usuario.

Debido a que el Monitor Serie es una aplicación general para los microcontroladores de la familia HSC12, se debe realizar una configuración en su código para que funcione correctamente con el microcontrolador MC9S12NE64. Se debe configurar la memoria, el oscilador y un interruptor para correr y cargar programas.

En la Tabla B-1 se presenta la configuración y los cambios que se deben realizar al archivo S12SerMon2r1.def para que el monitor serie funcione con el MC9S12NE64.

Tabla B-1. Configuración al archivo del monitor serie.

Configuración	Modificaciones al archivo S12SerMon2r1.def

Frecuencia del Oscilador. EL oscilador debe operar a 24MHz y tiene una tolerancia de 4%, permitida por el estándar RS-232, lo que permite que el oscilador sea de una velocidad de 25MHz, lo cual se indica con la línea: **OscFreq: equ 25000.**	• Descomentar las líneas:
	`OscFreq: equ 25000 ;Enter Osc speed`
	`initSYNR: equ $00 ; mult by synr + 1 = 1 (25MHz)`
	`initREFDV: equ $00 ; divide by Refdiv + 1`
	• Comentar las líneas:
	`;OscFreq: equ 8000 ;Enter Osc speed`
	`;initSYNR: equ $02 ; mult by synr + 1 = 3 (24MHz)`
	`;initREFDV: equ $00`

Memoria. Se debe configurar para indicar al monitor serie dónde comienza y termina la memoria EEPROM, la FLASH y la RAM.	• Agregar las líneas de código:
	`NE64:`
	`EEpromStart: equ $0400 ;eeprom start`
	`EEpromEnd: equ $1FFF ;eeprom end`
	`RamStart: equ $2000 ;first RAM location`
	`SectorSize: equ $0200 ;Flash sector size`
	`FlashBlks: equ $04 ;# of flash blks`
	`LowestPage: equ $30 ;lowest page`
	`PagesBlk: equ $04 ;pages per block`
	`softwareID4: equ $8201 ;device ID #`

Interruptor Run/Load. El sistema que cuenta con un monitor serie debe incorporar un interruptor asociado a una terminal de entrada/salida de propósito general para correr el programa. En este caso se utiliza el puerto J en la terminal PTJ0. Dicha terminal se utiliza para indicar cuál de los dos modos de operación del microcontrolador (Monitor y Aplicación) va a tener el control del programa.
Cuando el PTJ0=0 el Monitor Serie se encarga de la ejecución del programa y cuando el PTJ0=1 el microcontrolador es el encargado de dicha ejecución.
	• Agregar las líneas:
	`SwPort: equ PTJ ;pushbutton sw connected to port J`
	`Switch: equ PTJ0 ;switch connected to PTJ bit-0`
	`SwPullup: equ PERJ ;pullup enable for sw port`
	`mSwPullup: equ PTJ0 ;Mask value to enable for sw port`

Descargar el Monitor Serie al microcontrolador

Una vez configurado, el Monitor Serie se debe descargar al microcontrolador usando el IDE CodeWarrior y un programador (véase el tema "Descargar un programa al microcontrolador" del apéndice C).

Apéndice C
Ambiente de Desarrollo Integrado (IDE) CodeWarrior

Descripción del IDE CodeWarrior

La compañía *Freescale Semiconductor* proporciona un Ambiente de Desarrollo Integrado (IDE – *Integrated Development Environment*), denominado *CodeWarrior*, para la programación de sus microcontroladores. Este IDE es útil para escribir código para muchos modelos de microcontroladores de *Freescale* y permite realizar depuración, emulación y simulación de sus dispositivos.

Existen varias versiones del *IDE CodeWarrior*, las cuales permiten compilar y ensamblar código para diferentes familias de microcontroladores y varían en la cantidad de código a compilar. Existen versiones libres y versiones con costo. La versión utilizada para el desarrollo de esta tesis es la versión libre 5.9 que se obtiene de la página de *Freescale* y es una edición especial para el microcontrolador NE64 limitada a compilar 32 KB de programa.

CodeWarrior incluye una máquina virtual que permite simular el CPU del microcontrolador, los periféricos y las interrupciones. Con esta opción es posible hacer pruebas del comportamiento del microcontrolador de una manera fácil y práctica.

La creación del código con CodeWarrior se puede hacer en diferentes lenguajes, como ensamblador, C, C++ y java, entre otros, para la programación de microcontroladores de 8, 16 y 32 bits.

La programación del código se organiza en proyectos. A la colección de documentos con los cuales se construye un archivo de salida se le denomina *objetivo*, un proyecto contiene uno o más objetivos.

Los objetivos indican la interfaz de conexión entre la computadora y el dispositivo programador, por ejemplo un programador conectado por el puerto serie del microcontrolador. Cuando se utiliza el puerto serie como puerto de programación se debe haber cargado previamente en la memoria del microcontrolador un programa llamado *Monitor Serie* (revisar el apéndice B).

El compilador crea específicamente una colección de documentos para cada interfaz de comunicación (como el programador o el simulador del microcontorlador), los cuales se encargan de crear el archivo de salida necesario para cada interfaz.

Cada objetivo puede contener diferentes tipos de archivos como archivos de código fuente, librerías, archivos de texto y archivos de configuraciones. Cada objetivo en un proyecto puede compartir los mismos archivos, pero tiene sus propias configuraciones.

Creación de un proyecto y programación del código

Para la programación del código es necesario crear un proyecto de CodeWarrior, se debe abrir el IDE y seguir los siguientes pasos:

1. Hacer click en el menú *File* y seleccionar la opción *New* (ver Figura C.1).

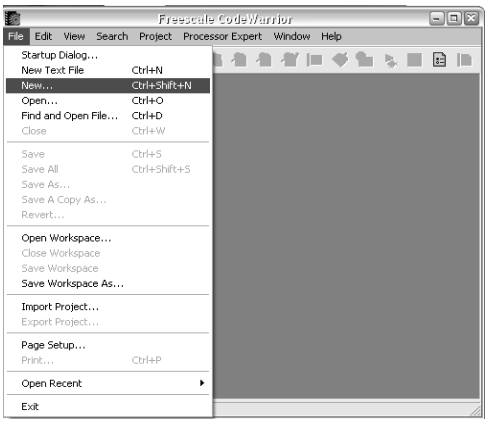


Figura C.1. Creación de un proyecto en CodeWarrior

2. En la ventana que aparece se debe seleccionar la pestaña *Project* y poner el nombre del proyecto en el campo *Project name*, escribir la ruta en la cual se va a guardar el proyecto en el campo *Location* y hacer click en *Aceptar* (ver Figura C.2).

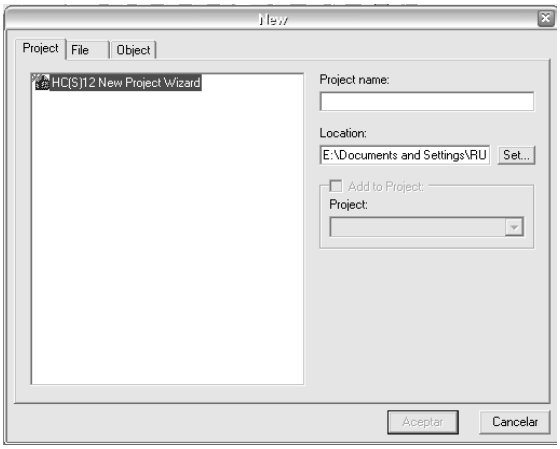


Figura C.2. Nombre y ruta del nuevo proyecto

3. En la nueva ventana que aparece hacer click en en botón *Siguiente*.

4. Seleccionar el nombre del microcontrolador que se va a usar en el nuevo proyecto y hacer click en el botón *Siguiente* (ver Figura C.3).

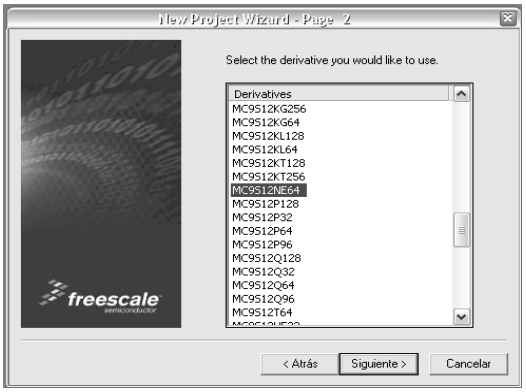


Figura C.3. Selección del microcontrolador a programar

5. Seleccionar el lenguaje en el que se va a realizar el proyecto y hacer click en el botón *Siguiente* (ver Figura C.4).

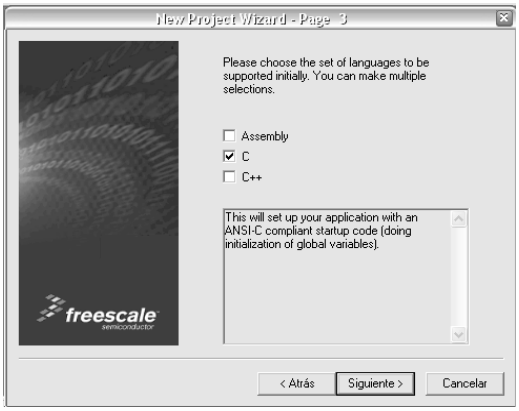


Figura C.4. Selección del lenguaje de programación

6. Hacer click en el botón *Siguiente* de las cuatro ventanas que aparecen después.

7. En la ventana siguiente seleccionar las opciones
   - Full chip simulation
   - P&E Multilink
   - HSC12 Serial Monitor

   y hacer click en el botón Finalizar (ver Figura C.5).

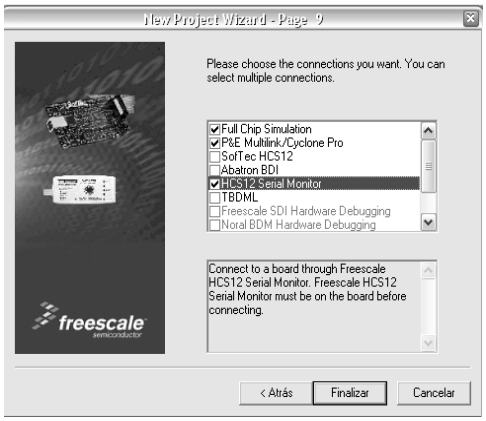


Figura C.5. Selección de los dispositivos que se usaran para la programación

Estructura de un proyecto en CodeWarrior

En la Figura C.6 se muestra la estructura del proyecto de CodeWarrior para lenguaje C que se acaba de crear con el proceso anterior. El contenido de las carpetas se muestra en la Figura C.7.

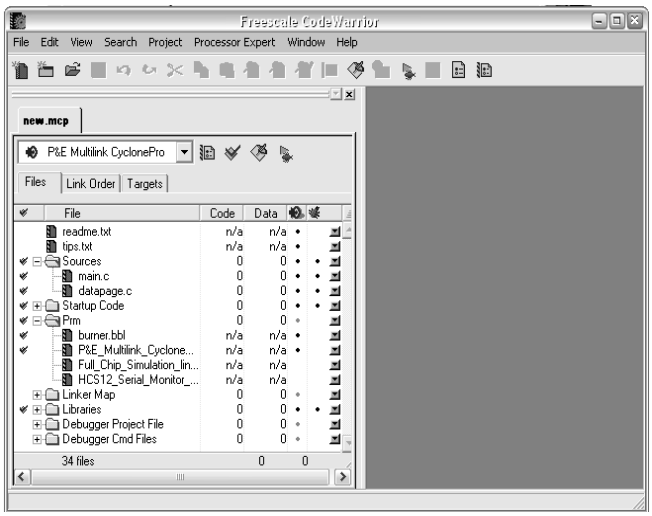


Figura C.6. Estructura de un proyecto en C

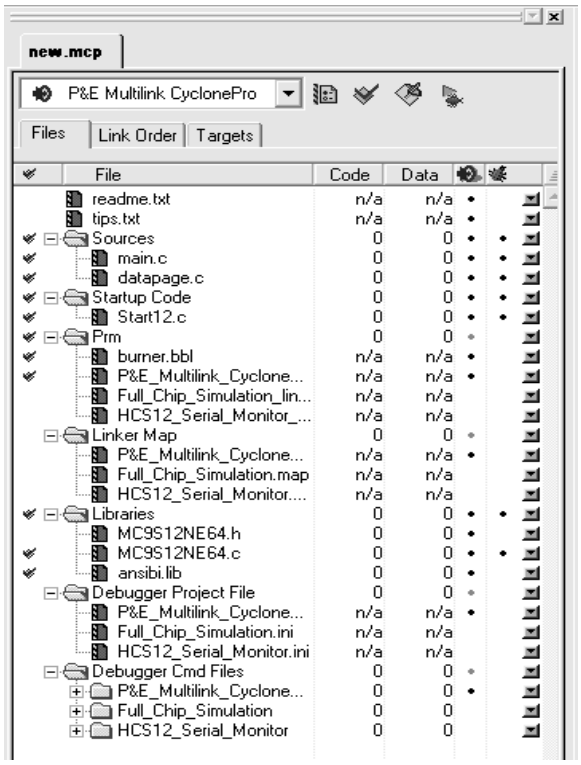


Figura C.7. Contenido de las carpetas de un proyecto

El proyecto contiene diferentes archivos organizados en una estructura de árbol, incluyendo archivos de texto, código fuente, librerías y los archivos de configuraciones propios para cada objetivo. La Tabla C.1 muestra el contenido de un proyecto.

Tabla C.1. Descripción de las carpetas de un proyecto

Archivo o carpeta	Contenido
archivo *readme.txt*	Breve descripción del proyecto, información que puede ser de utilidad, la estructura del proyecto e información a cerca de dónde se puede obtener más ayuda.
carpeta *Sources*	Código fuente de la aplicación, incluye la función *main.c.*

carpeta *Startup Code*	Archivos que sirven para inicializar el proyecto (invoca el código de usuario).
carpeta *Prm*	Archivos para generar registros S para el depurador y archivos que definen los segmentos de memoria (**.prm*).
carpeta *Libraries*	Librerías necesarias para la utilización de funciones para programación en lenguaje C.
carpeta *Debugger Project File*	Archivo **.ini* de inicialización para cada objetivo o dispositivo al que se va conectar el depurador del ambiente de desarrollo.
carpeta *Dedugger Cmd Files*	Archivos de comandos para la conexión con cada uno de los dispositivos u objetivos.

Simulación y depuración de código

Para verificar los programas escritos en CodeWarrior se utiliza el botón *Make* o el menú *Project* → *Make*. Al ejecutar este comando se muestra una ventana que indica los errores, advertencias o si todo es correcto. Si se encuentran errores o advertencias aparece un mensaje indicando cuál fue el error y en dónde se produjo. En la figura C.8 se muestra la ventana obtenida para una ejecución de prueba.

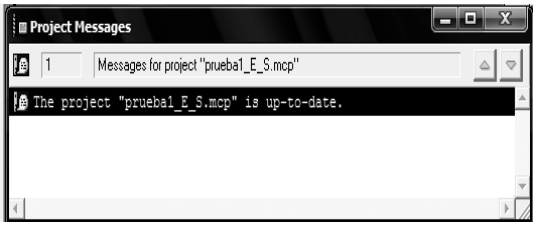


Figura C.8 Resultado de la opción Make para el programa de prueba

Para ejecutar el programa se debe seleccionar de la lista que aparece en la parte superior de la estructura del proyecto (ver figura C.9) el dispositivo usado para la programación y hacer click en el botón *Debug* o seleccionar la opción *Project* → *Debug*, la cual permite depurar y simular el sistema en tiempo real. Para este fin aparece una nueva ventana en el ambiente de desarrollo llamada *True–Time Simulator and Real–Time Debugger,* que se muestra en la Figura C.10.

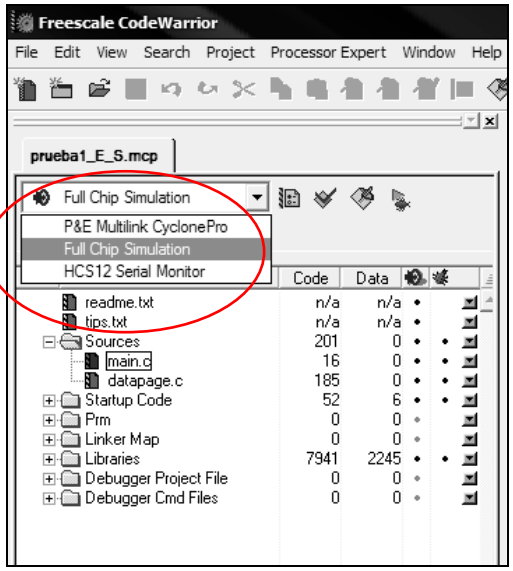


Figura C.9 Selección del programador a usar en la ejecución

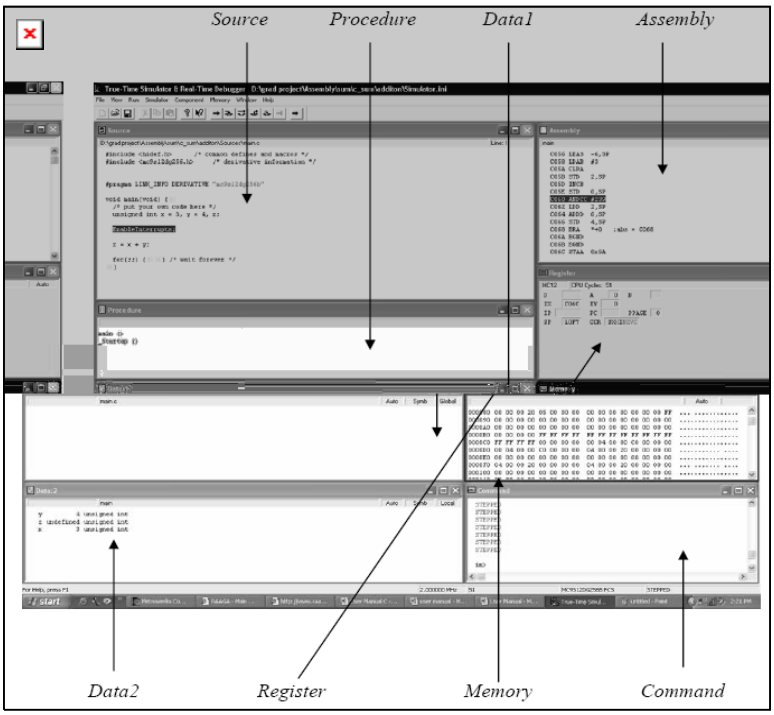
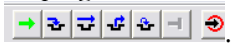


Figura C.10. Ventana *True–Time Simulator and Real–Time Debugger*

Esta ventana cuenta con varias secciones (pequeñas ventanas) que permiten conocer el estado de diferentes procesos del microcontrolador y sus variables (ver Tabla C-2).

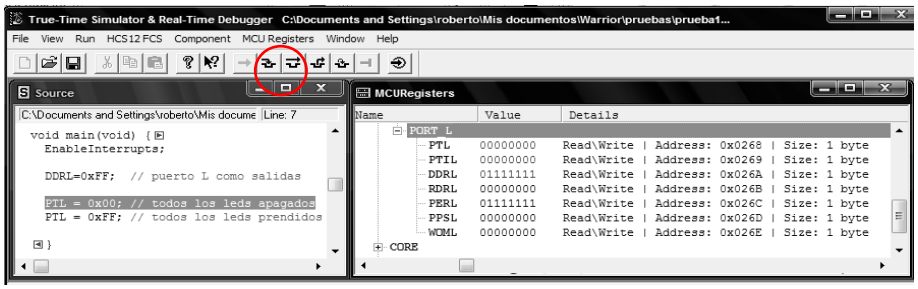
Tabla C-2 Descripción del contenido de las ventanas de la Figura C.9

Ventana	Contenido
Source	Muestra el código fuente que se está usando.
Procedure	Indica el proceso que se está llevando a cabo o la función que se está ejecutando.
Data1, *Data2*	Muestran los valores de las variables que intervienen en el programa.
Assembly	Muestra el código del programa en lenguaje ensamblador.
Register	Muestra el contenido de los registros del modelo de programación del Microcontrolador.
Memory	Muestra el contenido de las localidades de memoria.
Command	Muestra la acción que está ejecutando el depurador.

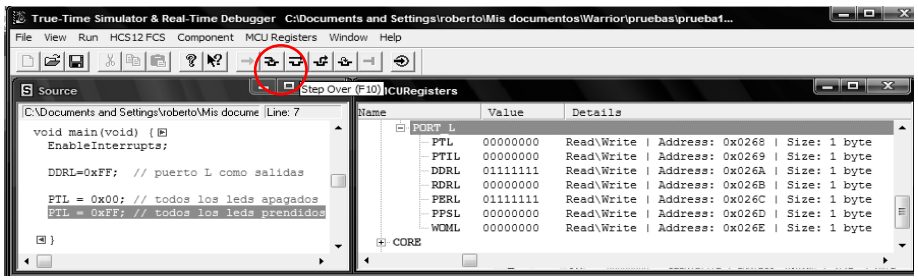
Para simular o jecutar un programa desde CodeWarrior se utilizan los botones de ejecución de la parte superior de la ventana:  . Con el primer botón (*Start/Continue*) se ejecuta el código de manera continua, es decir, una instrucción trás otra. El segundo botón (*Single Step*) permite ejecutar las instrucciones una a una cada vez que se presiona, pero si la instrucción corresponde a la invocación de una función, la instrucción que se ejecuta es la primera de dicha función, es decir, la ejecución binca al código de la función invocada y al terminar dicha función la ejecución continúa desde el punto donde fue invocada. El tercer botón (*Step Over*) permite ejecutar las instrucciones una a una, pero si la instrucción corresponde a una función la ejecución no brinca al código de dicha función, sino que ejecuta toda la función como una sóla instrucción. Para salir de una función y regresar al código que la invocó se utiliza el cuarto botón (*Step Out*). El último botón permite detener una ejecución en curso.

En las Figura C.11 se presentan los resultados de la simulación de un programa que configura el puerto L como salidas, modificando el contenido del registro DDRL y posteriormente escribe valores en las salidas de dicho puerto. Las instrucciones del programa se ejcutan con el botón *Step Over* (tercer botón) y el resultado tras presionar el botón en cada ocasión se observa en las localidades de memoria correspondientes a los registros del puerto L.

a)



b)



c)

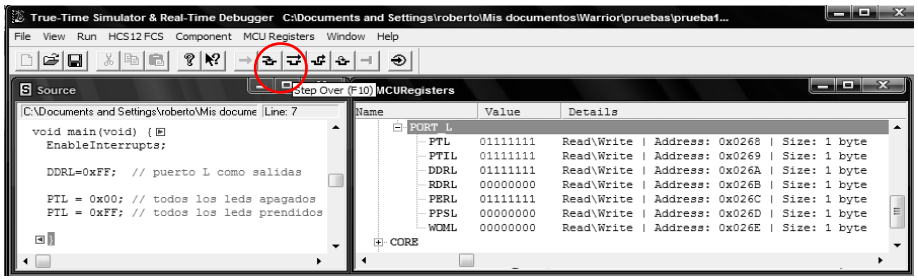


Figura C.11. Resultados de la simulación del programa.
a) Se ejecuta la instrucción DDRL=0; Se observa el resultado en el registro DDRL.
b) Se ejecuta la instrucción PTL=0x00; Se observa el resultado en el registro PTL.
c) Se ejecuta la instrucción PTL=0xFF; Se observa el resultado en el registro PTL.

Descargar un programa al microcontrolador

Para descargar el programa a la memoria del microcontrolador existen dos formas:

- **Con un programador**

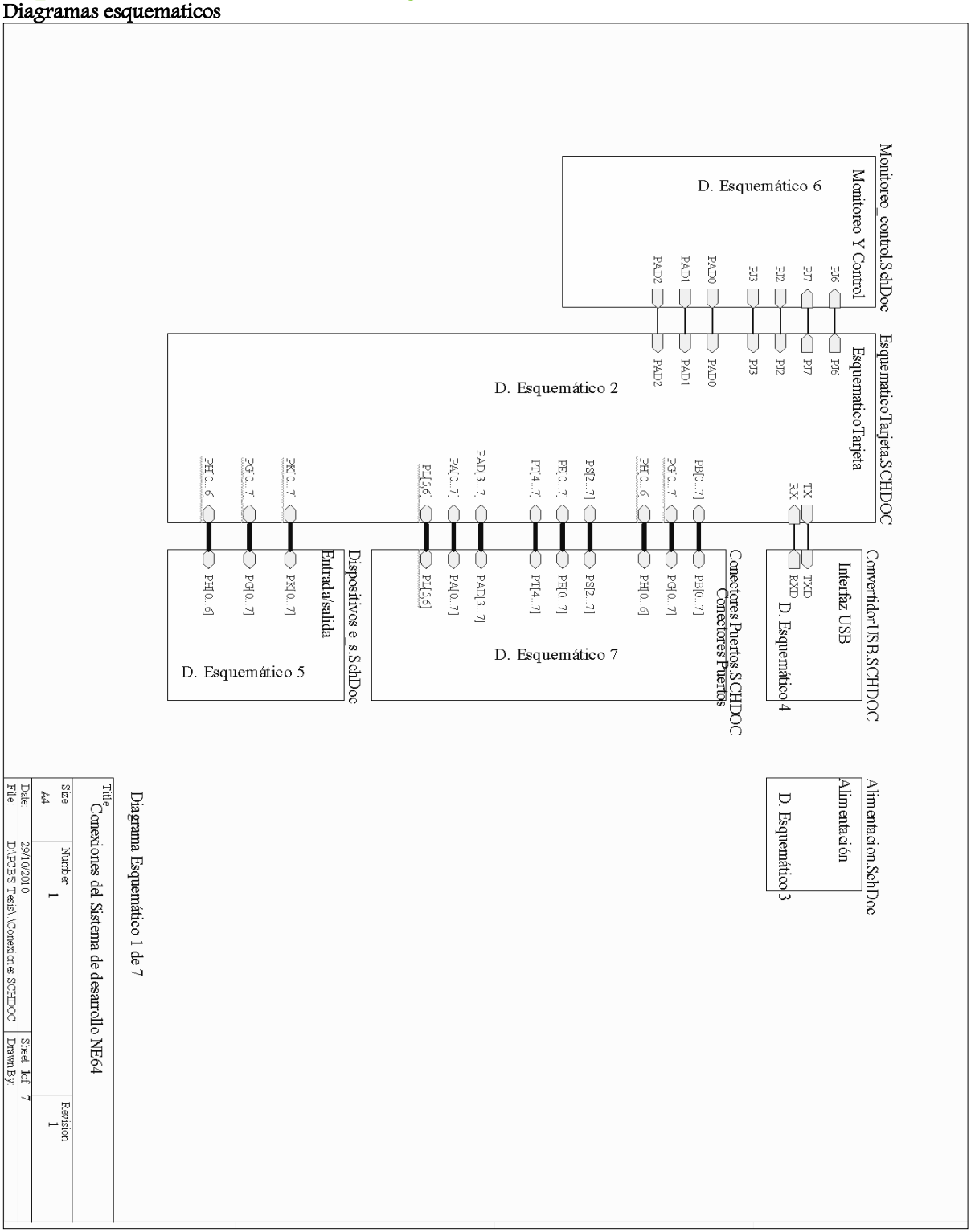
  Se debe seleccionar el programador *P&E Multilink* de la lista de dispositivos usados para la programación y hacer click en el botón *Debug*. Si la comunicación entre el programador, el microcontrolador y la computadora es correcta, entonces el código se descarga al microcontrolador. En el momento en el que el código ya se encuentra en la memoria, si la ventana del depurador continúa abierta, el control del programa lo tiene el depurador y es posible usar las mismas opciones que tiene la simulación del programa, cuyo resultado se puede ver directamente en el microcontrolador (depuración del sistema).
  Si la ventana de depuración se cierra y se presiona el botón de reset del microcontrolador, éste asume el control del programa almacenado en su memoria.

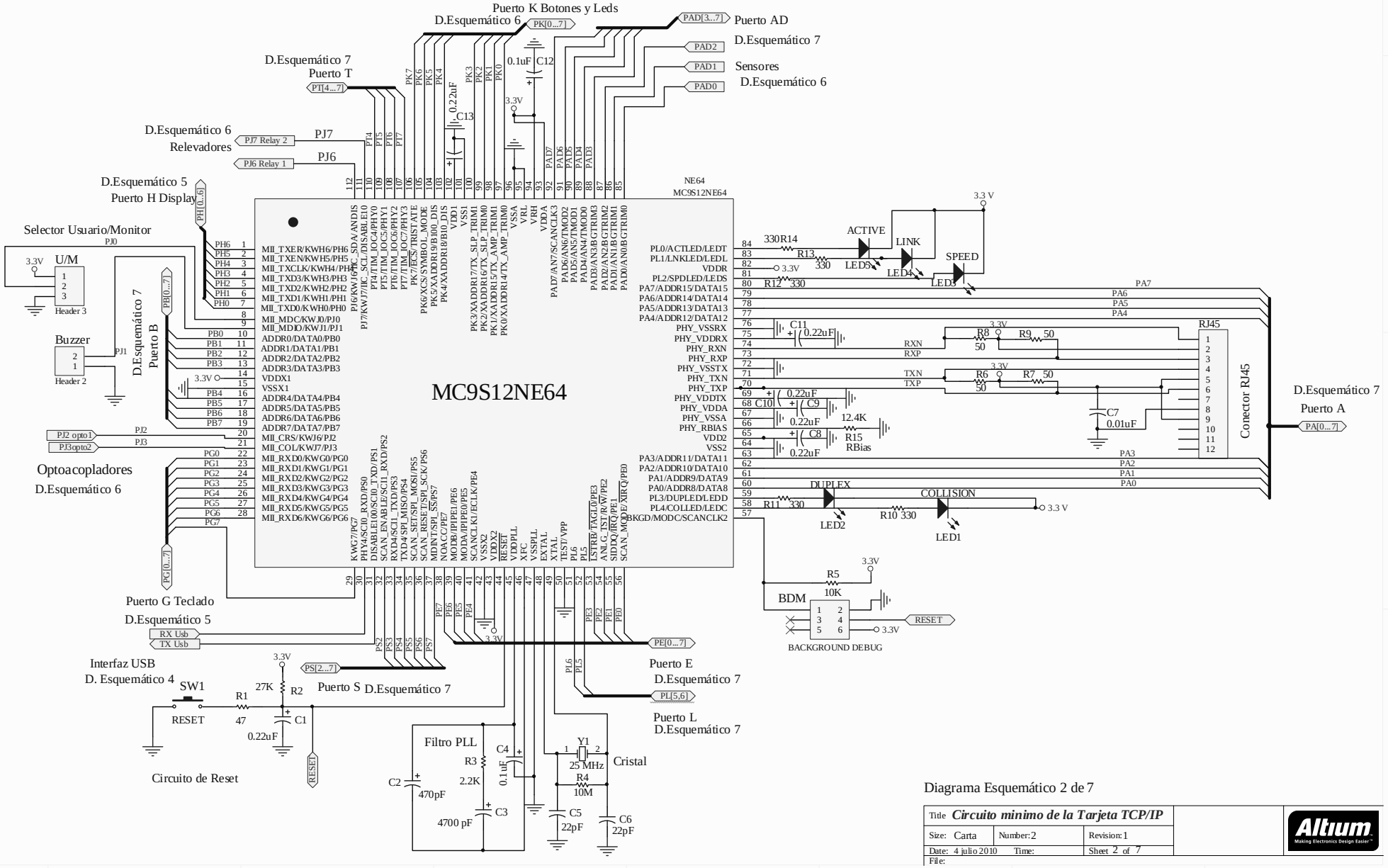
- **A través del Monitor Serie.**

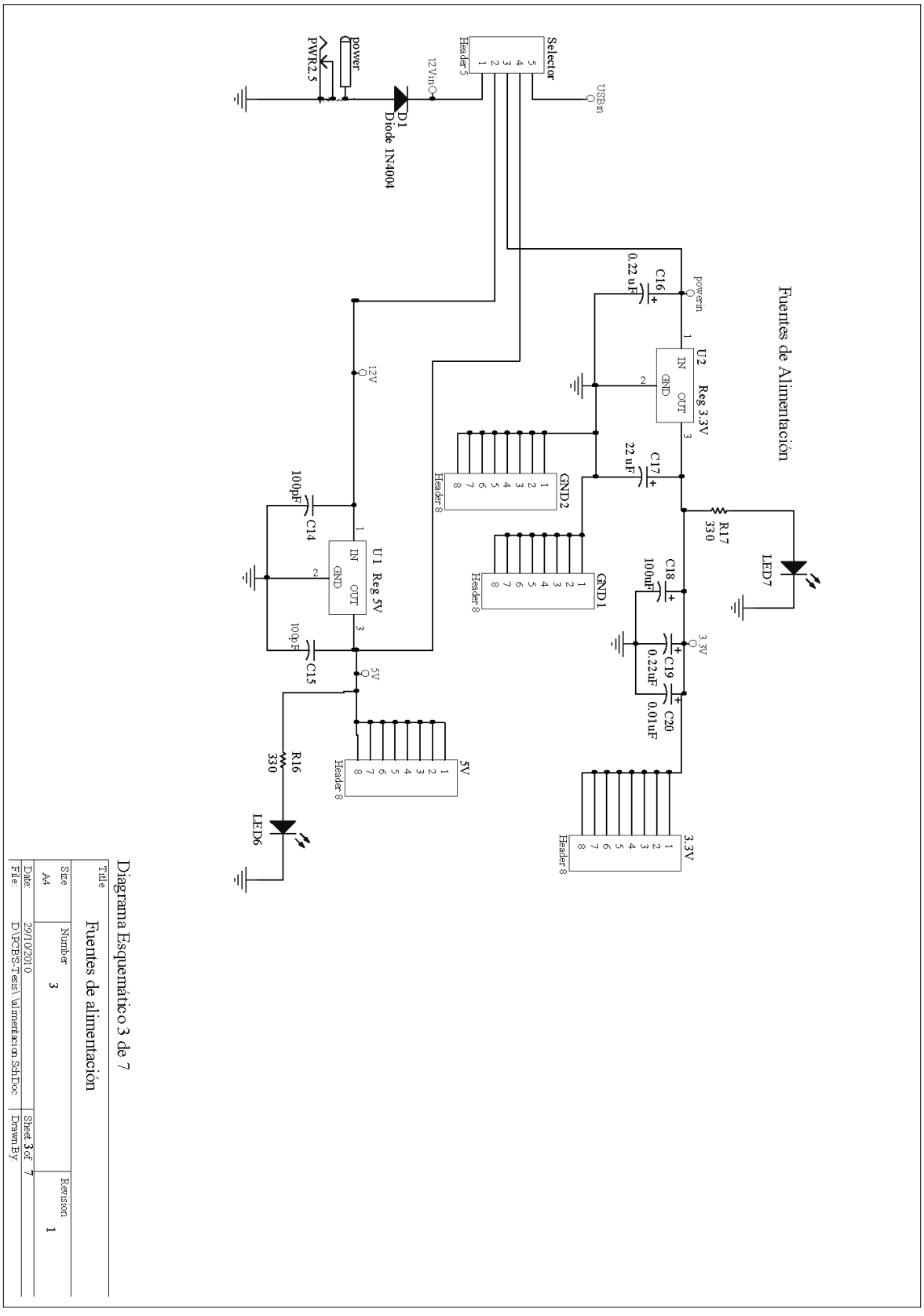
  Se debe seleccionar como programador el *Monitor* de la lista de dispositivos usados para la programación y hacer click en el botón *Debug,* el cual busca una conexión establecida en un puerto serie virtual de la computadora, si se muestra un error se debe especificar en qué número de puerto está instalado el puerto serie virtual del convertidor Serie–USB (chip FT232 de la tarjeta de desarrollo) y reintentar establecer la comunicación, para lo cual será necesario presionar el botón de reset del microcontrolador y así poder iniciar la programación.

Apéndice D
Especificaciones de las tarjetas

Diagramas esquematicos




MC9S12NE64
NE64
MC9S12NE64
Puerto K Botones y Leds
D.Esquemático 6
PK[0...7]
Puerto AD
PAD[3...7]
D.Esquemático 7
PAD2
PAD1
PAD0
Sensores
D.Esquemático 6
D.Esquemático 7
Puerto T
PT[4...7]
D.Esquemático 6
Relevadores
PJ7 Relay 2
PJ7
PJ6 Relay 1
PJ6
D.Esquemático 5
Puerto H Display
PH[0...6]
Selector Usuario/Monitor
PJ0
U/M
Header 3
Buzzer
PJ1
Header 2
D.Esquemático 7
Puerto B
PB[0...7]
PJ2 opto1
PJ3opto2
PJ2
PJ3
Optoacopladores
D.Esquemático 6
Puerto G Teclado
D.Esquemático 5
PG[0...7]
RX Usb
TX Usb
Interfaz USB
D. Esquemático 4
SW1
RESET
R1
47
C1
0.22uF
27K
R2
Circuito de Reset
PS[2...7]
Puerto S D.Esquemático 7
PE[0...7]
Puerto E
D.Esquemático 7
PL[5,6]
Puerto L
D.Esquemático 7
Filtro PLL
R3
2.2K
C4
0.1uF
C2
470pF
C3
4700 pF
Y1
25 MHz
Cristal
R4
10M
C5
22pF
C6
22pF
0.1uF C12
0.22uF C13
ACTIVE
LINK
SPEED
LED3
LED4
LED5
330R14
R13
330
R12
330
3.3 V
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
C11
0.22uF
PHY_VSSRX
PHY_VDDRX
PHY_RXN
PHY_RXP
PHY_VSSTX
PHY_TXN
PHY_TXP
PHY_VDDTX
PHY_VDDA
PHY_VSSA
PHY_RBIAS
VDD2
VSS2
C10
C9
C8
0.22uF
12.4K
R15
RBias
RXN
RXP
TXN
TXP
R8
R9
50
R6
R7
50
RJ45
Conector RJ45
C7
0.01uF
D.Esquemático 7
Puerto A
PA[0...7]
DUPLEX
COLLISION
LED2
LED1
R11
330
R10
330
BDM
R5
10K
RESET
BACKGROUND DEBUG
Diagrama Esquemático 2 de 7
Title Circuito minimo de la Tarjeta TCP/IP
Size: Carta
Number: 2
Revision: 1
Date: 4 julio 2010
Time:
Sheet 2 of 7
File:
Altium
Making Electronics Design Easier


Fuentes de Alimentación
Selector
Header 5
12VinO
USBin
power
PWR2.5
D1
Diode 1N4004
C16
0.22 uF
powerin
U2 Reg 3.3V
IN
GND
OUT
C17
22 uF
R17
330
LED7
C18
100uF
C19
0.22uF
C20
0.01uF
3.3V
GND2
GND1
12V
C14
100pF
U1 Reg 5V
C15
100pF
5V
R16
330
LED6
Diagrama Esquemático 3 de 7
Fuentes de alimentación
Number 3
Revision 1
Size A4
Date 29/10/2010
File D:\PCBS-Tesis\alimentacion.SchDoc
Sheet 3 of 7
Drawn By:

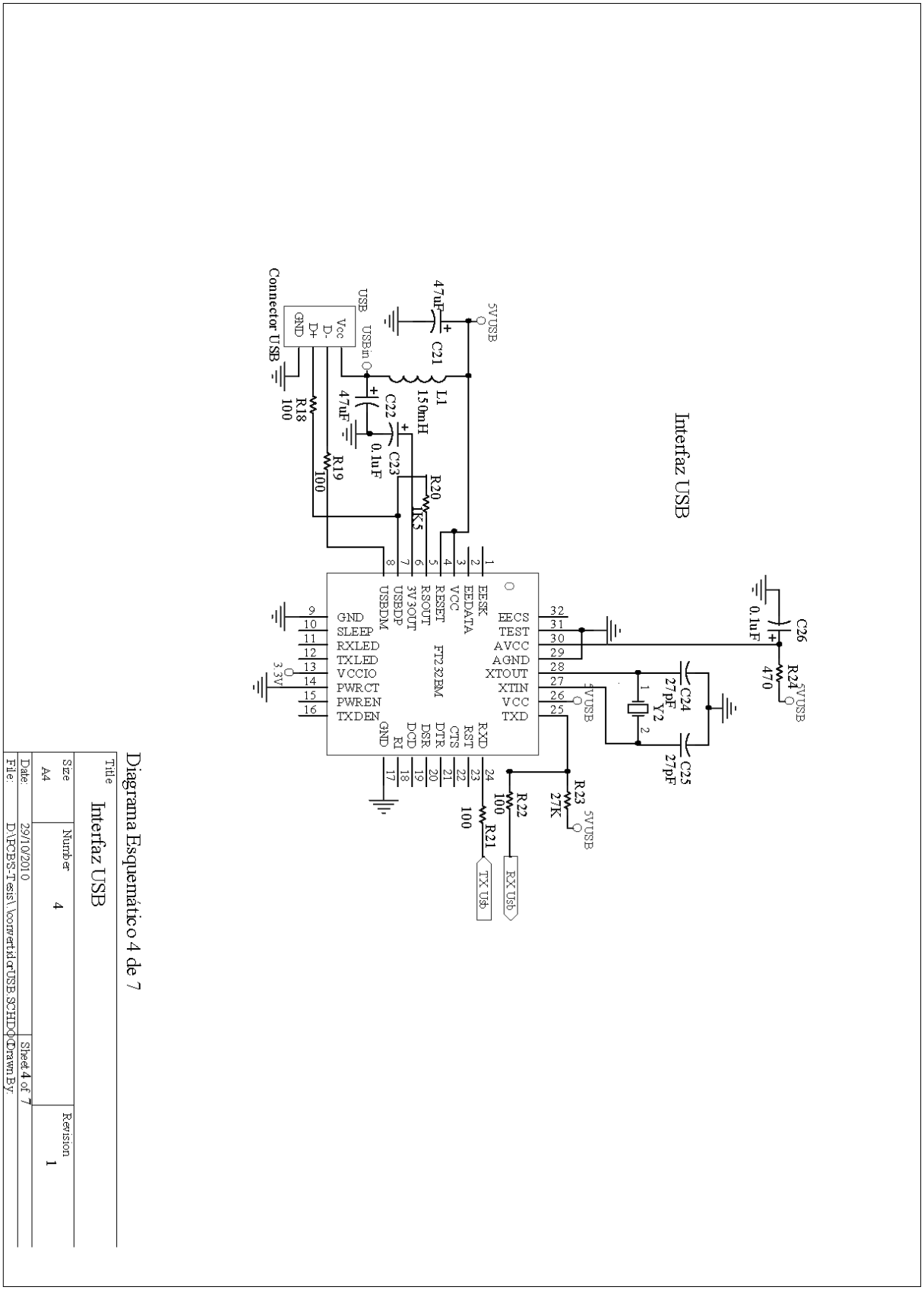
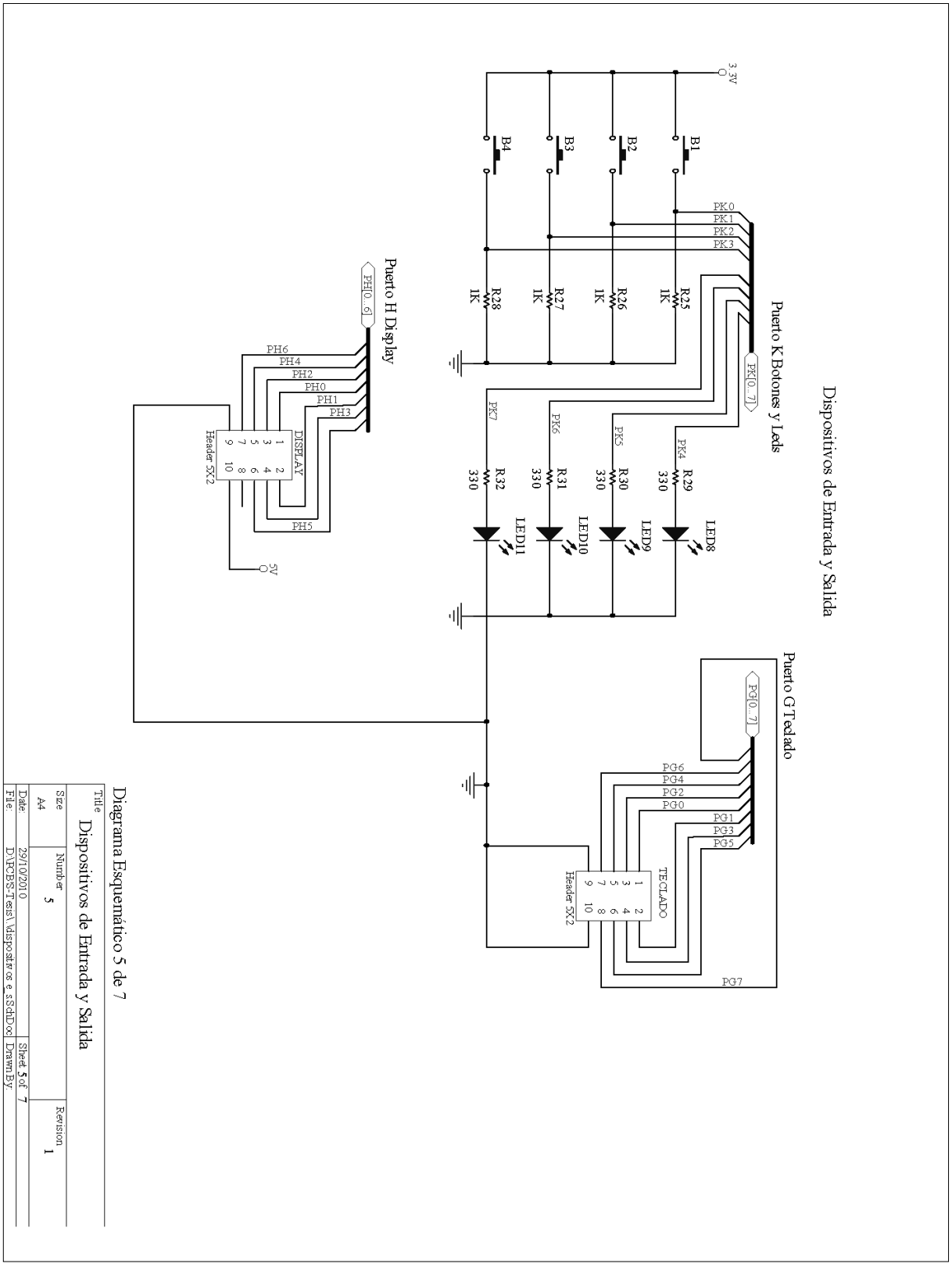
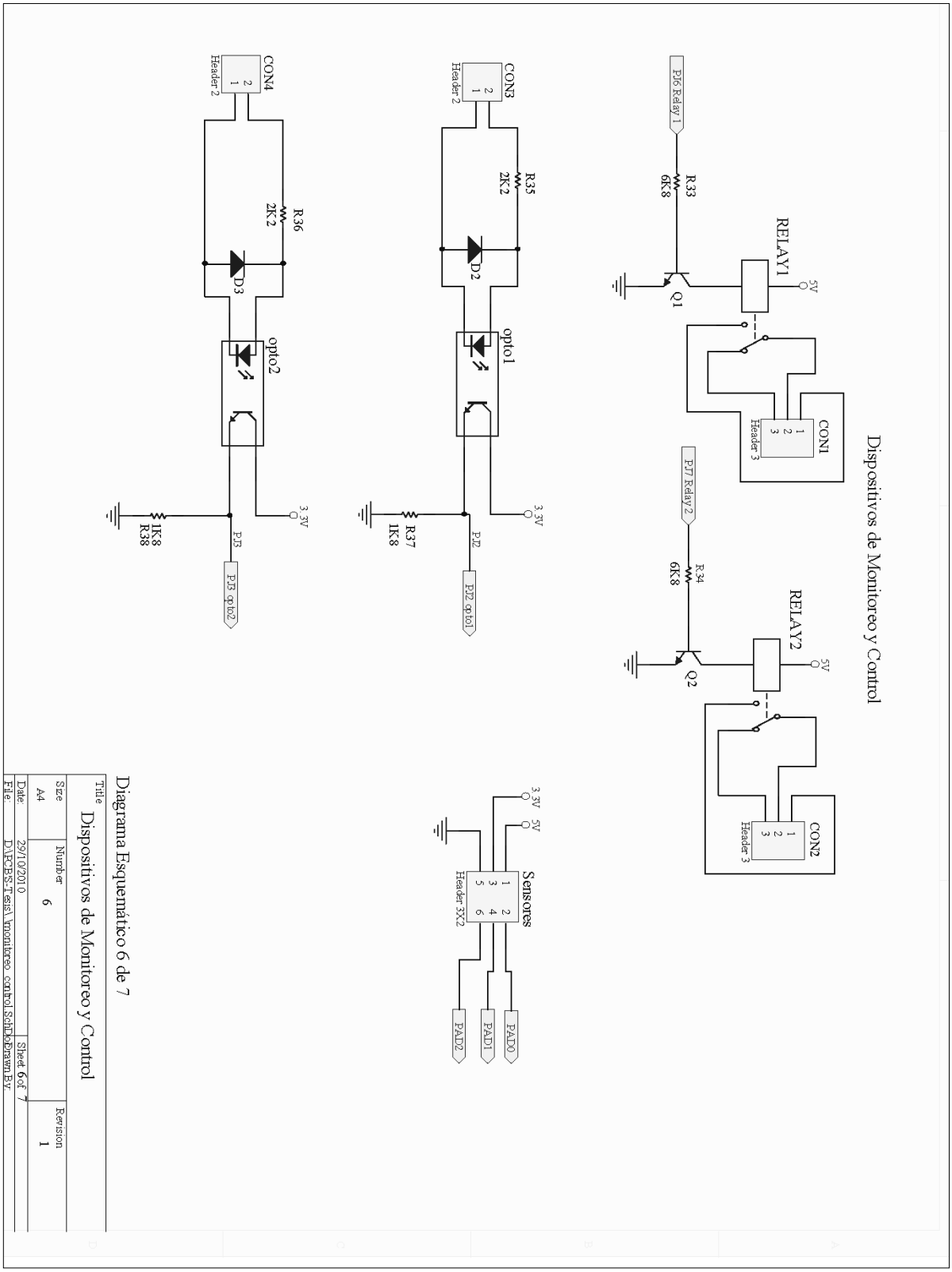


Diagrama Esquemático 4 de 7

Title	Interfaz USB	
Size: A4	Number: 4	Revision: 1
Date: 29/10/2010	Sheet 4 of 7	
File: D:\PCBS-Tesis\conversidorUSB.SCHDoc	Drawn By:	


Dispositivos de Entrada y Salida
Puerto K Botones y Leds
Puerto H Display
Puerto G Teclado
3.3V
5V
B1
B2
B3
B4
PK0
PK1
PK2
PK3
PK[0..7]
R25 1K
R26 1K
R27 1K
R28 1K
R29 330
R30 330
R31 330
R32 330
LED8
LED9
LED10
LED11
PH[0..6]
PH6
PH4
PH2
PH0
PH1
PH3
PH5
DISPLAY
Header 5X2
PG[0..7]
PG6
PG4
PG2
PG0
PG1
PG3
PG5
PG7
TECLADO
Diagrama Esquemático 5 de 7
Title
Dispositivos de Entrada y Salida
Size
A4
Number
5
Revision
1
Date
29/10/2010
Sheet 5 of 7
File
D:\PCBS-Tesis\dispositivos e s.SchDoc
Drawn By


Dispositivos de Monitoreo y Control
RELAY1
RELAY2
CON1
CON2
CON3
CON4
Header 3
Header 2
Header 3X2
R33
6K8
R34
6K8
R35
2K2
R36
2K2
R37
1K8
R38
1K8
Q1
Q2
D2
D3
opto1
opto2
5V
3.3V
P16 Relay 1
P17 Relay 2
P17 opto1
P13 opto2
Sensores
PAD0
PAD1
PAD2
Diagrama Esquemático 6 de 7
Title
Dispositivos de Monitoreo y Control
Size
A4
Number
6
Revision
1
Date
29/10/2010
Sheet 6 of 7
File
D:\PCBS-Tesis\monitoreo_control.SchDoc
Drawn By

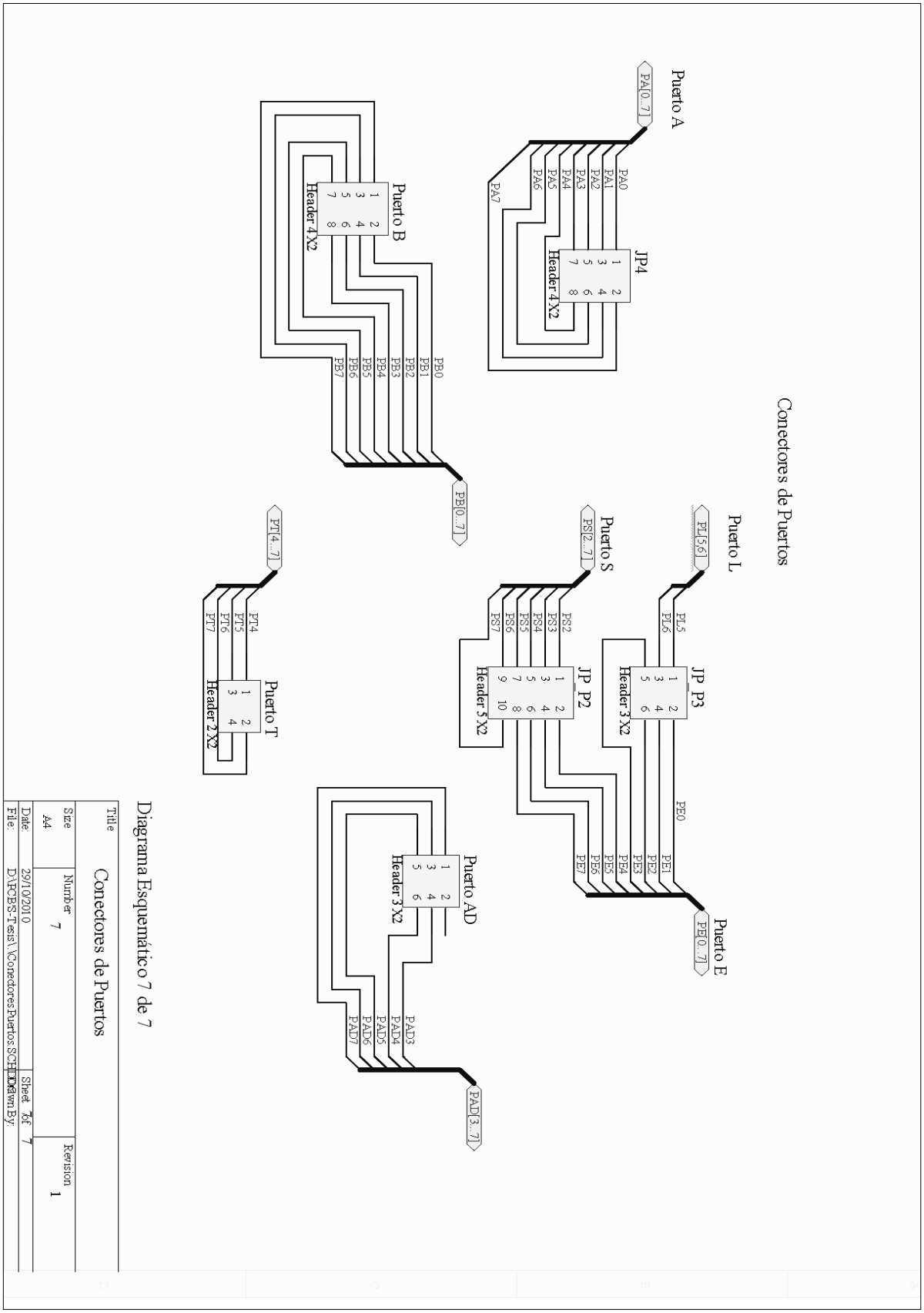

Conectores de Puertos
Puerto A
PA[0..7]
JP4
Header 4 X2
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
Puerto B
Header 4 X2
PB[0..7]
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
Puerto L
PL[5,6]
JP_P3
Header 3 X2
PL5
PL6
Puerto S
PS[2..7]
JP_P2
Header 5 X2
PS2
PS3
PS4
PS5
PS6
PS7
Puerto E
PE[0..7]
PE0
PE1
PE2
PE3
PE4
PE5
PE6
PE7
Puerto T
PT[4..7]
Header 2 X2
PT4
PT5
PT6
PT7
Puerto AD
PAD[3..7]
Header 3 X2
PAD3
PAD4
PAD5
PAD6
PAD7
Diagrama Esquemático 7 de 7
Title
Conectores de Puertos
Size
A4
Number
7
Revision
1
Date:
29/10/2010
Sheet
7
of
7
File:
D:\PCB'S-Tesis\Conectores Puertos.SCHDoc
Drawn By:

Tabla D-1 Listado de componentes utilizados en la tarjeta de desarrollo, costos sin IVA

CIRCUITO	CANT	IDENTIFICADOR	DESCRIPCIÓN/ VALOR	PROVEEDOR	PRECIO UNITARIO	PRECIO TOTAL
MICROCONTROLADOR	1	MC9S12NE64	Microcontrolador	Newark Núm: 70K3320 17dls	204.00	204.00
SELECTOR USUARIO/MONITOR	1	U/M	HEADER 3X1	AG Electronica	2.60	2.60
RESET	1	SW1	PUSH BUTON	AG Electronica	1.70	1.70
	1	R1	47 Ω 1/2 w	AG Electronica	0.90	0.90
	1	R2	27 KΩ 1w	AG Electronica	1.70	1.70
	1	C1	0.22 µF/50v ceramico	AG Electronica	2.60	2.60
FILTRO PLL	1	R3	2.2 KΩ 1 w	AG Electronica	1.70	1.70
	1	C2	470 pF/ 1000v ceramico	AG Electronica	2.60	2.60
	1	C3	4700 pF /500v ceramico	AG Electronica	2.60	2.60
	1	C4	0.1 µF / 50v ceramico	AG Electronica	2.60	2.60
CRISTAL	1	Y1	25 Mhz Cristal de cuarzo mini	Newark Núm:59K8191 / 1.25 dls	15.00	15.00
	1	R4	10 MΩ 1w	AG Electronica	1.70	1.70
	2	C5, C6	22 pF /1000v ceramico	AG Electronica	2.60	5.20
CONECTOR BDM	1	BDM	HEADER 3X2	AG Electronica	5.20	5.20
	1	R5	10 kΩ 1w	AG Electronica	1.70	1.70
CONECTOR DE RED LAN	1	RJ45	CONECTOR RJ45	Newark Núm: 08P2970 13.dls	156.00	156.00
	4	R6, R7, R8, R9	50 Ω 1/2 w	Núm. Newark: 08P2970	0.90	3.60
	1	C7	0.01 uF /50 v ceramico	Núm. Fabricante: 7499011121	2.60	2.60
	5	R10, R11, R12, R13, R14	330 Ω 1/2 w	AG Electronica	0.90	4.50
	1	R15	12.4 kΩ 1w	AG Electronica	1.30	1.30
	5	LED1, LED2, LED3, LED4, LED5	LED verde 5 mm	AG Electronica	1.70	8.50
CAPACITORES DE FILTRO DE ALIMENTACIÓN	6	C8, C9, C10, C11, C12, C13	0.22 µF /50v ceramico	AG Electronica	2.60	15.60
ALIMENTACIÓN	1	power	conector para eliminador	AG Electronica	10.00	10.00
	1	U1	L7805 Regulador 5v/1A	AG Electronica	6.00	6.00
	1	U2	LF33CV Regulador 3.3 v/500mA	AG Electronica	17.00	17.00

	1	C14	0.22 µF/50v ceramico	AG Electronica	2.60	2.60
	1	C15	0.01 uF /50 v ceramico	AG Electronica	2.60	2.60
	2	R16, R17	330 Ω 1/2 w	AG Electronica	0.90	1.80
	2	LED6, LED7	Led rojo 3mm	AG Electronica	1.70	3.40
	2	C16, C19	0.22 µF/50v ceramico	AG Electronica	2.60	5.20
	1	C17	22 µF /50 v electrolitico	AG Electronica	3.50	3.50
	1	C18	100 µF /25v electrolitico	AG Electronica	2.60	2.60
	1	C20	0.01 uF /50 v ceramico	AG Electronica	2.60	2.60
	1	D1	1N4004 Rectificador 1A/400v	AG Electronica	0.90	0.90
	1	selector	HEADER 5x1	AG Electronica	2.60	2.60
	4	5V, 3.3V, GND1, GND2	HEADER 8X1	AG Electronica	2.60	10.40
CONVERTIDOR USB	1	FT232	FT232BL Convertidor usb-serie	AG Electronica	112.00	112.00
	2	C21,C22	47µF /50v electrolitico	AG Electronica	2.60	5.20
	2	C23, C26	0.1 µF / 50v ceramico	AG Electronica	2.60	5.20
	1	L1	150 mH	Newark Num: 04P1893 0.75dls	9.00	9.00
	4	R18, R19, R21, R22	100 Ω 1/2w	AG Electronica	0.90	3.60
	1	R20	1.5 kΩ 1/2w	AG Electronica	0.90	0.90
	1	R23	27 KΩ 1w	AG Electronica	1.70	1.70
	1	Y2	6 MHz Cristal de cuarzo mini	AG Electronica	7.00	7.00
	2	C24, C25	27 pF /50 v ceramico	AG Electronica	2.60	5.20
	1	USB	Conector Usb tipo B recto	AG Electronica	10.00	10.00
	1	R24	470 Ω 1/2w	AG Electronica	0.90	0.90
DISPOSITIVOS E/S	4	B1, B2, B3, B4	PUSH BUTON	AG Electronica	1.70	6.80
	4	R25, R26, R27, R28	1 KΩ 1/2 w	AG Electronica	0.90	3.60
	4	R29, R30, R31, R32	330 Ω 1/2w	AG Electronica	0.90	3.60
	2	DISPLAY, TECLADO	HEADER 5x2	AG Electronica	5.20	10.40
	1	Buzzer	Buzzer piezoelectrico 5mA	AG Electronica	20.00	20.00

	4	LED8, LED9, LED10, LED 11	Led Rojo 5mm	AG Electronica	1.70	6.80
DISPOSITIVOS DE MONITOREO Y CONTROL	2	R33, R34	6.8 kΩ 1/2w	AG Electronica	1.70	3.40
	2	Q1, Q2	BC547 B transistor NPN 50v 0.2 A	AG Electronica	2.60	5.20
	2	RELAY1, RELAY2	Relevador 10A/5v	AG Electronica	17.00	34.00
	2	CON3, CON4	Bornera 2 terminales	AG Electronica	5.20	10.40
	2	CON1, CON2	Bornera 3 terminales	AG Electronica	5.20	10.40
	2	D2, D3	1N4148 D. de conmutacion75mA	AG Electronica	0.90	1.80
	2	Bases Optos	Bases de 6 Pines para C.I.	AG Electronica	2.60	5.20
	2	opto1, opto2	4N25 optoacoplador de salida	AG Electronica	4.30	8.60
	2	R35, R36	2.2 KΩ 1 w	AG Electronica	1.70	3.40
	2	R37, R38	1.8 kΩ 1/2w	AG Electronica	0.90	1.80
	1	sensores	HEADER 3x2	AG Electronica	5.20	5.20
						816.40

*En los costos que contemplan dólares se considera un tipo de cambio $12.00 por dolar

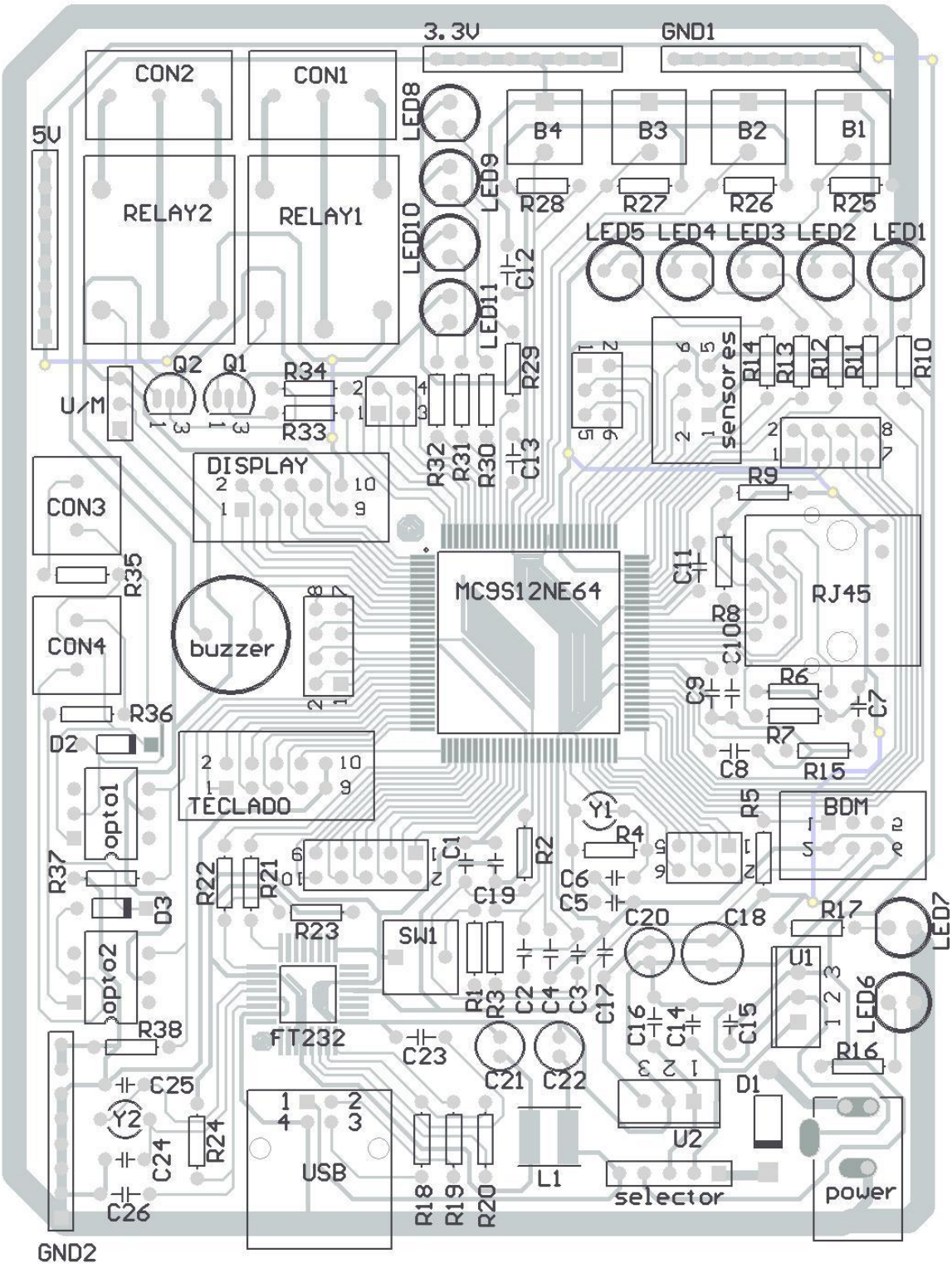


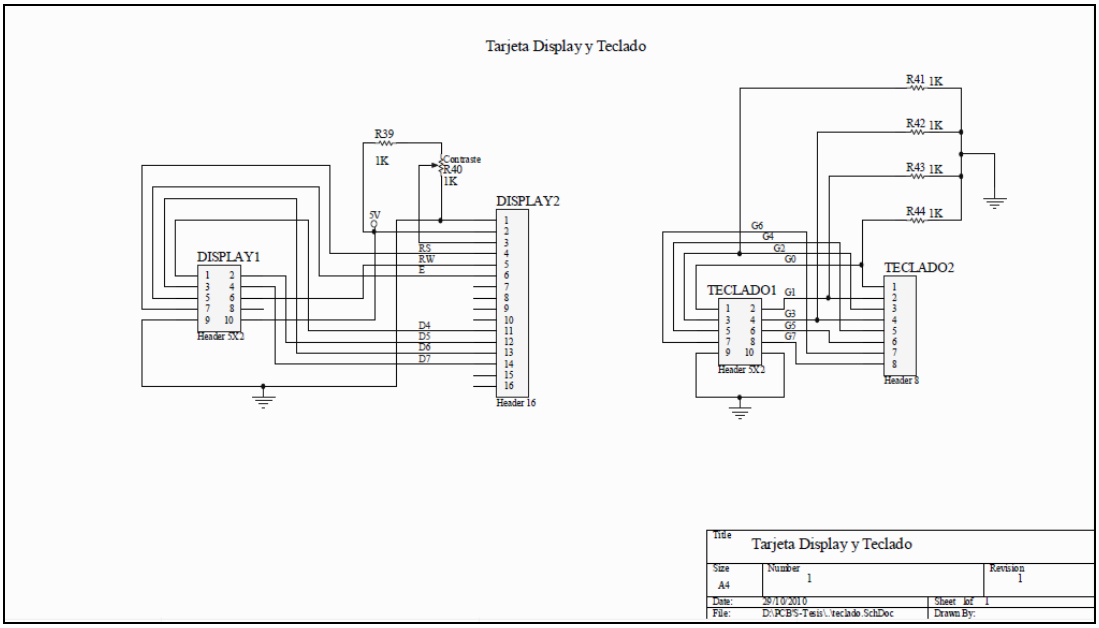
Figura D.1 PCB de la tarjeta de desarrollo con los nombres de los componentes

Tarjetas externas

A continuación se muestran los diseños esquemáticos y circuitos impresos de las tarjetas externas construidas para el display, teclado y sensores que demuestran el correcto funcionamiento de la tarjeta de desarrollo implementada a lo largo del capítulo3.

El display y el teclado se conectan en una tarjeta externa por medio de los conectores de Display y Teclado que se encuentran en la tarjeta de desarrollo y los conectores DISPLAY1 y TECLADO1 que se encuentran en la tarjeta externa. Los conectores DISPLAY2 y TECLADO2 sirven para conectar el display 16x2 y el teclado matricial de 16 teclas, respectivamente

Los tres sensores se conectan en una tarjeta externa por medio del conector *sensores* que se encuentra en la tarjeta de desarrollo y el conector *sensores1* que se encuentra en la tarjeta externa



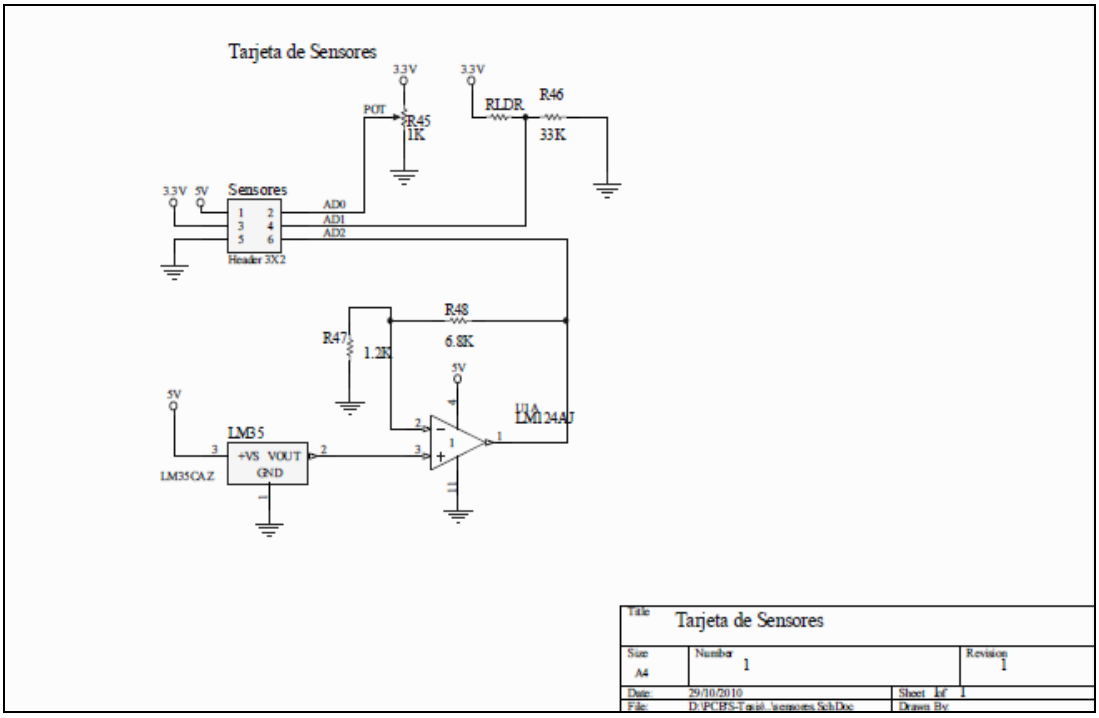


Tabla D-2 Listado de componentes de tarjetas externas, costos sin IVA

TARJETAS EXTERNAS						
TARJETA DISPLAY Y TECLADO	1	R39	40 kΩ 1w	AG Electronica	1.70	1.70
	1	R40	PRESET 10 kΩ	AG Electronica	12.00	12.00
	1	DISPLAY1	HEADER 5x2	AG Electronica	5.20	5.20
	1	DISPLAY2	HEADER 16x1	AG Electronica	2.60	2.60
	4	R41, R42, R43, R44	1 kΩ 1/2 w	AG Electronica	0.90	3.60
	1	TECLADO1	HEADER 5x2	AG Electronica	5.20	5.20
	1	TECLADO2	HEADER 8x1	AG Electronica	2.60	2.60
TARJETA SENSORES	1	R45	PRESET 1 kΩ	AG Electronica	8.60	8.60
	1	RLDR	Fotorresistencia LDR	AG Electronica	5.00	5.00
	1	R46	33 kΩ 1w	AG Electronica	1.70	1.70
	1	sensores1	HEADER 3x2	AG Electronica	5.20	5.20
	1	LM35	LM35 Sensor de Temperatura	AG Electronica	42.00	42.00
	1	R47	1.2 kΩ 1/2w	AG Electronica	0.90	0.90
	1	R48	6.8 kΩ 1/2w	AG Electronica	0.90	0.90
	1	U1A	LM124 A.O. baja frecuencia	AG Electronica	28.50	28.50
						125.70

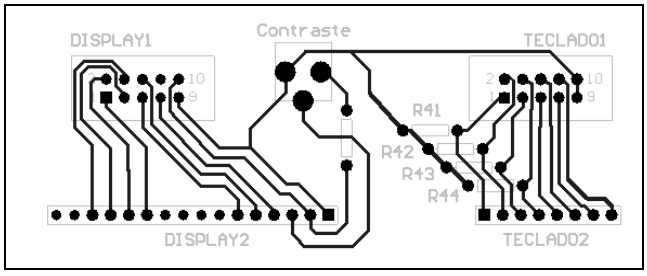


Figura D.2 PCB del circuito impreso del display

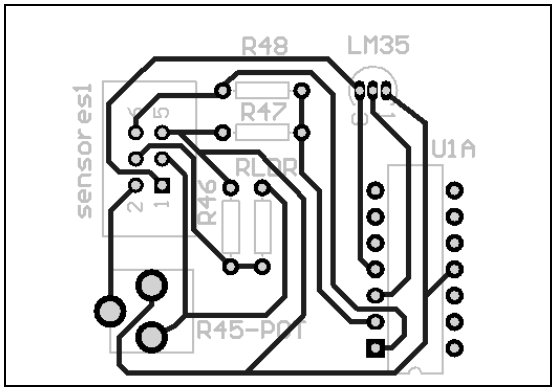


Figura D.3 PCB del circuito impreso de los sensores

Apéndice E
Desarrollo de una aplicación paso a paso (Con el Sistema de desarrollo)

El propósito de este apéndice es crear una aplicación paso a paso para que un usuario que utilice el sistema de desarrollo que se ha descrito en los capítulos de esta tesis pueda programar la aplicación web que requiera, empleando las funciones de la pila TCP/IP.

Como se explicó en el capítulo 7, la aplicación se compone de dos partes: la aplicación del lado servidor y la aplicación del lado cliente, cada una con su propia lógica pero con reglas que ambas deben seguir para poder realizar la comunicación.

Aplicación del lado servidor

El lado servidor debe programarse en lenguaje C, y ocupar para ello el IDE codeWarrior. En los siguientes temas se explicará la forma de realizar una aplicación del lado servidor mostrando los pasos a seguir, archivos que se ocupan, su ubicación dentro del proyecto y la estructura de las carpetas del proyecto.

Ubicación de las carpetas y archivos del proyecto

Crear la aplicación del lado servidor requiere conocer la estructura del proyecto en codeWarrior para el sistema desarrollado. La carpeta *pilaTcpIpFilink* contiene el proyecto de codeWarrior con la pila TCP/IP. Su contenido completo se muestra en la Figura E.1.

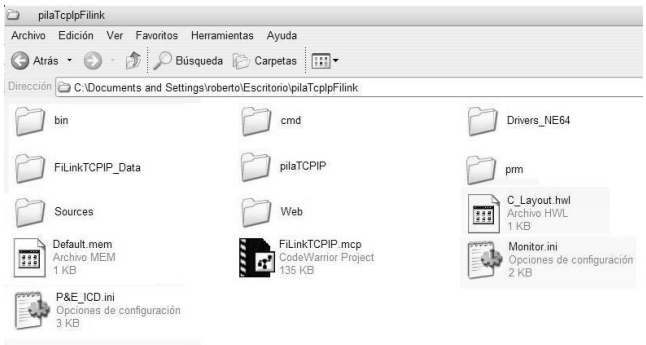


Figura E.1 Carpetas del proyecto *pilaTcpIpFilink* en *codeWarrior*

La carpeta *Drivers_NE64* contiene las funciones del controlador de red que proporciona *Freescale*. En la carpeta *pilaTCPIP* se encuentran las funciones que implementan los protocolos de la pila TCP/IP descrita en el capítulo 5. Su contenido se muestra en la Figura E.2.

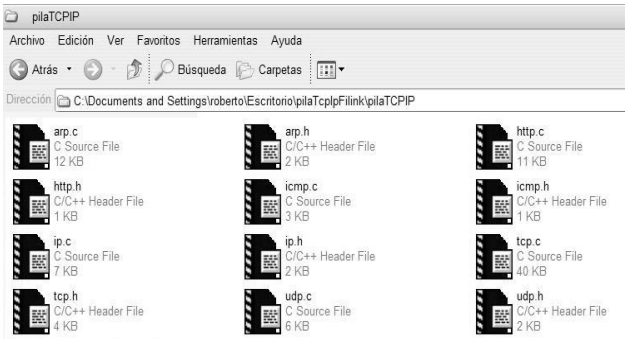


Figura E.2 Carpeta con los protocolos de la pila TCP/IP

En la carpeta *Sources* se encuentran las funciones que manejan los diferentes periféricos del microcontrolador y dispositivos conectados a él, como el módulo ADC, el módulo SCI, los *timers* y el display. En esta carpeta también se encuentra el archivo *main.c* que contiene la función principal del proyecto.

En la carpeta *Web* podemos encontrar los archivos con los que se implementa el servidor web, como los archivos fuente de las páginas e imágenes.

Cada vez que se requiera realizar un proyecto diferente es conveniente trabajar sobre una copia de la carpeta del proyecto *pilaTcpIpFilink* para iniciar el desarrollo. Si se decide crear un nuevo proyecto primero se deben agregar al proyecto los archivos de la pila y las funciones de los periféricos que se utilicen.

Planteamiento y especificaciones del ejemplo

En esta etapa del desarrollo se deben especificar los puertos para la comunicación así como el protocolo de transporte, los comandos que recibirá el servidor y de forma clara lo que debe realizar la aplicación.

En este caso se programará una aplicación que cambie el estado de los leds del puerto K de la tarjeta de desarrollo, explicando paso a paso la programación y el manejo de los archivos del proyecto.

La forma más práctica de implementar la comunicación es mediante el protocolo UDP para el envío y recepción de datos, debido a que es más rápido. Los comandos enviados por el cliente se recibirán en el puerto 2003 del servidor. La codificación de cada comando debe ser enviada desde el lado cliente al puerto indicado (ver Tabla E-1).

Tabla E-1. Puertos que utilizan la aplicación y sus comandos

Sección de la aplicación	Puerto del servidor	Comando	Código	Descripción del comando
Cambiar el estado de los leds de la tarjeta	2003	1	0001	Cambiar el estado del led del bit 4 del puerto K
		2	0010	Cambiar el estado del led del bit 5 del puerto K
		3	0011	Cambiar el estado del led del bit 6 del puerto K
		4	0100	Cambiar el estado del led del bit 7 del puerto K

Archivos involucrados

El archivo *pilaTcpIpFilink.mcp* es el proyecto que debe abrirse con el IDE codeWarrior. Una vez abierto, del lado izquierdo aparece la pestaña *Files* con la estructura del proyecto.

Para realizar esta aplicación se trabajará principalmente con dos archivos, el primero es el archivo *main.c* que contiene la función principal del precyeto y el segundo es un archivo en donde se implementa la aplicación de usuario, para este ejemplo será *udp_user_application.* Para crear el segundo archivo e incluirlo en el proyecto se deben realizar los siguientes pasos:

- Ingresar a la carpeta Sources (aunque puede ser en cualquier otra) y crear un nuevo archivo de texto con el nombre *udp_user_application.c.* La extensión del archivo creado debe cambiarse de .txt a .c.
- Con el proyecto de codeWarrior abierto, se debe hacer click derecho en alguna carpeta de la estructura que aparece en la pestaña *Files.*
- Seleccionar la opción *Add files* y buscar el archivo creado en la ruta en la que se haya guardado (ver Figura E.3).

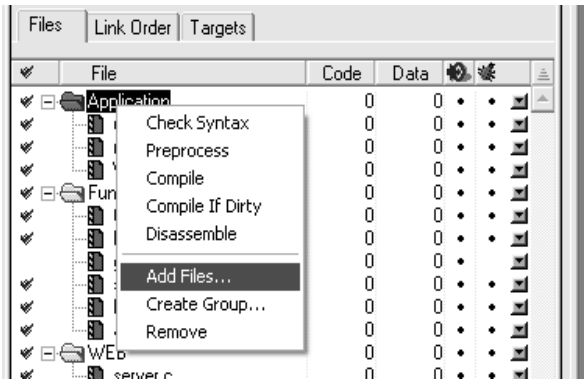


Figura E.3 Agregar un archivo al proyecto

Una vez creado el archivo, en la estructura del proyecto pueden identificarse tanto el archivo *main.c* como el *udp_user_application.c* (ver Figura E.4).

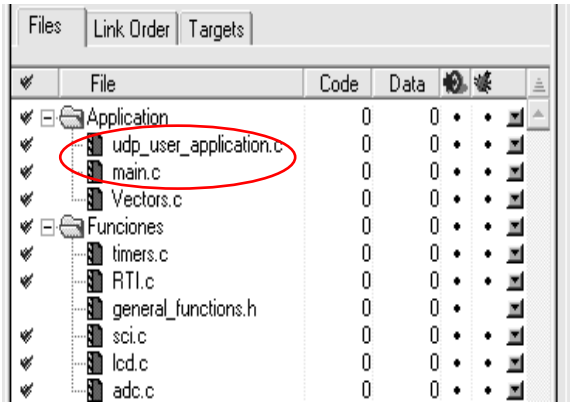


Figura E.4 Archivos main.c y udp_user_application.c

Para el desarrollo del programa se debe conocer la estructura y contenido del archivo *main.c*, al cual se le realizan pocos cambios para esta aplicación. El archivo *main.c* tiene las secciones siguientes:

- **Inclusión de los archivos.** En esta sección se incluyen todos los archivos que requiere la función principal para su funcionamiento, para el caso de esta aplicación paso a paso, no es necesario incluir el protocolo TCP. Si se crean nuevos archivos con funciones que se llamen desde la función principal será necesario incluirlos en esta sección. Los archivos que no se utilicen se pueden comentar o eliminar de esta parte del archivo (ver Figura E.5).

```
/*INCLUSION DE ARCHIVOS*/
#include <hidef.h>
#include <MC9S12NE64.h>
#pragma LINK_INFO DERIVATIVE "mc9s12ne64"
#include "timers.h"
#include "general_functions.h"
#include "ne64driver.h"
#include “address.h”
#include "arp.h"
#include "mBuf.h"
#include "ne64api.h"
#include "ip.h"
/*#include "tcp.h"*/
#include "udp.h"
#include "http.h"
#include "server.h"
```

Figura E.5 Inclusión de archivos

- **Prototipo de las funciones.** Los prototipos de funciones que utiliza la función principal se deben indicar antes de ésta para que se puedan ejecutar, así como la declaración de las variables globales o

externas necesarias. Las variables externas se declaran con la palabra reservada "extern", que indica que la variable ya está declarada en otro archivo que se ha incluido y que dicha variable es la que se usará, no se está declarando una nueva variable (ver Figura E.6).

```
/*PROTOTIPOS DE FUNCIONES*/
void udp_app_server(void);
// buffer para almacenar los datos de los niveles superiores
extern unsigned char data_buff[DATA_BUFF_LEN];
        unsigned char ethernet_receive(void);

extern TEthernetFrame received_frame;
```

Figura E.6 Prototipos de funciones

- **La función principal**. Inicializa los servicios del servidor web y espera a que se realice una petición de algún recurso o alguna acción que debe realizar el microcontrolador. La inicialización incluye los servicios de HTTP, ARP, sockets UDP y TCP, función que inicializa la aplicación de usuario, entre otros servicios.

  En el ciclo infinito (for) se verifica si llega un paquete Ethernet, en cuyo caso se recibe y se procesa, y posteriormente se ejecutan las funciones arp_manage(), tcp_poll() y http_run(), que son las rutinas periódicas de los protocolos ARP, TCP y HTTP, respectivamente (ver Figura E.7).

```
//inicio de main
void main(void) {
    INTCR_IRQEN = 0;
    setPLL(1,1); sci_init(); EnableInterrupts;
    /* inicialización de los servicios */
    RTI_init();
    mBufInit ();
    tcp_socket_init();
    timers_init();
    arp_init();
    udp_socket_init();
    EtherInit();
    udp_app_server();
    http_init_server();
    for(;;){
        if(NE64ValidFrameReception()){
          //se recibe el paquete ethernet
           ethernet_receive();
        }
        arp_manage();
        tcp_poll();
        http_run();
    }//fin for
} //fin MAIN
```

Figura E.7 Función principal del servidor

- **La función ethernet_receive.** Procesa el paquete Ethernet cuando éste se recibe. Se llama dentro del ciclo infinito de la función principal (ver Figura E.8).

```
unsigned char ethernet_receive(void){
  unsigned char i;
  //validación de los datos del frame ethernet: el paquete es para mi?
  for(i=0;i<ETH_ADDRS_LEN;i++)
      if(received_frame.destination[i]!=hard_addr[i] && received_frame.destination[i]!= 0xFF){
          NE64FreeReceiveBuffer(); //liberar el buffer de recepción
          return; // el paquete no es para mi
      }
  if(received_frame.protocol==ARP_ETYPE){
      write_string("Paquete ARP recibido...");
      arp_receive(received_frame.buf_index);
  }else if(received_frame.protocol==IP_ETYPE){
      ip_receive(received_frame.buf_index);
  }else{
      NE64FreeReceiveBuffer(); //liberar el buffer de recepción
      return; // protocolo desconocido
  }
  NE64FreeReceiveBuffer(); //liberar el buffer de recepción
```

```
    return ;   // todo bien, el paquete se procesó
}
```

Figura E.8 Función de recepción de paquetes Eethernet

En general el archivo de la aplicación *udp_user_application.c,* debe contener:

- Inclusión de archivos de periféricos u otras funciones que el programa requiera.
- Definiciones del usuario en caso de ser necesarias.
- Prototipo de las funciones programadas en ese archivo o en los archivos agregados.
- La función para abrir un socket para recibir y enviar los datos.
- La función que realice las acciones que indique el cliente y regrese datos si es necesario.

Desarrollo del programa

Para iniciar el programa se debe abrir el archivo *udp_user_application.c* haciendo doble click sobre él. La programación en este archivo se realizará en cinco pasos:

Paso 1 de 5. Inclusión de archivos. Las líneas de código que se presentan en la Figura E.9 indican los archivos que requiere esta aplicación para su funcionamiento.

```
//SECCION ARCHIVOS A INCLUIR
#include "udp.h"
#include "ne64api.h"
#include <MC9S12NE64.h>
#include "general_functions.h"
#include <hidef.h>
#pragma LINK_INFO DERIVATIVE "mc9s12ne64"
extern struct udp_sock  udp_socket[UDP_SOCKETS_NUM];
```

Figura E.9 Archivos que incluye la aplicación de ejemplo

Paso 2 de 5. Definiciones del usuario. En la Figura E.10 se presenta la definición del puerto usado por el servidor para enviar y recibir datos, esta definición se utiliza en la función de inicialización de la aplicación de usuario.

```
//SECCION  DEFINICIONES CONFIGURABLES POR EL USUARIO
#define UDP_LED_PORT  2003
```

Figura E.10 Definición del valor del puerto que usa la aplicación

Paso 3 de 5. Prototipo de las funciones programadas. Los prototipos de las funciones utilizadas dentro de este archivo se deben indicar al inicio del mismo (ver Figura E.11).

```
//SECCION PROTOTIPOS DE FUNCIONES ->agregar Si se crea una nueva función
void udp_led_app(char, unsigned int, unsigned int);
```

Figura E.11 Prototipo de la función que realiza el cambio de estado de los leds

Paso 4 de 5. Programación de la función que inicializa la aplicación de ususario. En esta función se abre un socket UDP tipo servidor para que pueda realizarse la comunicación, se asocia a la función que realizará el cambio de estado de los leds, al puerto por el que se recibirán los datos y se configura la parte alta del puerto K como salidas (ver Figura E.12).

```
//FUNCION PARA OBTENER UN SOCKET PARA LA APLICACION
void udp_app_server(){
  char socket1;
  socket1 = udp_get_sock();//obtener un socket para la comunicación
  if(socket1==-1)
     return;//No se puede iniciar el servicio porque no hay sockets disponibles
  // la siguiente línea: abrir el socket como aplicación servidor y se asocia a la función del
 //cambio de estado de los leds
  udp_open_sock(socket1,UDP_SERVER, 0, 0, UDP_LED_PORT, udp_led_app);
  DDRK |= 0xF0;  // las terminales de la parte alta del puerto K como salida
}
```

Figura E.12 Función que abre el socket para la aplicación

Paso 5 de 5. Función que realiza el cambio de estado del led indicado. En esta función se lee el comando enviado desde el cliente y a partir de su valor se cambia el estado del led indicado, posteriormente se lee el

valor de cada bit y se envía como respuesta al cliente para poderse presentar en la página el estado de cada led (ver Figura E.13).

```
//FUNCION QUE REALIZA EL CAMBIO DE ESTADO DE LOS LEDS
void udp_led_app(char socket, unsigned int app_data_buff, unsigned int datalen){
    unsigned char command; char i;
    i=UDP_HEADER_OFFSET;// en la siguiente linea: se validan datos recibidos
    if(app_data_buff<HEADERS_LEN || datalen==0  || udp_socket[socket].state != UDP_OPEN)
        return;  //datos no validos o socket ocupado
    NE64InitializeOffsetToReadRxBuffer(app_data_buff); //se inicializa el buffer para leer datos
    command = NE64ReadByte();//se lee el comando enviado desde el cliente
    if(command=='1'){//en este IF se cambia el valor del led que se haya indicado en el comando
        PORTK_BIT4=~PORTK_BIT4;
    }else if(command=='2'){
        PORTK_BIT5=~PORTK_BIT5;
    }else if(command=='3'){
        PORTK_BIT6=~PORTK_BIT6;
    }else if(command=='4'){
        PORTK_BIT7=~PORTK_BIT7;
    }else return; // si no es ningún comando  no se realiza ninguna acción en los leds
    data_buff[i++]=PORTK_BIT4;
    data_buff[i++]=PORTK_BIT5;
    data_buff[i++]=PORTK_BIT6;
    data_buff[i++]=PORTK_BIT7;
    udp_send(socket,data_buff,4);//enviar a través de UDP
}
```

Figura E.13 Función que realiza el cambio de estado de los leds

De esta forma ya se tiene programado el lado servidor de la aplicación. El cual responde a los comandos enviados desde el cliente.

Aplicación del lado cliente

La aplicación del lado cliente consiste, en este caso, en el envío de los comandos al servidor y la recepción de los datos leídos correspondientes al estado de los leds. Este envío se realiza por medio de botones que indican qué led se debe cambiar de estado.

Para la programación del lado cliente se utiliza el lenguaje PHP, el cual debe estar instalado en la computadora desde la cual se realiza el envío de los comandos o en otro servidor al que se tenga acceso para subir los archivos de la aplicación cliente. Para la instalación en sistemas operativos Windows existen paquetes de instalación como XAMPP y WAMP, que se descargan de www.apachefriends.org/es/xampp.html y de www.wampserver.com, respectivamente, o algún otro paquete que el usuario ya conozca.

Habilitar manejo de sockets en PHP

La aplicación utiliza los sockets para enviar información al servidor, por este motivo los sockets deben estar habilitados, en caso de no estarlo (al intentar usarlos, PHP enviará un error indicando que no se reconocen las funciones utilizadas de manejo de sockets) se deben habilitar editando el archivo *php.ini* de la instalación del programa, eliminando el carácter ";" (punto y coma) de la línea que aparece en la Figura E.14.

```
;extension=php_sockets.dll
```

Figura E.14 Habilitar el módulo para manejo de sockets en PHP

Funciones de manejo de sockets

La aplicación cliente requiere enviar y recibir datos a través de un puerto, para tal propósito PHP tiene funciones que manejan el envío y recepción de datos por medio de sockets. Las funciones que se utilizan en esta aplicación se presentan en la Tabla E-2.

Tabla E-2. Funciones de PHP para manejo de sockets en esta aplicación

Función	Descripción
socket_create	Crea un socket
socket_sendto	Envía un mensaje a un socket que puede estar conectado o no (TCP o UDP)
socket_read	Lee los datos recibidos por un socket previamente abierto
socket_close	Cierra un socket

Estructura del programa en PHP y página HTML

La aplicación se puede codificar en algún editor de HTML o en block de notas de Windows. El programa se debe escribir en un archivo con extensión ".php", que en este caso es leds.php, el cual incluye el código de PHP y código HTML para la presentación de los datos.

Desarrollo del programa

El programa se ha dividido en 5 pasos que se explican a continuación:

Paso 1 de 5. Identificar el botón presionado y asignar el comando. En la Figura E.15 se presenta el código que identifica que botón fue presionado y asigna el comando a enviar en una variable.

```
<?php //inicia el programa con PHP
//se revisa el identificador del botón que se presionó, se reciben por el método POST
if(isset($_POST['led4'])){//para el botón asociado al led 4 se envía el comando 1
      $dataToSend=0x01;
}elseif(isset($_POST['led5'])){ //para el botón asociado al led 5 se envía el comando 2
      $dataToSend=0x02;
}elseif(isset($_POST['led6'])){     //para el botón asociado al led 6 se envía el comando 3
      $dataToSend=0x03;
}elseif(isset($_POST['led7'])){ //para el botón asociado al led 7 se envía el comando 4
      $dataToSend=0x04;
}
```

Figura E.15 Parte del programa en PHP que identifica el botón presionado

Paso 2 de 5. Abrir el socket, enviar los datos y esperar respuesta. Para enviar el comando al servidor se debe abrir un socket que enviará y recibirá por UDP los datos, en la Figura E.16 se presenta el código que abre el socket, envía los datos y espera una respuesta del servidor.

```
//se crea el socket para envío de datos por UDP
$sock = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
//enviar comando cambiar estado de led asignado en el paso 1 a la variable
//$dataToSend por el socket $sock a la IP del servidor y el puerto 2003
socket_sendto($sock,$dataToSend, 1, 0, '192.168.2.3', 2003);
//se lee la respuesta del servidor y se asigna a la variable $received
$received =socket_read($sock,1024);
```

Figura E.16 Obtención del socket, envío de los datos y recepción de la respuesta

Paso 3 de 5. Convertir los datos recibidos. Ya que se han recibido los datos desde el servidor y se tienen almacenados en una variable, éstos se presentaran como valor numérico. En la Figura E.17 se muestra el código que lee cada byte recibido y lo convierte en el carácter que corresponde, cada carácter se guarda en una localidad de un arreglo.

```
$hex= array();//se delcara una variable como arreglo
   foreach (str_split($received) as $chr) //para cada dato recibido
      //se convierte en el carácter hexadecimal que representa uno ó cero,
      //que son los posibles valores y se agrega al arreglo
      $hex[] = sprintf("%02X", ord($chr));
```

Figura E.17 Convertir los datos recibidos para poderse interpretar según su valor ASCII

Paso 4 de 5. Asignar los valores a presentar en la página HTML. Para cada valor que se ha guardado en el arreglo se tienen dos posibles valores que representan el estado de los leds: cero para apagado y uno para

encendido. En esta parte se lee cada valor del arreglo y se asigna a una variable el valor que representa el color verde para encendido o rojo para apagado (ver Figura E.18).

```
if($hex[0]==0){//si el valor del bit 4 del PTK es 0 se asigna el color rojo a la variable
    $colorLed4="red";
}elseif($hex[0]==1){ //si el valor es 1 se asigna el color verde
    $colorLed4="green";
}
if($hex[1]==0){ //si el valor del bit 5 del PTK es 0 se asigna el color rojo a la variable
    $colorLed5="red";
}elseif($hex[1]==1){ //si el valor es 1 se asigna el color verde
    $colorLed5="green";
}
if($hex[2]==0){ //si el valor del bit 6 del PTK es 0 se asigna el color rojo a la variable
    $colorLed6="red";
}elseif($hex[2]==1){ //si el valor es 1 se asigna el color verde
    $colorLed6="green";
}
if($hex[3]==0){ //si el valor del bit 7 del PTK es 0 se asigna el color rojo a la variable
    $colorLed7="red";
}elseif($hex[3]==1){ //si el valor es 1 se asigna el color verde
    $colorLed7="green";
}
socket_close($sock);//se cierra el socket una vez que se ha dejado de utilizar
//por ultimo se cierra la etiqueta de php para indicar que el programa termina
?>
```

Figura E.18 Asignación del color de fondo para indicar el estado de cada led

Paso 5 de 5. Presentar la respuesta en la página HTML. Para la programación de la presentación de los datos se deben poner las etiquetas básicas de HTML y crear un formulario en el cual se tendrán los botones que al presionarlos enviarán los datos, usando el método POST, a la misma página para que el código de PHP envíe los comandos. Cuando ya se han enviado los datos al servidor y se ha recibido la respuesta, después de convertir los valores, estos son presentados en esta página (ver Figura E.19).

```
<html >
<head><link rel="stylesheet" type="text/css" href="./sigeelstyle.css" />
<meta http-equiv="Content-Type" content="txt/html; charset=utf-8" />
<title>APLICACI&Oacute;N CLIENTE</title>
</head>
<body>
    <form action="leds.php" method="POST">
        <INPUT type="submit" name="led7" class="blue_button" value="LED 1" >
<!—se presenta el color de fondo según se asignó su valor en la lectura de los datos recibidos- ->
        <INPUT type="TEXT" maxlength="10" size="16" readonly style="background-color:<?php echo $colorLed7?>;">
        <INPUT type="submit" name="led6" class="blue_button" value="LED 2" >
<!—se presenta el color de fondo según se asignó su valor en la lectura de los datos recibidos- ->
        <INPUT type="TEXT" maxlength="10" size="16" readonly style="background-color:<?php echo $colorLed7?>;">
        <INPUT type="submit" name="led5" class="blue_button" value="LED 3" >
<!—se presenta el color de fondo según se asignó su valor en la lectura de los datos recibidos- ->
        <INPUT type="TEXT" maxlength="10" size="16" readonly style="background-color:<?php echo $colorLed7?>;">
        <INPUT type="submit" name="led4" class="blue_button" value="LED 4" >
<!—se presenta el color de fondo según se asignó su valor en la lectura de los datos recibidos- ->
        <INPUT type="TEXT" maxlength="10" size="16" readonly style="background-color:<?php echo $colorLed7?>;">
</form>
</body>
</html>
```

Figura E.19 Código HTML para presentar los datos y los botones que envían los comandos

De esta forma la aplicación explicada paso a paso se tiene terminada tanto del lado servidor como del lado cliente.

Para el desarrollo de una aplicación en específico se deben tomar en cuenta los requerimientos de la aplicación, los alcances del sistema de desarrollo, los conocimientos de lenguajes de programación del desarrollador y los recursos e infraestructura con que se cuenta, dichos factores determinan la forma en que se realizará la aplicación.

Apéndice F
Código Fuente de la pila TCP /IP

Como se especifica en el capítulo 5 los protocolos que se desarrollan en esta tesis son:

Capa	Protocolo
Aplicación	HTTP
Transporte	TCP, UDP
Red	IP, ICMP, ARP
Enlace	Ethernet

A continuación se muestra el código fuente de dichos protocolos según se describió en el capítulo 5 (archivos de cabecera *.h y código *.c), además se añade el código del archivo principal (main).

Main Servidor

```
#include <hidef.h>      /* common defines and macros */
#include <MC9S12NE64.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12ne64"

#include "timers.h"
#include "general_functions.h"
#include "ne64driver.h"
#include "arp.h"
#include "mBuf.h"
#include "ne64api.h"
#include “address.h”
#include "ip.h"
#include "tcp.h"
#include "udp.h"
#include "http.h"
#include "server.h"

void udp_app_server(void);

// buffer para almacenar los datos de los niveles superiores
extern unsigned char data_buff[DATA_BUFF_LEN];

unsigned char ethernet_receive(void);
extern TEthernetFrame received_frame;


//inicio de main

_____
_____
void main(void) {

INTCR_IRQEN = 0;

setPLL(1,1);
sci_init();

write_string("Iniciando MAIN...");
write_string("Inicializando servicios...");


EnableInterrupts;

/* inicialización de los servicios */

RTI_init();
mBufInit ();
EtherInit();
timers_init();
arp_init();
udp_socket_init();
tcp_socket_init();
http_init_server();
udp_app_server();
/*_____*/

write_string("Configuracion del sistema completado");

for(;;) {


if(NE64ValidFrameReception()) {
ethernet_receive();
}
arp_manage();
tcp_poll();
http_run();

}//fin for

} /////fin MAIN

unsigned char ethernet_receive(void) {
unsigned char i;

//mostrar ethernet recibido
write_char(0x0D); write_char(0x0A);
write_string("Paquete Ethernet recibido:");
for(i=0;i<6;i++)
write_hx(received_frame.destination[i]);
for(i=0;i<6;i++)
write_hx(received_frame.source[i]);
i=received_frame.protocol>>8;
write_hx(i);
write_hx((char)received_frame.protocol&0x00FF);
write_char(0x0A); write_char(0x0D);

//validación de los datos del frame ethernet

//el paquete es para mi???
for(i=0;i<ETH_ADDRS_LEN;i++)
if(received_frame.destination[i]!=hard_addr[i]                    &&
received_frame.destination[i]!= 0xFF ) {
write_string("Paquete Ethernet con direccion MAC erronea");
NE64FreeReceiveBuffer(); //liberar el buffer de recepción
return; // el paquete no es para mi
}


if(received_frame.protocol==ARP_ETYPE) {
write_string("Paquete ARP recibido...");
arp_receive(received_frame.buf_index);
}
else if(received_frame.protocol==IP_ETYPE) {
write_string("Paquete IP recibido...");
ip_receive(received_frame.buf_index);
}
else {
```

```
write_string("Paquete Ethernet desconocido, no se proceso.");
NE64FreeReceiveBuffer(); //liberar el buffer de recepción
return; // protocolo desconocido
}

NE64FreeReceiveBuffer(); //liberar el buffer de recepción
return ;   // todo bien, el paquete se procesó!!!
}
```

PROTTOCOLO ARP

arp.h

```
struct arp_message{
unsigned int  HWtype;
unsigned int  protocol;
unsigned char HWdirlen;
unsigned char protdirlen;
unsigned int  opcode;
unsigned char MACaddSource[6];
unsigned char IPaddSource[4];
unsigned char MACaddDest[6];
unsigned char IPaddDest[4];
};

struct arp_table{
unsigned char status;
unsigned char type;
unsigned char ttl;
unsigned char retries;
unsigned char MACaddr[6];
unsigned char IPaddr[4];
} ;

#define ARP_ETYPE 0x0806
#define ARP_LEN   28

//valores de la estructura ARP
#define TYPE_ETHERNET  1
#define IP_ETYPE       0x0800
//#define HW_LEN          6   ;ya está definido en api.h como
ETH_ADDRS_LEN
#define IP_DIR_LEN     4
#define ARP_REQUEST    1
#define ARP_REPLY      2


// estados de las entradas ARP
#define ARP_FREE     0
#define ARP_PENDING  1
#define ARP_RESOLVED 2

//tipos de entradas
#define  STATIC_ENTRY  0
#define  DYNAMIC_ENTRY 1

// tabla arp
#define ARP_TABLE_SIZE    10
#define ARP_TABLE_MANAGE  1
#define ARP_RETRIES       5
#define ARP_TTL           60
#define ARP_RESEND        2


//prototipos de funciones
void arp_init(void);
void arp_send(unsigned int, unsigned char *, unsigned char
*);
void arp_add_entry(unsigned char *, unsigned char *);
unsigned char * arp_get_mac(unsigned char *);
void arp_receive(unsigned int);
void arp_add_response(unsigned char *, unsigned char *);
void arp_manage(void);
unsigned char equal_ip(unsigned char *, unsigned char *);
unsigned char same_subnet(unsigned char *);
```

arp.c

```
#include "arp.h"
#include "timers.h"
#include "ne64api.h"
#include "general_functions.h"
#include "address.h"

struct arp_table arp_table_entry[ARP_TABLE_SIZE];
unsigned char arp_timer;
extern TEthernetFrame send_frame;

//////////////////// INICIALIZACIÓN DE LA TABLA
////////////////////
void arp_init(void){
unsigned char i;

for(i=0;i<ARP_TABLE_SIZE;i++)
arp_table_entry[i].status=ARP_FREE;
arp_timer=create_timer(DOWN,ARP_TABLE_MANAGE,SEC);

}

//////////////////// BUSQUEDA DE MAC EN LA TABLA
ARP ///////////////
unsigned char * arp_get_mac(unsigned char *ip){

unsigned char i, resolved_entry = 0, minttl=ARP_TTL+1, type
= DYNAMIC_ENTRY;
char free_entry = -1;

if(equal_ip(ip_gateway,ip))
type = STATIC_ENTRY;

write_string("Buscando MAC en la tabla ARP");
// primero se recorre la tabla para buscar la dirección
for(i=0;i<ARP_TABLE_SIZE;i++){
switch(arp_table_entry[i].status){
case ARP_FREE:
if(free_entry == -1)
free_entry = i;
break;
case ARP_PENDING:
if(equal_ip(arp_table_entry[i].IPaddr,ip))
return (void *)0; //se encontró, pero no está lista aún
else
break;
case ARP_RESOLVED:
if(equal_ip(arp_table_entry[i].IPaddr,ip))
return  arp_table_entry[i].MACaddr;  //se  encontró  y  se
regresa la mac correspondiente
else
resolved_entry++;
break;
} //fin del switch
} // fin del for

// si llegamos hasta aqui significa que la ip aún no está en la
tabla
// si ya no habia entradas libres verificamos las entradas ya
resueltas
// para borrar la mas vieja
if(free_entry==-1 && resolved_entry)
for(i=0;i<ARP_TABLE_SIZE;i++)
if(arp_table_entry[i].status      ==      ARP_RESOLVED      &&
arp_table_entry[i].type == DYNAMIC_ENTRY)
if(arp_table_entry[i].ttl<minttl){
free_entry = i;
minttl = arp_table_entry[i].ttl;
}

if(free_entry!=-1){
arp_table_entry[free_entry].status = ARP_PENDING;
arp_table_entry[free_entry].ttl = ARP_RESEND;
arp_table_entry[free_entry].retries = ARP_RETRIES;
arp_table_entry[free_entry].type =type;
for(i=0;i<IP_DIR_LEN;i++)
arp_table_entry[free_entry].IPaddr[i] = *(ip++);
arp_send(ARP_REQUEST,ip,((unsigned char *)0));
}
```

```
// si no habia entradas resueltas, y tampoco habia entradas libres
// todas estaban esperando una respuesta ARP, por lo tanto no se
// puede borrar ninguna, no hay nada más que hacer, aqui acaba la función

return (void *)0;
}

//////////////////// ENVIO DE MENSAJE ARP ////////////////////////
void arp_send(unsigned int op, unsigned char *ip, unsigned char *mac){
struct arp_message arp_packet;
unsigned char i, rem_mac[6];
int hex;

//verificar si se trata del envío de una solicitid o de una respuesta
if(op == ARP_REQUEST){
//write_string("Envio de solicitud ARP");
for(i=0;i<ETH_ADDRS_LEN;i++)
rem_mac[i]=0xFF;  //si es solicitud la mac se pone en la dirección broadcast
}
else if(op == ARP_REPLY){
//write_string("Envio de respuesta ARP:");
for(i=0;i<ETH_ADDRS_LEN;i++)
rem_mac[i]=*(mac++);  //si es respuesta proporciona mi propia mac
}
else return;  //se desconoce el código de operación

//construcción del encabezado del frame ethernet
for(i=0;i<ETH_ADDRS_LEN;i++){
send_frame.destination[i] = rem_mac[i];
send_frame.source[i] = hard_addr[i];
}
send_frame.protocol = ARP_ETYPE;

//construcción del mensaje ARP
arp_packet.HWtype = TYPE_ETHERNET;
arp_packet.protocol = IP_ETYPE;
arp_packet.HWdirlen = ETH_ADDRS_LEN;
arp_packet.protdirlen = IP_DIR_LEN;
arp_packet.opcode = op;
for(i=0;i<ETH_ADDRS_LEN;i++){
arp_packet.MACaddSource[i] = hard_addr[i];
arp_packet.MACaddDest[i] = rem_mac[i];
}
for(i=0;i<IP_DIR_LEN;i++){
arp_packet.IPaddSource[i] = ip_address[i];
arp_packet.IPaddDest[i] = *(ip++);
}


//escritura en buffer y envío de la trama
NE64InitializeTransmissionBuffer();
NE64WriteEthernetHeaderToTxBuffer(&send_frame);
NE64WriteBytes(((void*)&arp_packet),ARP_LEN);
write_char(0x0D); write_char(0x0A);
NE64StartFrameTransmission(42);

}


//////////////////// RECEPCIÓN DE MENSAJE ARP /////////////////
void arp_receive(unsigned int address_buff){

struct arp_message arp_packet;
unsigned char i;

NE64InitializeOffsetToReadRxBuffer(address_buff);

//write_string("Paquete ARP recibido: ");

arp_packet.HWtype = NE64ReadWord();
//if(arp_packet.HWtype!=TYPE_ETHERNET){
//  write_string("Paquete ARP con tipo de hardware erroneo");
// return;
//}

arp_packet.protocol = NE64ReadWord();
//if(arp_packet.protocol!=IP_ETYPE){
// write_string("Paquete ARP no contiene datos de red IP");
// return;
//}

arp_packet.HWdirlen = NE64ReadByte();
arp_packet.protdirlen = NE64ReadByte();

arp_packet.opcode = NE64ReadWord();
for(i=0;i<ETH_ADDRS_LEN;i++){
arp_packet.MACaddSource[i] = NE64ReadByte();
}
for (i=0;i<IP_DIR_LEN;i++){
arp_packet.IPaddSource[i] = NE64ReadByte();
}
for(i=0;i<ETH_ADDRS_LEN;i++){
arp_packet.MACaddDest[i] = NE64ReadByte();
}
for(i=0;i<IP_DIR_LEN;i++){
arp_packet.IPaddDest[i]=NE64ReadByte();
}

//write_char(0x0A); write_char(0x0D);

//examinar si la dirección física es válida
//for(i=0;i<ETH_ADDRS_LEN;i++)
//if(arp_packet.MACaddDest[i]!=hard_addr[i] && arp_packet.MACaddDest[i]!= 0xFF && arp_packet.MACaddDest[i]!= 0){
// write_string("Paquete MAC con direccion fisica erronea");
// return; //dirección física incorrecta, no es mi MAC ni la dirección broadcast
//}

//examinar si la dirección de red es válida, si lo es, procesar el paquete
// if(equal_ip(arp_packet.IPaddDest,ip_address)){
if(arp_packet.opcode==ARP_REQUEST){
//write_string("Solicitud ARP");
arp_send(ARP_REPLY,arp_packet.IPaddSource,arp_packet.MACaddSource);
}
else if(arp_packet.opcode==ARP_REPLY){
//write_string("Respuesta ARP recibida, añadir respuesta a la tabla...");
arp_add_response(arp_packet.IPaddSource,arp_packet.MACaddSource);
return;
} else // la opción no fue ni solicitud ni respuesta
return;
// }// FIN si la ip corresponde a mi ip
// else{
//       write_string("Paquete ARP con direccion de red erronea");
// }

//write_string("Añadir paquete ARP a la tabla...");
arp_add_entry(arp_packet.IPaddSource,arp_packet.MACaddSource);
}


//////////////////// AÑADIR NUEVA ENTRADA A LA TABLA //////////////////
void arp_add_entry(unsigned char *ip, unsigned char *mac){
unsigned char i, type=DYNAMIC_ENTRY;

if(equal_ip(ip,ip_gateway)){
//write_string("Añadiendo la IP del gateway");
```

```
type=STATIC_ENTRY;
}
// cuando se agrega una entrada puede refrescarse una entrada ya resuleta,
// ser una entrada nueva, o una ip que no pertenece a la subred

if(!same_subnet(ip)){
//write_string("La IP no pertenece a esta subred, no se añade");
return; // no está en la subred, no se puede agregar a la tabla
}

// sí pertenece a la subred, se busca en la tabla para ver si ya existía
for(i=0;i<ARP_TABLE_SIZE;i++)
if(arp_table_entry[i].status==ARP_RESOLVED)
if(equal_ip(arp_table_entry[i].IPaddr,ip)){
//write_string("La direccion ya esta resuelta, se refrescara");
break;
}

if(i==ARP_TABLE_SIZE) // indica que la ip no estaba en la tabla, se busca una entrada libre
for(i=0;i<ARP_TABLE_SIZE;i++)
if(arp_table_entry[i].status==ARP_FREE){
//write_string("Añadir las direcciones en una entrada libre");
break;
}

if(i==ARP_TABLE_SIZE){// indica que la ip no estaba en la tabla y ya no había entradas libres
//write_string("No fue posible añadir direccion a la tabla");
return;
}

// si llegó hasta aqui, encontró una posición para insertar los datos en la tabla

arp_table_entry[i].status = ARP_RESOLVED;
arp_table_entry[i].type = type;
arp_table_entry[i].ttl = ARP_TTL;
arp_table_entry[i].retries = ARP_RETRIES;
arp_table_entry[i].MACaddr[0] = *(mac++);
arp_table_entry[i].MACaddr[1] = *(mac++);
arp_table_entry[i].MACaddr[2] = *(mac++);
arp_table_entry[i].MACaddr[3] = *(mac++);
arp_table_entry[i].MACaddr[4] = *(mac++);
arp_table_entry[i].MACaddr[5] = *(mac++);
arp_table_entry[i].IPaddr[0] = *(ip++);
arp_table_entry[i].IPaddr[1] = *(ip++);
arp_table_entry[i].IPaddr[2] = *(ip++);
arp_table_entry[i].IPaddr[3] = *(ip++);
mac-=6; ip-=4;

//write_string("Posicion donde se insertaron las direcciones:");
//write_hx(i); write_char(0x0D); write_char(0x0A);
//write_string("Las siguientes direcciones se añadieron a la tabla:");
//write_string("MAC:");
//for(i=0;i<6;i++)
//write_hx(mac[i]);
//write_char(0x0D); write_char(0x0A);
//write_string("IP:");
//for(i=0;i<4;i++)
//write_hx(ip[i]);
//write_char(0x0D); write_char(0x0A);
}


////////////////////// AÑADIR RESPUESTA A LA TABLA ////////////////
void arp_add_response(unsigned char *ip, unsigned char *mac){
unsigned char i;
for(i=0;i<ARP_TABLE_SIZE;i++){
if(equal_ip(arp_table_entry[i].IPaddr,ip)){
arp_table_entry[i].MACaddr[0]=*(mac++);
arp_table_entry[i].MACaddr[1]=*(mac++);
arp_table_entry[i].MACaddr[2]=*(mac++);
arp_table_entry[i].MACaddr[3]=*(mac++);
arp_table_entry[i].MACaddr[4]=*(mac++);
arp_table_entry[i].MACaddr[5]=*(mac++);
arp_table_entry[i].status = ARP_RESOLVED;
arp_table_entry[i].ttl = ARP_TTL;
} // fin if
} // fin for
}


////////////////// RUTINA DE MANTENIMIENTO DE LA TABLA ARP ////////////////
void arp_manage(void){
unsigned char i;

if(read_timer_ms(arp_timer))
return; // aun no termina el tiempo de mantenimiento
reset_timer_value(arp_timer,ARP_TABLE_MANAGE,SEC);

for(i=0;i<ARP_TABLE_SIZE;i++)
switch(arp_table_entry[i].status){
case ARP_FREE:
break;
case ARP_PENDING:
if(--arp_table_entry[i].ttl==0){
if(arp_table_entry[i].retries){
arp_table_entry[i].retries--;
arp_table_entry[i].ttl = ARP_RESEND;
arp_send(ARP_REQUEST,arp_table_entry[i].IPaddr, (unsigned char *)0);
} else {
if(arp_table_entry[i].type == STATIC_ENTRY)
arp_table_entry[i].retries = ARP_RETRIES;
else
arp_table_entry[i].status = ARP_FREE;
}
}
break;
case ARP_RESOLVED:
if(--arp_table_entry[i].ttl==0){
if(arp_table_entry[i].type == DYNAMIC_ENTRY)
arp_table_entry[i].status = ARP_FREE;
else{
arp_table_entry[i].ttl = ARP_RESEND;
arp_table_entry[i].retries = ARP_RETRIES;
arp_send(ARP_REQUEST,arp_table_entry[i].IPaddr, (unsigned char *)0);
}
}
break;
} //fin switch fin del for

}

//////// VERIFICAR SI 2 IPs SON IGUALES/////////////////
unsigned char equal_ip(unsigned char *ip1, unsigned char *ip2){
unsigned char i;
for(i=0;i<IP_DIR_LEN;i++)
if(*(ip1++)!=*(ip2++))
return FALSE;
return TRUE;
}


//////// VERIFICAR SI UNA IP ESTÁ DENTRO DE LA SUBRED /////////////////
unsigned char same_subnet(unsigned char *ip){
unsigned long ip1,ip2 ;
unsigned long ipsubnet1, ipsubnet2;
unsigned long netwmask;
```

```
netwmask = (*((unsigned long *)ip_netmask));
ip1 = (*((unsigned long *)ip));
ip2= (*((unsigned long *)ip_address));

ipsubnet1= ip1 & netwmask;
ipsubnet2= ip2 & netwmask;

return equal_ip(((unsigned char *)&ipsubnet1),((unsigned char *)&ipsubnet2));
}
```

PROTOCOLO UDP

udp.h

```
* consideraciones:

La suma de verificación no se implemeta, ya que el protocolo da la
libertad de implementarla o no hacerlo, para este caso no se realiza
con la finalidad de generar un código mas reducido, debido a las
limitaciones en la capacidad de nuestro sistema */

//cantidad de sockets udp disponibles
#define UDP_SOCKETS_NUM 4

//longitud del encabezado udp
#define UDP_LEN 8
#define UDP_HEADER_OFFSET UDP_LEN

//longitud de todos los encabezados
#define HEADERS_LEN 42 // 14 bytes de ethernet + 20 bytes de ip +8 bytes de udp

// los sockets udp solo pueden encontrarse en 3 estados
#define UDP_FREE 0
#define UDP_CLOSED 1
#define UDP_OPEN 2

//tipos de sockets
#define UDP_SERVER 0
#define UDP_CLIENT 1

//datagarama udp
struct udp_datagram{
unsigned int local_port;
unsigned int remote_port;
unsigned int message_len;
unsigned int checksum;
unsigned int data;
};

//socket udp
struct udp_sock{
unsigned char state;
unsigned char type;
unsigned int localport;
unsigned int remoteport;
unsigned char remoteip[4];
void(*udp_application)(char, unsigned int, unsigned int);
};


// prototipos de funciones
void udp_socket_init(void);
char udp_get_sock(void);
void udp_open_sock(char, unsigned char, unsigned char *, unsigned int, unsigned int,void(*)(char, unsigned char, unsigned int));
void udp_change_sock_state(char,char);
void udp_send(char, unsigned char *, unsigned int);
void udp_receive(unsigned int, unsigned int, unsigned char);
```

udp.c

```
#include "ip.h"
#include "udp.h"
#include "ne64api.h"
#include "ne64config.h"
#include "general_functions.h"
#include "sci.h"

// sockets udp disponibles
struct udp_sock udp_socket[UDP_SOCKETS_NUM];

//paquete udp recibido
struct udp_datagram udp_datagram_receive;

//funciones de manejo de sockets
_____

//inicialización de sockets
_____
void udp_socket_init(){
unsigned char i;
for(i=0;i<UDP_SOCKETS_NUM;i++)
udp_socket[i].state=UDP_FREE;
}


//obtener un socket libre
_____
char udp_get_sock(){
char i;

for(i=0;i<UDP_SOCKETS_NUM;i++)
if(udp_socket[i].state==UDP_FREE){
udp_socket[i].state = UDP_CLOSED; //el socket se ha seleccionado pero aún no se abre
return i;
}
//si llegamos hasta aqui, no se encontró algún socket libre
return -1;
}

//abrir un socket
_____
void udp_open_sock(char socket_id, unsigned char type, unsigned char *remip, unsigned int remport, unsigned int localport, void(*udp_app)(char, unsigned char, unsigned int)){
unsigned char i;

//validar los datos recibidos por la función
if(socket_id>=UDP_SOCKETS_NUM)
return;

if(type==UDP_SERVER)
if(localport==0 || udp_app==0)
return; //no podemos abir un socket servidor si no se especifica el puerto o la función que va a recibir datos

if(type==UDP_CLIENT)
if(*((unsigned long *)remip)==0 || remport==0)
return; //no podemos abrir un socket cliente si no se especifica el destino
//local port puede ser cero

udp_socket[socket_id].state = UDP_OPEN;
udp_socket[socket_id].type = type;
udp_socket[socket_id].localport = localport;
udp_socket[socket_id].udp_application=udp_app;

if(type == UDP_CLIENT){
for(i=0;i<4;i++)
udp_socket[socket_id].remoteip[i]=*(remip++);
udp_socket[socket_id].remoteport = remport;
}

}
```

```
// cerrar un socket o liberar un socket que ya no esté en uso_____
_____
void udp_change_sock_state(char socket_id, char new_state){
if(socket_id>=UDP_SOCKETS_NUM)
return; // invalid number of socket id

udp_socket[socket_id].state = new_state;
}
// terminan funciones de manejo de los sockets _____
_____

// envío de mensajes udp _____
_____

void udp_send(char socket_id, unsigned char *data, unsigned int datalen){
unsigned int tmp;

//validar los datos recibidos
if(socket_id>=UDP_SOCKETS_NUM)
return;

if(udp_socket[socket_id].state!=UDP_OPEN)
return; //el socket no está abierto, no se pueden enviar datos

//el tamaño de los datos no puede exceder el tamaño máximo del buffer
if(datalen>EMAC_TX_SZ-HEADERS_LEN)
datalen = EMAC_TX_SZ-HEADERS_LEN;

//llenar el inicio del buffer con los datos del encabezado udp
tmp=udp_socket[socket_id].localport;
*(data++)=tmp >>8;
*(data++)=tmp; //puerto local
tmp=udp_socket[socket_id].remoteport;
*(data++)=tmp >>8;
*(data++)=tmp; //puerto remoto
tmp=datalen + UDP_LEN;
*(data++)=tmp>>8;
*(data++)=tmp; // longitud del mensaje udp
*(data++)=0;
*(data++)=0; //checksum --> no se implementa

//enviar el datagrama udp a través de ip
ip_send(udp_socket[socket_id].remoteip, data-=UDP_LEN, tmp, UDP_ETYPE); // DONE

}
//_____
_____
_____

// recepción de mensajes udp
void udp_receive(unsigned int udp_addrs_buff, unsigned int datalen, unsigned char *ip){
char i;


NE64InitializeOffsetToReadRxBuffer(udp_addrs_buff);
udp_datagram_receive.local_port = NE64ReadWord(); // puede ser cero, asi que no se valida
udp_datagram_receive.remote_port= NE64ReadWord();
if(udp_datagram_receive.remote_port==0)
return; // no se procesa el paquete porque no está especificado a que aplicación va dirigido

udp_datagram_receive.message_len = NE64ReadWord();
if(udp_datagram_receive.message_len<UDP_LEN)
return; // el mensaje no contiene la cantidad de datos necesarios

udp_datagram_receive.checksum = NE64ReadWord(); // no se valida el checksum
udp_datagram_receive.data = udp_addrs_buff + UDP_LEN; //posición de los datos en el buffer

//buscar el socket correspondiente al puerto de recepción
for(i=0;i<UDP_SOCKETS_NUM;i++)
if(udp_socket[i].state == UDP_OPEN && udp_socket[i].localport == udp_datagram_receive.remote_port){

break;
}

//se encontró el socket????
if(i==UDP_SOCKETS_NUM){
write_string("No se encontro socket UDP asociado a la aplicacion");
return; // no se encontró el socket asociado a la aplicación
}

write_string("se recibieron datos válidos para un puerto específico");
//se recibieron datos válidos para un puerto específico
if(udp_socket[i].type == UDP_SERVER){
udp_socket[i].remoteport = udp_datagram_receive.local_port;
udp_socket[i].remoteip[0]=*(ip++);
udp_socket[i].remoteip[1]=*(ip++);
udp_socket[i].remoteip[2]=*(ip++);
udp_socket[i].remoteip[3]=*(ip++);
}

write_string("entregar el paquete a la aplicacion asociada al socket");
//entregar el paquete a la aplicación asociada al socket
udp_socket[i].udp_application(i,udp_datagram_receive.data, udp_datagram_receive.message_len-UDP_LEN);

}
```

PROTOCOLO IP

ip.h

```
#define IPv4 4

// números de protocolo que usan IP
#define ICMP_ETYPE 1
#define UDP_ETYPE 17
#define TCP_ETYPE 6

//#definiciones constantes de los mensajes
#define IP_VIHL 0x45
#define IP_MIN_LEN 5
#define IP_MAX_OP_LEN 10
#define IP_MAX_HLEN 15
#define IP_LEN_BYTE(x) x*4
#define IP_TTL 100

//valores del campo de banderas
#define IP_FLAG_DF 0x4000 //Don´t Fragment
#define IP_FLAG_MF 0x2000 //More Fragments
#define IP_OFFSET_MASK 0x1FFF //máscara para obtener el valor de OFFSET del fragmento

//definiciones del protocolo IP

struct ip_datagram{
unsigned char vihl;
unsigned char tos;
unsigned int tlen;
unsigned int id;
unsigned int flags_offset;
unsigned char ttl;
unsigned char protocol;
unsigned int checksum;
```

```
unsigned char IPaddSource[4];
unsigned char IPaddDest[4];
unsigned char opt[IP_LEN_BYTE(IP_MAX_OP_LEN)];
unsigned int  data;
};



//prototipos de funciones
char ip_send(unsigned char *, unsigned char *, unsigned int,
unsigned char);
void ip_receive(unsigned int);
unsigned int ip_dlen(unsigned int);
unsigned int ip_checksum(unsigned int *, unsigned int);
unsigned int ip_add_to_checksum(unsigned int, unsigned
int);
```

ip.c

```
#include "ip.h"
#include "arp.h"
#include "ne64api.h"
#include "address.h"
#include "sci.h"

struct ip_datagram ip_packet_send;
struct ip_datagram ip_packet_receive;
unsigned int ip_id=0;
extern struct TEthernetFrame send_frame;

// envío de datagramas ip
_____
char ip_send(unsigned char *ip, unsigned char *data,
unsigned int len, unsigned char protocol){
unsigned char *mac, i, tmp_ip[4];

//verificar si la ip pertenece a esta subred o está fuera,
//en cuyo caso la petición se envía al gateway
if(!same_subnet(ip)){
write_string("La IP esta fuera de la red, se transfiere el
paquete al gateway");
tmp_ip[0] = ip_gateway[0];
tmp_ip[1] = ip_gateway[1];
tmp_ip[2] = ip_gateway[2];
tmp_ip[3] = ip_gateway[3];
}
else{
tmp_ip[0] = *(ip++);
tmp_ip[1] = *(ip++);
tmp_ip[2] = *(ip++);
tmp_ip[3] = *(ip++);
ip-=4;
}

// buscar la dirección MAC preguntando a ARP, si aún no
está lista se aborta el proceso
mac=arp_get_mac(tmp_ip);
if(mac==0)
return -1; // indica que la mac no se encuantra aún
disponible

//frame ethernet
for(i=0;i<ETH_ADDRS_LEN;i++){
send_frame.destination[i]=*mac++;
send_frame.source[i]=hard_addr[i];
}
send_frame.protocol=IP_ETYPE;
send_frame.buf_index = ETH_ADDRS_LEN;

//paquete ip
ip_packet_send.vihl=IP_VIHL;
ip_packet_send.tos=0;
ip_packet_send.tlen=IP_LEN_BYTE(IP_MIN_LEN)+len;  //la
longitud total se expresa en bytes
ip_packet_send.id=ip_id++;
ip_packet_send.flags_offset=0;
ip_packet_send.ttl=IP_TTL;
ip_packet_send.protocol=protocol;
ip_packet_send.checksum=0;
for(i=0;i<IP_DIR_LEN;i++){
ip_packet_send.IPaddSource[i]=ip_address[i];
ip_packet_send.IPaddDest[i]=*(ip++);
}
ip_packet_send.data=ETH_HDR_LEN+IP_LEN_BYTE(IP_MIN
_LEN);
ip_packet_send.checksum=ip_checksum(  &ip_packet_send,
IP_LEN_BYTE(IP_MIN_LEN) );

//ahora que está el datagrama ip ensamblado, se almacena el
el buffer Tx y se envía por la red
NE64InitializeTransmissionBuffer();
NE64WriteEthernetHeaderToTxBuffer(&send_frame);
NE64WriteBytes(((tU08
*)&ip_packet_send),IP_LEN_BYTE(IP_MIN_LEN));
NE64WriteBytes(data,len);
NE64StartFrameTransmission(IP_LEN_BYTE(IP_MIN_LEN)+le
n+14);
return 1;
}


//calcular cuántos datos puede enviar ip
unsigned int ip_dlen(unsigned int dlen){

dlen = dlen + IP_LEN_BYTE(IP_MIN_LEN);
dlen = NE64dlen (dlen);
dlen = dlen - IP_LEN_BYTE(IP_MIN_LEN);

return dlen;
}

//recepción de datagramas ip
_____
void ip_receive(unsigned int ip_addrs_buff){
unsigned char i;
unsigned int datalen, optionlen;

NE64InitializeOffsetToReadRxBuffer(ip_addrs_buff);
ip_packet_receive.vihl=NE64ReadByte();

i=ip_packet_receive.vihl&0xF0;
if(i>>4!=IPv4)
return;
i=ip_packet_receive.vihl & 0x0F;
if(i<IP_MIN_LEN)
return;  //la cabecera no contiene los datos suficientes

ip_packet_receive.tos=NE64ReadByte();
ip_packet_receive.tlen=NE64ReadWord();
ip_packet_receive.id=NE64ReadWord();
ip_packet_receive.flags_offset=NE64ReadWord();
ip_packet_receive.ttl=NE64ReadByte();
ip_packet_receive.protocol=NE64ReadByte();
ip_packet_receive.checksum=NE64ReadWord();

for(i=0;i<IP_DIR_LEN;i++)
ip_packet_receive.IPaddSource[i]=NE64ReadByte();
for(i=0;i<IP_DIR_LEN;i++)
ip_packet_receive.IPaddDest[i]=NE64ReadByte();

write_char(0x0A); write_char(0x0D);

i=ip_packet_receive.vihl & 0x0F;
datalen=ip_packet_receive.tlen-IP_LEN_BYTE(i);
ip_packet_receive.data= (ip_addrs_buff) + (IP_LEN_BYTE(i));

//verificar si el paquete recibido es para mi
if(!equal_ip(ip_packet_receive.IPaddDest,ip_address))
return;

//las opciones de ip no se implementan en esta pila,
```

```
//se deja el código por si es necesario algún procesamiento de ellas

/*
i=ip_packet_receive.vihl&0x0f;
optionlen=IP_LEN_BYTE(i-IP_MIN_LEN);
for(i=0;i<optionlen;i++)
ip_packet_receive.opt[i]=NE64ReadByte();*/

//ya tenemos el paquete completo, validar los datos que contiene
//para ver si es un paquete válido o se debe descartar

//debido a los recursos limitados de nuestro microcontrolador
//no se procesan paquetes fragmentados, si se detecta uno se descarta
if(ip_packet_receive.flags_offset&IP_FLAG_MF)
return;  //el paquete estaba fragmentado
if(ip_packet_receive.flags_offset&IP_OFFSET_MASK)
return;  //también corresponde a un paquete fragmentado

//validar la suma de verificación
i=ip_packet_receive.vihl & 0x0F;
if( ip_checksum( &ip_packet_receive, IP_LEN_BYTE(i)) )
return;  //suma de verificación alterada

write_string("Ckecksum IP OK!!!");

//todo bien, entregar el paquete al siguiente nivel para continuar el procesamiento
switch(ip_packet_receive.protocol){
case  ICMP_ETYPE:
write_string("Paquete ICMP recibido");
icmp_receive(ip_packet_receive.data,datalen,ip_packet_receive.IPaddSource);
break;
case  UDP_ETYPE:
write_string("Paquete UDP recibido");
udp_receive(ip_packet_receive.data,datalen,
ip_packet_receive.IPaddSource);
break;
case  TCP_ETYPE:
write_string("Paquete TCP recibido");
tcp_receive(ip_packet_receive.data,datalen,
ip_packet_receive.IPaddSource);
break;
default:
return;  // no se reconoce el código de protocolo
}

}



//cálculo del checksum ip
unsigned int ip_checksum(unsigned int *data, unsigned int len){
unsigned int oddbyte;
unsigned long sum=0,tmp;


//la longitud está dada en bytes, y el cálculo se hace en palabras
oddbyte=len%2;
len=len/2;


while(len){
sum += *(data++);
tmp=sum&0xFFFF0000;
if(tmp){          // existe acarreo en la suma
sum &= 0x0000FFFF;  // nos quedamos con los 16 bits
sum++;            // y sumamos el acarreo
}
len--;
}

if(oddbyte){
tmp=*data;
tmp&=0xFF00;
sum+=tmp;
tmp=sum&0xFFFF0000;
if(tmp){        // existe acarreo en la suma
sum &= 0x0000FFFF;  // nos quedamos con los 16 bits
sum++;            // y sumamos el acarreo
}
}

sum=~sum;
return (unsigned int)sum;
}



//añadir valores al cálculo del checksum
unsigned int ip_add_to_checksum(unsigned int sum, unsigned int data){
unsigned long cs, tmp;

cs=((unsigned long)sum)+((unsigned long)data);
tmp=cs&0xFFFF0000;
if(tmp){
cs&=0x0000FFFF;
cs++;
}

return (unsigned int)cs;

}
```

PROTOCOLO ICMP

icmp.h

```
struct icmp_echo_message{
unsigned char type;     // 8 echo request; 0 echo reply
unsigned char code;     // para mensajes icmp
unsigned int  checksum;
unsigned int  id;
unsigned int  secnum;
unsigned int datalen;
};


//definiciones para los mensajes icmp
#define ICMP_ECHO_REQUEST  8
#define ICMP_ECHO_REPLY    0
//#define ICMP_CODE          0
#define ICMP_LEN           8  // longitud de los campos fijos de los mensajes echo icmp


//prototipos de fucniones
void icmp_receive(unsigned int, unsigned int, unsigned char*);
void icmp_send_echo_response(unsigned char *,unsigned int);
```

icmp.c

```
#include "ip.h"
#include "icmp.h"
#include "ne64api.h"
#include "sci.h"
#include "general_functions.h"

struct icmp_echo_message icmp_message_receive;
unsigned int icmp_id = 0;
unsigned int icmp_secnum = 0;

//recepción de mensajes icmp; sólo tiene soporte para el tipo ECHO
void icmp_receive(unsigned int icmp_addrs_buff, unsigned int len, unsigned char *ip){
```

```
unsigned int cs=0, i=0;

NE64InitializeOffsetToReadRxBuffer(icmp_addrs_buff);
icmp_message_receive.type=NE64ReadByte();
if(icmp_message_receive.type != ICMP_ECHO_REQUEST &&
icmp_message_receive.type != ICMP_ECHO_REPLY)
return;  // no se procesa el mensaje porque no corresponde
al tipo ECHO

icmp_message_receive.code = NE64ReadByte();

icmp_message_receive.checksum = NE64ReadWord();
icmp_message_receive.id = NE64ReadWord();
icmp_message_receive.secnum = NE64ReadWord();
icmp_message_receive.datalen=len-ICMP_LEN;  // longitud
del paquete icmp menos los 8 bytes de  los campos anteriores

//verificar si la suma de comprobación es correcta

NE64InitializeOffsetToReadRxBuffer(icmp_addrs_buff);
for(i=0;i<len;i++)
data_buff[i]=NE64ReadByte();
cs=ip_checksum(data_buff,len);

if(cs){
write_string("Checksum ICMP incorrecto");
return;  //valor incorrecto de cs
}

write_string("Checksum ICMP OK!!");

//todo bien, paquete icmp listo para procesarse
if(icmp_message_receive.type==ICMP_ECHO_REQUEST){
//enviar una respuesta
icmp_send_echo_response(ip,len);

}

}

void icmp_send_echo_response(unsigned char *ip, unsigned
int len){
unsigned int cs=0, i=0;
//crear paquete icmp, y calcular la suma de verificación
write_string("Generando respuesta ICMP...");
data_buff[0]=ICMP_ECHO_REPLY;
data_buff[2]=0;
data_buff[3]=0;
cs=ip_checksum(data_buff,len);
data_buff[2]=(cs>>8);
data_buff[3]=(cs & 0x00FF);

// ya esta listo el paquete para enviar la respuesta
ip_send(ip,data_buff,ICMP_LEN+icmp_message_receive.data
len,ICMP_ETYPE);

}
```

PROTOCOLO TCP

tcp.h

```
//cantidad de sockets TCP disponibles
#define TCP_SOCKETS_NUM  6

//longitud mínima de una cabecera TCP (20 bytes = 5
palabras)
#define TCP_MIN_LEN  5
//longitud máxima de las opciones de la cabecera (40 bytes
= 10 palabras)
#define TCP_MAX_OPT  10
//tamaño del MTU definido para TCP
#define TCP_MTU  512
#define TCP_LEN_BYTE(x) x*4

//espacio en el buffer de escritura para aplicaciones que
deseen enviar datos
#define                                  TCP_HEADER_OFFSET
TCP_LEN_BYTE(TCP_MIN_LEN)

//longitud de todos los encabezados para aplicaciones que
usen tcp
#define HEADERS_LEN_  54 // 14 bytes de ethernet + 20
bytes de ip + 20 bytes de tdp

//número de retransmisiones TCP
#define TCP_RETRIES  7
//tiempo de espera para realizar retransmisiones
#define TCP_TIME_OUT  4
//tiempo de restransmisión inicial
#define TCP_INIT_TIME_OUT 1
//tiempo que espera una conexión TCP para cerrarse si no se
reciben datos
#define TCP_IDLE_TIME  120

#define INITIAL_SEC_NUM 100

//banderas del encabezado TCP
#define TCP_URG 0x20
#define TCP_ACK 0x10
#define TCP_PSH 0x08
#define TCP_RST 0x04
#define TCP_SYN 0x02
#define TCP_FIN 0x01

//tipos de sockets TCP
#define TCP_SERVER       1
#define TCP_CLIENT       2
#define TCP_SERVER_CLIENT 3

//estados de un socket TCP
#define TCP_FREE          0
#define TCP_CLOSED        1
#define TCP_LISTENING     3
#define TCP_SYN_SENT      4
#define TCP_SYN_RECEIVED  5
#define TCP_CONNECTED     6
#define TCP_FIN_WAIT1     7
#define TCP_FIN_WAIT2     8
#define TCP_CLOSING       9
#define TCP_TIME_WAIT     10
#define TCP_CLOSE_WAIT    11
#define TCP_LAST_ACK      12


//eventos notificados a la aplicación
#define TCP_EVENT_REQUEST_CONNECTION  1
#define TCP_EVENT_CONNECTED       2
#define TCP_EVENT_CLOSE           3
#define TCP_EVENT_ABORT           4
#define TCP_EVENT_ACK             5
#define TCP_EVENT_DATA            6
#define TCP_EVENT_RESEND          7

//paquete tcp
struct tcp_packet{
unsigned int  localport;
unsigned int  remport;
unsigned long secnum;
unsigned long acknum;
unsigned int  hlen_flags;
unsigned int  window;
unsigned int  checksum;
unsigned int  urgpoint;
unsigned char tcpopt[TCP_LEN_BYTE(TCP_MAX_OPT)];
unsigned int  data;
};


//socket tcp
```

```
struct tcp_sock{
unsigned char state;
unsigned char type;
unsigned char remip[4];
unsigned int  remport;
unsigned int  localport;
unsigned long secnum;
unsigned long remsecnum;
unsigned char data_unack;
unsigned char retries;
unsigned char closepending;
unsigned char retransmitclock;
unsigned char timetoliveclock;
unsigned int  dlen;

char (*application)(char, unsigned char);
};


//prototipos de funciones
void tcp_socket_init(void);
char tcp_get_sock(unsigned char, char(*)(char, unsigned char));
char tcp_free_sock(char);
char tcp_sock_listen(char, unsigned int);
char tcp_sock_connect(char, unsigned char *, unsigned int, unsigned int);
char tcp_get_sockstate(char);
char tcp_data_send(char, char *, unsigned int);
char tcp_send(char, unsigned char, unsigned char *);
char tcp_receive(unsigned int tcp_addrs_buff, unsigned int datalen, unsigned char *ip);
void tcp_poll(void);
char tcp_find_sock(struct tcp_packet *, unsigned char *);
void tcp_reset(struct tcp_packet *, unsigned char *);
unsigned int tcp_dlen(unsigned int);
char tcp_close(char);
char tcp_abort(char);
unsigned int add_pseudoheader_to_cs(unsigned int, unsigned char *, unsigned int);
```

tcp.c

```
#include "tcp.h"
#include "ip.h"
#include "ne64api.h"
#include "timers.h"
#include "address.h"
#include "general_functions.h"
#include "sci.h"

struct tcp_packet tcp_received_packet;
struct tcp_packet tcp_sent_packet;
struct tcp_sock   tcp_socket[TCP_SOCKETS_NUM+1];

//inicialización de sockets
//------------------------------------------------------------------------
void tcp_socket_init(void){
char i;
for(i=0;i<=TCP_SOCKETS_NUM;i++){
tcp_socket[i].state = TCP_FREE;
tcp_socket[i].dlen = 0;
tcp_socket[i].data_unack = FALSE;
tcp_socket[i].closepending = FALSE;
}
}

//obtención de un socket libre
//------------------------------------------------------------------------
char tcp_get_sock(unsigned char type, char(* tcp_app)(char, unsigned char)){
char i;

if(tcp_app == 0)
return -1;

//buscar un socket libre
for(i=0;i<TCP_SOCKETS_NUM;i++)
if(tcp_socket[i].state==TCP_FREE){
tcp_socket[i].retransmitclock = create_timer(DOWN,0,SEC);
tcp_socket[i].timetoliveclock = create_timer(DOWN,0,SEC);
tcp_socket[i].state = TCP_CLOSED;
tcp_socket[i].type = type;
tcp_socket[i].application = tcp_app;
return i;
}

return -1; //ya no hay sockets disponibles
}


//liberar un socket
//------------------------------------------------------------------------
/*  valores de retorno:
-1  información de socket incorrecta
-2  incapaz de liberar el socket porque se encuentra en uso
1   operación realizada con éxito
*/

char tcp_free_sock(char socket){

if(socket>=TCP_SOCKETS_NUM || socket < 0)
return -1;  // el identificador de socket no es válido

if(tcp_socket[socket].state!=TCP_FREE &&
tcp_socket[socket].state != TCP_CLOSED)
return -2;  // el socket está en uso y no se puede liberar

tcp_socket[socket].state = TCP_FREE;
tcp_socket[socket].localport = 0;
tcp_socket[socket].remport = 0;
tcp_socket[socket].remip[0] = 0;
tcp_socket[socket].remip[1] = 0;
tcp_socket[socket].remip[2] = 0;
tcp_socket[socket].remip[3] = 0;
free_timer(tcp_socket[socket].retransmitclock);
free_timer(tcp_socket[socket].timetoliveclock);

return 1;
}

//verificar el estado del socket
//------------------------------------------------------------------------
char tcp_get_sockstate(char socket){
if(socket>=TCP_SOCKETS_NUM || socket < 0)
return -1;  //error

return tcp_socket[socket].state;
}


//abrir un socket para poner a un servidor a escuchar
//------------------------------------------------------------------------
char tcp_sock_listen(char socket, unsigned int localport){

if(socket>=TCP_SOCKETS_NUM || socket < 0)
return -1;

if(localport==0)
return -1;

if(tcp_socket[socket].type != TCP_SERVER &&
tcp_socket[socket].type != TCP_SERVER_CLIENT)
return -1;

if(tcp_socket[socket].application == 0)
return -1;

if(tcp_socket[socket].state != TCP_CLOSED)
return -1;
```

```
tcp_socket[socket].data_unack = FALSE;
tcp_socket[socket].dlen = 0;
tcp_socket[socket].state = TCP_LISTENING;
tcp_socket[socket].localport=localport;

return 1;
}

//abrir un socket cliente para realizar una conexión a un servidor _____
char tcp_sock_connect(char socket, unsigned char *remip, unsigned int remport, unsigned int localport) {


if(*((unsigned long *)remip) == 0 || remport == 0 || localport == 0)
return -1;

if(socket>=TCP_SOCKETS_NUM || socket < 0)
return -1;

if(tcp_socket[socket].type == TCP_SERVER)
return -1;

if(tcp_socket[socket].type == TCP_SERVER_CLIENT)
if( tcp_socket[socket].state!=TCP_LISTENING &&
tcp_socket[socket].state!=TCP_CLOSED     )
return -1;

if(tcp_socket[socket].state!=TCP_CLOSED)
return -1;

tcp_socket[socket].remip[0]= *remip++;
tcp_socket[socket].remip[1]= *remip++;
tcp_socket[socket].remip[2]= *remip++;
tcp_socket[socket].remip[3]= *remip++;
remip-=4;

tcp_socket[socket].remport = remport;
tcp_socket[socket].localport = localport;
tcp_socket[socket].secnum = INITIAL_SEC_NUM;
tcp_socket[socket].remsecnum = 0;
tcp_socket[socket].data_unack = FALSE;
tcp_socket[socket].dlen = 0;

tcp_send(socket,TCP_SYN,data_buff);

//ajustes al socket que maneja la conexión
tcp_socket[socket].state = TCP_SYN_SENT;
tcp_socket[socket].secnum ++;    //siguiente número de secuencia a enviar en el próximo paquete
tcp_socket[socket].retries = TCP_RETRIES;
reset_timer_value(tcp_socket[socket].retransmitclock,TCP_INIT_TIME_OUT,SEC);

return 1;
}

//cantidad de datos que se le permite enviar a tcp _____
unsigned int tcp_dlen(unsigned int dlen) {

dlen = dlen + TCP_LEN_BYTE(TCP_MIN_LEN);
if(dlen >TCP_MTU)
dlen = TCP_MTU;
dlen = ip_dlen(dlen);

return dlen - TCP_LEN_BYTE(TCP_MIN_LEN);
}

//enviar datos a través de un socket tcp _____
char tcp_data_send(char socket, char *data, unsigned int dlen) {
unsigned int cs;

dlen = tcp_dlen(dlen);
if(socket>=TCP_SOCKETS_NUM || socket < 0)
return -1;

//validar el socket de comunicación
if(tcp_socket[socket].state != TCP_CONNECTED)
return -1;  //no se pueden enviar datos porque el socket aun no se encuentra conectado

if(tcp_socket[socket].data_unack == TRUE)
return -1;  //no se pueden enviar datos porque no se ha recibido la confirmación de datos anteriores

if(data==0 || dlen==0)
return -1;

tcp_socket[socket].dlen = dlen;
tcp_send(socket, TCP_ACK|TCP_PSH, data);

//ajustes al socket que maneja la conexión
tcp_socket[socket].secnum += tcp_socket[socket].dlen;
//próximo número de secuencia a enviar
tcp_socket[socket].data_unack = TRUE;
tcp_socket[socket].retries = TCP_RETRIES;
reset_timer_value(tcp_socket[socket].retransmitclock, TCP_TIME_OUT, SEC);
reset_timer_value(tcp_socket[socket].timetoliveclock, TCP_IDLE_TIME, SEC);

return 1; //procesado correctamente
}


//envio de todo tipo de paquetes tcp _____
char tcp_send(char socket, unsigned char flags, unsigned char *data) {
unsigned int cs;
int data_sent;

if(socket>=TCP_SOCKETS_NUM || socket < 0)
return -1;

//crear el paquete tcp para enviar
*((unsigned int *)data) = tcp_socket[socket].localport;
data+=2;
*((unsigned int *)data) = tcp_socket[socket].remport;
data+=2;
*((unsigned long *)data)= tcp_socket[socket].secnum;
data+=4;
*((unsigned long *)data)= tcp_socket[socket].remsecnum;
data+=4;
*((unsigned int *)data) = ((TCP_MIN_LEN<<12)|flags);
data+=2;
*((unsigned int *)data) = TCP_MTU;
data+=2;
*((unsigned int *)data) = 0;                    data+=2;
*((unsigned int *)data) = 0;                    data-=(TCP_LEN_BYTE(TCP_MIN_LEN)-2);

//cálculo del checksum
cs=ip_checksum(data,TCP_LEN_BYTE(TCP_MIN_LEN)+tcp_socket[socket].dlen);
cs=add_pseudoheader_to_cs(cs,tcp_socket[socket].remip,tcp_socket[socket].dlen+TCP_LEN_BYTE(TCP_MIN_LEN));
*(data+16) = cs>>8;
*(data+17) = cs;

//envío de datos a través de IP
ip_send(tcp_socket[socket].remip,                data ,TCP_LEN_BYTE(TCP_MIN_LEN)+tcp_socket[socket].dlen,TCP_ETYPE);

return 1;
```

```
}

//recepción de paquetes tcp
_____
char tcp_receive(unsigned int tcp_addrs_buff, unsigned int len, unsigned char *ip){
unsigned char optionlen, hlen, socket, flags=0;
unsigned int cs, dlen,i;
unsigned long tmp;

NE64InitializeOffsetToReadRxBuffer(tcp_addrs_buff);
tcp_received_packet.localport = NE64ReadWord();
if(tcp_received_packet.localport==0)
return -1;   //puerto local del dispositivo remoto no especificado
tcp_received_packet.remport = NE64ReadWord();
if(tcp_received_packet.remport == 0)
return -1;  // puerto de destino no especificado
tcp_received_packet.secnum  = NE64ReadWord();
tcp_received_packet.secnum  = tcp_received_packet.secnum << 16;
tcp_received_packet.secnum |= NE64ReadWord();
tcp_received_packet.acknum  = NE64ReadWord();
tcp_received_packet.acknum  = tcp_received_packet.acknum << 16;
tcp_received_packet.acknum |= NE64ReadWord();
tcp_received_packet.hlen_flags = NE64ReadWord();
hlen=tcp_received_packet.hlen_flags >> 12;
if(hlen < TCP_MIN_LEN)
return -1;  //el paquete tcp no contiene la cantidad de datos mínima
hlen=TCP_LEN_BYTE(hlen);
if(len<hlen)
return -1;  //cantidad de datos incorrecta
dlen = len-hlen;
tcp_received_packet.window = NE64ReadWord();
tcp_received_packet.checksum = NE64ReadWord();
tcp_received_packet.urgpoint = NE64ReadWord();
tcp_received_packet.data = tcp_addrs_buff + hlen;


optionlen=hlen - TCP_LEN_BYTE(TCP_MIN_LEN);
if(optionlen > TCP_LEN_BYTE(TCP_MAX_OPT))
return;
for(i=0;i<optionlen;i++)
tcp_received_packet.tcpopt[i]=NE64ReadByte();


//verificar la suma de comprobación tcp
cs=ip_checksum(&tcp_received_packet,hlen);
//añadir los datos al checksum
cs=~cs;
for(i=dlen;i>1;i-=2)
cs=ip_add_to_checksum(cs,NE64ReadWord());
if(i)
cs=ip_add_to_checksum(cs,(NE64ReadWord()&0xff00));
cs=~cs;
//añadir los datos del pseudoencabezado
cs=add_pseudoheader_to_cs(cs,ip,len);
if(cs){
write_char(0x0a); write_char(0x0d);
write_string("Checksum TCP incorrecto");
return -1;  //suma de verificación alterada
}

write_char(0x0a); write_char(0x0d);
write_string("Checksum TCP OK!!");



//buscar el socket asociado a la conexión
socket = tcp_find_sock(&tcp_received_packet, ip);
if(socket==-1){ //algún problema con los datos recibidos, enviar una trama de reset
tcp_reset(&tcp_received_packet,ip);
return -1;
}

NE64InitializeOffsetToReadRxBuffer(tcp_received_packet.data);

//validación del contenido del paquete

//si se recibe un paquete de reset se termina la comunicación
//
if(tcp_received_packet.hlen_flags & TCP_RST){
write_string("Reset recibido");                    // R
if(tcp_socket[socket].state != TCP_LISTENING)      //
S
tcp_socket[socket].application(socket,TCP_EVENT_ABORT);
// T
if(tcp_socket[socket].type & TCP_SERVER)           //
tcp_socket[socket].state = TCP_LISTENING;          // F
else                                               // L
tcp_socket[socket].state = TCP_CLOSED;             // A
return -1;                                         // G
}
//_____
_____


//procesar paquete con datos válidos
switch(tcp_socket[socket].state){      ////**** S W I T C H ****////
case TCP_LISTENING:
//verificar que el paquete recibido tenga la bandera SYN
if((tcp_received_packet.hlen_flags & TCP_ACK)==0)   //no contiene ACK
if(tcp_received_packet.hlen_flags & TCP_SYN){       //y sí contiene SYN
//informar a la aplicación
write_string("paquete SYN recibido");
tmp = tcp_socket[socket].application(socket, TCP_EVENT_REQUEST_CONNECTION);
if(tmp>0){ // la aplicación puede aceptar una conexión ahora
write_string("enviar SYN + ACK...");
tcp_socket[socket].remsecnum = tcp_received_packet.secnum + 1;
tcp_socket[socket].secnum = INITIAL_SEC_NUM;
tcp_socket[socket].dlen = 0;
tcp_send(socket,TCP_SYN | TCP_ACK, data_buff);

//ajustes al estado del socket
tcp_socket[socket].secnum++;      //próximo número de secuencia a enviar
tcp_socket[socket].state = TCP_SYN_RECEIVED;
tcp_socket[socket].retries = TCP_RETRIES;
reset_timer_value(tcp_socket[socket].retransmitclock, TCP_TIME_OUT, SEC);
return 1;
}
write_string("La aplicacion no puede aceptar la conexion ahora...");
}

//algo está mal con el paquete recibido o la aplicación no puede aceptar la conexión
write_string("Enviar reset");
tcp_reset(&tcp_received_packet, ip);
return -1;
break;

case TCP_SYN_SENT:
//verificar que el paquete recibido contenga las banderas correctas
if(tcp_received_packet.hlen_flags & TCP_SYN)
if(tcp_received_packet.hlen_flags & TCP_ACK){

//contiene ambas banderas, verificar la confirmación de mi secnum
write_string("SYN + ACK recibido");
```

```
if(tcp_received_packet.acknum != tcp_socket[socket].secnum){
write_string("El numero de secuencia es incorrecto");
return -1; //número de secuencia incorrecto
}

//guardar el número de secuencia remoto para confirmar
tcp_socket[socket].remsecnum = tcp_received_packet.secnum + 1;

write_string("Enviar confirmacion, ... CONECTADO ...");
//enviar la confirmación
tcp_socket[socket].dlen = 0;
tcp_send(socket,TCP_ACK,data_buff);

//ajustar el estado del socket
tcp_socket[socket].state = TCP_CONNECTED;
tcp_socket[socket].retries = TCP_RETRIES;
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
reset_timer_value(tcp_socket[socket].timetoliveclock,TCP_ID
LE_TIME,SEC);

//informar a la aplicación
tcp_socket[socket].application(socket,TCP_EVENT_CONNECT
ED);
return 1;
} else { //el paquete contiene SYN pero no ACK
//confirmar el SYN con SYN + ACK
write_string("SYN recibido, enviar confirmacion");
tcp_socket[socket].remsecnum = tcp_received_packet.secnum + 1;
tcp_socket[socket].secnum = INITIAL_SEC_NUM;
tcp_socket[socket].dlen = 0;
tcp_send(socket, TCP_SYN | TCP_ACK, data_buff);

//ajustes al estado del socket
tcp_socket[socket].secnum++; //próximo número de
secuencia a enviar
tcp_socket[socket].state = TCP_SYN_RECEIVED;
tcp_socket[socket].retries = TCP_RETRIES;
reset_timer_value(tcp_socket[socket].retransmitclock,TCP_TI
ME_OUT,SEC);
return 1;
}

//el paquete no contenía SYN, paquete con información
erronea
tcp_reset(&tcp_received_packet,ip);
return -1;
break;

case TCP_SYN_RECEIVED:
//verificar que el paquete recibido contenga las banderas
correctas
if(tcp_received_packet.hlen_flags & TCP_SYN) {
//el paquete contiene la bandera de SYN, quizá se perdió mi
paquete SYN+ACK
write_string("Reenviar SYN + ACK, mi respuesta no llego");
tcp_socket[socket].remsecnum = tcp_received_packet.secnum + 1;
tcp_socket[socket].secnum = INITIAL_SEC_NUM;
tcp_socket[socket].dlen = 0;
tcp_send(socket, TCP_SYN | TCP_ACK, data_buff);

//ajustes al estado del socket
tcp_socket[socket].data_unack = FALSE;
tcp_socket[socket].secnum++; //próximo número de
secuencia a enviar
tcp_socket[socket].state = TCP_SYN_RECEIVED;
tcp_socket[socket].retries = TCP_RETRIES;
reset_timer_value(tcp_socket[socket].retransmitclock,TCP_TI
ME_OUT,SEC);
return 1;
}

//verificar si sólo contiene la bandera ACK
if(tcp_received_packet.hlen_flags & TCP_ACK){
write_string("Confirmacion de mi paquete SYN + ACK");
if(tcp_received_packet.acknum != tcp_socket[socket].secnum){
write_string("Numero de confirmacion incorrecto");
return -1; //la confirmación es incorrecta
}

write_string("... CONECTADO ...");
//ajustes al socket
tcp_socket[socket].state = TCP_CONNECTED;
tcp_socket[socket].application(socket,TCP_EVENT_CONNECT
ED);
reset_timer_value(tcp_socket[socket].timetoliveclock,TCP_ID
LE_TIME,SEC);
return 1;
}

//algo estubo mal con el paquete recibido
tcp_reset(&tcp_received_packet,ip);
write_string("Existe algun problema, enviando reset");
return -1;
break;

case TCP_CONNECTED:

//verificar si se trata de un paquete SYN+ACK (previamente
//confirmado) cuya confirmación no llegó a su destino
if((tcp_received_packet.hlen_flags & (TCP_SYN | TCP_ACK))
== (TCP_SYN | TCP_ACK))
if(tcp_received_packet.acknum ==
tcp_socket[socket].secnum &&
tcp_received_packet.secnum
==(tcp_socket[socket].remsecnum -1) ){
tcp_socket[socket].remsecnum =
tcp_received_packet.secnum+1;
tcp_socket[socket].dlen = 0;
tcp_send(socket, TCP_ACK, data_buff);
reset_timer_value(tcp_socket[socket].timetoliveclock,
TCP_IDLE_TIME, SEC);
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
tcp_socket[socket].retries = TCP_RETRIES;
return 1;
} else
return -1; // número de secuencia o confirmación
incorrectos
else{
if(tcp_received_packet.hlen_flags & TCP_SYN)
return -1; // es una solicitud de conexión, no se maneja en
este estado (conectado)
}


//verificar si ya se confirmaron los últimos datos enviados
if(tcp_socket[socket].data_unack == TRUE)
if(tcp_received_packet.hlen_flags & TCP_ACK){ // llegó la
confirmación de datos que se esperaba
if(tcp_received_packet.acknum ==
tcp_socket[socket].secnum){
write_string("Llego confirmacion de datos enviados");
tcp_socket[socket].data_unack = FALSE;
tcp_socket[socket].dlen = 0;
tcp_socket[socket].application(socket,TCP_EVENT_ACK);
return 1;
}
} else{
write_string("se recibieron datos, pero aun no llega la
confirmacion de datos enviados");
return -1; //hay datos esperando ser confirmados
}

//verificar si el número de secuencia es correcto
if( tcp_received_packet.secnum!=
tcp_socket[socket].remsecnum){
write_string("Numero de secuencia incorrecto");
tcp_socket[socket].dlen = 0;
```

```
tcp_send(socket,TCP_ACK,data_buff);
return -1;
}

write_string("Datos recibidos para la aplicacion");
//los datos del paquete son correctos
tcp_socket[socket].dlen = dlen;
tcp_socket[socket].application(socket,TCP_EVENT_DATA);
tcp_socket[socket].remsecnum += dlen;  // próximo número
de secuencia remoto que se espera recibir

//verificar si el paquete contenía alguna solicitud de FIN
if(tcp_received_packet.hlen_flags & TCP_FIN){
write_string("Paquete con bandera de FIN");
if(tcp_socket[socket].data_unack == FALSE)
tcp_socket[socket].application(socket,TCP_EVENT_CLOSE);
tcp_socket[socket].remsecnum++;
tcp_socket[socket].dlen = 0;
write_string("Enviar FIN + ACK");
tcp_send(socket,TCP_ACK | TCP_FIN, data_buff);
tcp_socket[socket].secnum++;
tcp_socket[socket].state = TCP_LAST_ACK;
tcp_socket[socket].retries = 1;
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
return 1;
}

//confirmar los datos enviados si había alguno
if(dlen){
write_string("Enviar confirmacion de datos");
tcp_socket[socket].dlen = 0;
tcp_send(socket, TCP_ACK, data_buff);
tcp_socket[socket].retries = TCP_RETRIES;
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
reset_timer_value(tcp_socket[socket].timetoliveclock,
TCP_IDLE_TIME, SEC);
return 1;
}
break;

case TCP_LAST_ACK:
//verificar que el paquete contenga la bandera de ACK
if(tcp_received_packet.hlen_flags & TCP_ACK){
write_string("Llego confirmacion");
if(tcp_received_packet.acknum                    !=
tcp_socket[socket].secnum){
write_string("Numero de confirmacion incorrecto");
return -1;  // no contiene el número correcto de
confirmación
}
write_string("...cerrar conexion...");
if(tcp_socket[socket].type & TCP_SERVER)
tcp_socket[socket].state = TCP_LISTENING;
else
tcp_socket[socket].state = TCP_CLOSED;
return 1;
}

//si llegó de nuevo el fin, se envía FIN+ACK nuevamente
if(tcp_received_packet.hlen_flags & TCP_FIN){
write_string("Llego FIN de nuevo, volver a enviar SIN +
ACK");
tcp_socket[socket].remsecnum                     =
tcp_received_packet.secnum + 1;
tcp_socket[socket].remsecnum += dlen;
tcp_socket[socket].dlen = 0;
tcp_send(socket,TCP_ACK | TCP_FIN, data_buff);
tcp_socket[socket].retries = 1;
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
return 1;
}
break;

case TCP_FIN_WAIT1:
//verificar si el paquete recibido contiene FIN+ACK
if(tcp_received_packet.hlen_flags & TCP_FIN &&
tcp_received_packet.hlen_flags & TCP_ACK){
write_string("Llego FIN + ACK");
if(tcp_received_packet.acknum                    !=
tcp_socket[socket].secnum){
write_string("Numero de confirmacion incorecto");
return -1; //Número de confirmación erroneo
}
write_string("Enviar ACK");
tcp_socket[socket].remsecnum                     =
tcp_received_packet.secnum + 1;
tcp_socket[socket].remsecnum += dlen;
tcp_socket[socket].data_unack = FALSE;
tcp_socket[socket].dlen = 0;
tcp_send(socket,TCP_ACK,data_buff);
tcp_socket[socket].secnum++;
tcp_socket[socket].state = TCP_TIME_WAIT;
tcp_socket[socket].retries = 0;
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
return 1;
}

//verificar si llegó la confirmación del FIN enviado
if(tcp_received_packet.hlen_flags & TCP_ACK){
write_string("Llego confirmacion de FIN");
if(tcp_received_packet.acknum                    !=
tcp_socket[socket].secnum){
write_string("Numero de confirmacion erroneo");
return -1; //Número de confirmación erroneo
}
tcp_socket[socket].data_unack = FALSE;
tcp_socket[socket].state = TCP_FIN_WAIT2;
return 1;
}

//verificar si se recibió un FIN
if(tcp_received_packet.hlen_flags & TCP_FIN){
write_string("Llego FIN, enviar ACK");
tcp_socket[socket].remsecnum                     =
tcp_received_packet.secnum + 1;
tcp_socket[socket].dlen = 0;
tcp_send(socket, TCP_ACK, data_buff);
tcp_socket[socket].secnum++;
tcp_socket[socket].state = TCP_CLOSING;
tcp_socket[socket].retries = 1;
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
return 1;
}
break;

case TCP_FIN_WAIT2:
//verrificar si llegó en FIN
if(tcp_received_packet.hlen_flags & TCP_FIN){
write_string("Llego FIN, enviar ACK");
tcp_socket[socket].remsecnum                     =
tcp_received_packet.secnum + 1;
tcp_socket[socket].remsecnum += dlen;
tcp_socket[socket].dlen = 0;
tcp_send(socket,TCP_ACK,data_buff);
tcp_socket[socket].state = TCP_TIME_WAIT;
tcp_socket[socket].retries = 0;
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
return 1;
} else
return -1;
break;

case TCP_CLOSING:
//verificar si se recibió el ACK
if(tcp_received_packet.hlen_flags & TCP_ACK){
write_string("Llego ACK");
if(tcp_received_packet.acknum                    !=
tcp_socket[socket].secnum){
```

```
write_string("Numero de confirmacion erroneo");
return -1; //Número de confirmación erroneo
}
tcp_socket[socket].data_unack = FALSE;
tcp_socket[socket].state = TCP_TIME_WAIT;
tcp_socket[socket].retries = 0;
reset_timer_value(tcp_socket[socket].retransmitclock,
TCP_TIME_OUT, SEC);
return 1;
}

//fin repetido??
if(tcp_received_packet.hlen_flags & TCP_FIN){
write_string("Llego FIN de nuevo, enviar ACKk de nuevo");
tcp_socket[socket].remsecnum                                    =
tcp_received_packet.secnum + 1;
tcp_socket[socket].remsecnum += dlen;
tcp_socket[socket].dlen = 0;
tcp_send(socket,TCP_ACK,data_buff);
return 1;
}
break;

case TCP_TIME_WAIT:
//verificar si llegó un FIN repetido
if(tcp_received_packet.hlen_flags & TCP_FIN){
write_string("Llego FIN de nuevo, enviar ACKk de nuevo");
tcp_socket[socket].remsecnum                                    =
tcp_received_packet.secnum + 1;
tcp_socket[socket].remsecnum += dlen;
tcp_socket[socket].dlen = 0;
tcp_send(socket,TCP_ACK,data_buff);
return 1;
} else
return -1;
break;

default:
write_string("Enviar reset, paquete desconocidoS");
tcp_reset(&tcp_received_packet,ip);
reset_code();
return -1;


}// fin de switch

}

//rutina de mantenimiento tcp
void tcp_poll(){
char id,tmp,flags, retries;;
//revisar cada estado del socket para realizar acciones de
mantenimiento
for(id=0;id<TCP_SOCKETS_NUM;id++){
switch(tcp_socket[id].state){
case TCP_FREE:
break;
case TCP_CLOSED:
break;
case TCP_LISTENING:
break;
case TCP_CONNECTED:
//verificar si hay una instrucción de cerrar pendiente
if(tcp_socket[id].closepending &&
tcp_socket[id].data_unack == FALSE ){
write_string("cerrando conexion, enviar FIN + ACK");
tcp_send(id, TCP_ACK | TCP_FIN,data_buff);
tcp_socket[id].secnum ++;
tcp_socket[id].state = TCP_FIN_WAIT1;
tcp_socket[id].retries = 1;
reset_timer_value(tcp_socket[id].retransmitclock,
TCP_TIME_OUT, SEC);
return;
}

//si la conexión ha estado inactiva por mucho tiempo, se
cierra
if(read_timer_ms(tcp_socket[id].timetoliveclock)==0){
write_string("conexin inactiva, se cerrara, enviar FIN +
ACK");
tcp_socket[id].dlen=0;
tcp_send(id,TCP_ACK|TCP_FIN,data_buff);
tcp_socket[id].secnum++;
tcp_socket[id].state = TCP_FIN_WAIT1;
tcp_socket[id].retries = 1;
reset_timer_value(tcp_socket[id].retransmitclock,
TCP_TIME_OUT, SEC);
tcp_socket[id].application(id,TCP_EVENT_CLOSE);
return;
}

if(tcp_socket[id].data_unack == FALSE)
return;

//sí hay datos no confirmados, verificar si venció el tiempo
de espera por ellos
if(read_timer_ms(tcp_socket[id].retransmitclock)!=0)
return;
write_string("vencio tiempo de espera de confirmacion");
if(tcp_socket[id].retries==0){
write_string("Se acabaron los intentos de transmision, reinicir
conexion");
//ya no quedan mas intentos de enviar mis datos, reiniciar
tcp_socket[id].dlen = 0;
tcp_send(id,TCP_RST,data_buff);
tcp_socket[id].application(id,TCP_EVENT_ABORT);
if(tcp_socket[id].type & TCP_SERVER)
tcp_socket[id].state = TCP_LISTENING;
else
tcp_socket[id].state = TCP_CLOSED;
return;
}

write_string("Volver a enviar los datos");
//aún quedan intentos de enviar datos
tcp_socket[id].retries--;
retries=tcp_socket[id].retries;
tcp_socket[id].data_unack = FALSE;
tcp_socket[id].secnum-=tcp_socket[id].dlen;
tmp=tcp_socket[id].application(id,TCP_EVENT_RESEND);
tcp_socket[id].retries = retries;
if(tmp<0){
//la aplicación no tiene datos pendientes por enviar
tcp_socket[id].dlen = 0;
tcp_send(id,TCP_RST,data_buff);
tcp_socket[id].application(id,TCP_EVENT_ABORT);
if(tcp_socket[id].type & TCP_SERVER)
tcp_socket[id].state = TCP_LISTENING;
else
tcp_socket[id].state = TCP_CLOSED;
return;
}
return;
break;
case TCP_SYN_SENT:
case TCP_SYN_RECEIVED:
if(read_timer_ms(tcp_socket[id].retransmitclock)!=0)
return;

if(tcp_socket[id].retries==0){
write_string("se acabron intentos de rerstransmision,
reiniciar");
//ya no quedan mas intentos de enviar mis datos, reiniciar
tcp_socket[id].dlen = 0;
tcp_send(id,TCP_RST,data_buff);
tcp_socket[id].application(id,TCP_EVENT_ABORT);
if(tcp_socket[id].type & TCP_SERVER)
tcp_socket[id].state = TCP_LISTENING;
else
tcp_socket[id].state = TCP_CLOSED;
return;
}
```

```
write_string("vencio tiempo de espera, volver a enviar datos");
tcp_socket[id].retries --;
if(tcp_socket[id].state == TCP_SYN_SENT){
flags=TCP_SYN;
reset_timer_value(tcp_socket[id].retransmitclock, TCP_INIT_TIME_OUT, SEC);
}
else{
flags = TCP_SYN|TCP_ACK;
reset_timer_value(tcp_socket[id].retransmitclock, TCP_TIME_OUT, SEC);
}
tcp_socket[id].secnum--;

tcp_socket[id].dlen = 0;
tcp_send(id,flags,data_buff);
tcp_socket[id].secnum ++;
return;
break;
case TCP_TIME_WAIT:
if(read_timer_ms(tcp_socket[id].retransmitclock)!=0)
return;

if(tcp_socket[id].retries){
write_string("se acabron intentos de rerstransmision, reiniciar");
tcp_socket[id].retries--;
reset_timer_value(tcp_socket[id].retransmitclock, TCP_TIME_OUT, SEC);
break;
}

write_string("cerrar conexion");
if(tcp_socket[id].type & TCP_SERVER)
tcp_socket[id].state = TCP_LISTENING;
else
tcp_socket[id].state = TCP_CLOSED;
break;
case TCP_LAST_ACK:
case TCP_FIN_WAIT1:
case TCP_CLOSING:
if(read_timer_ms(tcp_socket[id].retransmitclock)!=0);
return;

if(tcp_socket[id].retries == 0){
write_string("se acabron intentos de rerstransmision, reiniciar");
tcp_socket[id].dlen = 0;
tcp_send(id,TCP_RST,data_buff);
tcp_socket[id].application(id,TCP_EVENT_ABORT);
if(tcp_socket[id].type & TCP_SERVER)
tcp_socket[id].state = TCP_LISTENING;
else
tcp_socket[id].state = TCP_CLOSED;
return;
}

write_string("vencio tiempo de espera, volver a enviar FIN + ACK");
tcp_socket[id].retries--;
tcp_socket[id].dlen = 0;
tcp_socket[id].secnum--;
tcp_send(id,TCP_ACK|TCP_FIN,data_buff);
tcp_socket[id].secnum++;
reset_timer_value(tcp_socket[id].retransmitclock, TCP_TIME_OUT, SEC);
return;
break;

case TCP_FIN_WAIT2:
if(read_timer_ms(tcp_socket[id].retransmitclock)!=0);
return;

if(tcp_socket[id].retries == 0){
write_string("vencio tiempo de espera, enviar RST");
tcp_socket[id].dlen = 0;
tcp_send(id,TCP_RST,data_buff);
tcp_socket[id].application(id,TCP_EVENT_ABORT);
if(tcp_socket[id].type & TCP_SERVER)
tcp_socket[id].state = TCP_LISTENING;
else
tcp_socket[id].state = TCP_CLOSED;
return;
}

tcp_socket[id].retries--;
reset_timer_value(tcp_socket[id].retransmitclock, TCP_TIME_OUT, SEC);
return;
break;
default:
break;

} // fin del switch
}  // fin del for
}

//buscar un socket que contenga los datos recibidos
_____
char tcp_find_sock(struct tcp_packet *tcp_pack, unsigned char *remip){
unsigned char i;
unsigned int flags;

for(i=0;i<TCP_SOCKETS_NUM;i++){
if(tcp_socket[i].state == TCP_FREE || tcp_socket[i].state == TCP_CLOSED || tcp_socket[i].state == TCP_LISTENING)
continue;  // socket que no contiene datos válidos
if(!equal_ip(tcp_socket[i].remip,remip))
continue;
if(tcp_socket[i].remport != tcp_pack->localport || tcp_socket[i].localport != tcp_pack->remport)
continue;
//se encontró el socket con todos los datos de la conexión
return i;
}

//si no era ninguno de los socket con estado de conexión, verificar los sockets en estado listening

if(!(tcp_pack->hlen_flags & TCP_SYN))
return -1; // el paquete no es una solicitud de conexión

//si es una solicitud de conexión, verificar que no contenga banderas inválidas
flags = TCP_ACK | TCP_RST | TCP_FIN;
if(tcp_pack->hlen_flags & flags)
return -1; //alguna de las banderas está puesta

// buscar un socket en listening esperando una solicitud de conexión

for(i=0;i<=TCP_SOCKETS_NUM;i++){
if(tcp_socket[i].state != TCP_LISTENING)
continue;
if(tcp_socket[i].localport == tcp_pack->remport){  //lo encontramos
tcp_socket[i].remport = tcp_pack->localport;
tcp_socket[i].remip[0] = *remip++;
tcp_socket[i].remip[1] = *remip++;
tcp_socket[i].remip[2] = *remip++;
tcp_socket[i].remip[3] = *remip++;
return i;
}
}

//no sse encontró el socket
return -1;
}
```

```
//envío de una trama de reset
_____
void tcp_reset(struct tcp_packet *tcp_pack, unsigned char *remip){

//si el paquete que recibo es un paquete de reset, no se puede enviar otro reset
if(tcp_pack->hlen_flags & TCP_RST)
return;

//llenar un socket con los datos necesarios
tcp_socket[TCP_SOCKETS_NUM].remip[0] = *remip++;
tcp_socket[TCP_SOCKETS_NUM].remip[1] = *remip++;
tcp_socket[TCP_SOCKETS_NUM].remip[2] = *remip++;
tcp_socket[TCP_SOCKETS_NUM].remip[3] = *remip++;
tcp_socket[TCP_SOCKETS_NUM].remport = tcp_pack->localport;
tcp_socket[TCP_SOCKETS_NUM].localport = tcp_pack->remport;

//puede que el paquete no contenga número de confirmación, por ejemplo si era una solicitud de conexión
if(tcp_pack->hlen_flags & TCP_ACK)
tcp_socket[TCP_SOCKETS_NUM].secnum = tcp_pack->acknum;
else
tcp_socket[TCP_SOCKETS_NUM].secnum = 0;

tcp_socket[TCP_SOCKETS_NUM].remsecnum = tcp_pack->secnum+1;
tcp_socket[TCP_SOCKETS_NUM].dlen = 0;

tcp_send(TCP_SOCKETS_NUM,TCP_ACK | TCP_RST, data_buff);

}

//cierre de una conexión por parte de una aplicación
_____
/* valores de retorno:
-2 --> no se puede cerrar la conexión porque espera una confirmación
-1 --> los datos del socket no son correctos
1 --> la conexión se cerró exitosamente
2 --> se envió una solicitud de desconexión, se espera por la confirmación
*/
char tcp_close(char id){
if(id>=TCP_SOCKETS_NUM || id < 0)
return -1;

if(tcp_socket[id].state > TCP_LAST_ACK)
return -1;

switch(tcp_socket[id].state){

case TCP_LISTENING:
tcp_socket[id].state = TCP_CLOSED;
return 1;
break;
case TCP_SYN_SENT:
tcp_socket[id].state = TCP_CLOSED;
return 1;
break;
case TCP_SYN_RECEIVED:
tcp_socket[id].dlen = 0;
tcp_send(id,TCP_ACK|TCP_FIN,data_buff);
tcp_socket[id].secnum++;
tcp_socket[id].state = TCP_FIN_WAIT1;
tcp_socket[id].retries = 1;
reset_timer_value(tcp_socket[id].retransmitclock, TCP_TIME_OUT, SEC);
return 2;
break;
case TCP_FIN_WAIT1:
case TCP_FIN_WAIT2:
case TCP_TIME_WAIT:
case TCP_LAST_ACK:
case TCP_CLOSING:
return 2;
break;
case TCP_CONNECTED:
//verificar si ya se recibió confirmación de datos enviados
if(tcp_socket[id].data_unack == FALSE){
write_string("no hay datos pendientes por confirmar, se cerrara la conexion");
tcp_socket[id].dlen = 0;
tcp_send(id,TCP_ACK|TCP_FIN,data_buff);
tcp_socket[id].secnum++;
tcp_socket[id].state = TCP_FIN_WAIT1;
tcp_socket[id].retries = 1;
reset_timer_value(tcp_socket[id].retransmitclock, TCP_TIME_OUT, SEC);
return 2;
} else{
write_string("Aun no se puede cerrar la conexion porque no ha llegado la confirmacion");
tcp_socket[id].closepending = TRUE;
return -2;
}
break;
}

}

//abortar la conexión
_____
char tcp_abort(char id){

if(id>=TCP_SOCKETS_NUM || id < 0)
return -1;

if(tcp_socket[id].state > TCP_LAST_ACK)
return 0;

if(tcp_socket[id].state == TCP_FREE || tcp_socket[id].state == TCP_CLOSED)
return 0;

if(tcp_socket[id].state == TCP_LISTENING || tcp_socket[id].state == TCP_TIME_WAIT){
tcp_socket[id].state = TCP_CLOSED;
return 1;
}

tcp_socket[id].dlen = 0;
tcp_send(id,TCP_RST,data_buff);
tcp_socket[id].state = TCP_CLOSED;

return 1;
}

//añadir los datos del pseudo encabezado a la suma de comprobación
_____
unsigned int add_pseudoheader_to_cs(unsigned int cs, unsigned char *ip,unsigned int len){
unsigned int protocol=TCP_ETYPE;

cs=~cs;

cs=ip_add_to_checksum(cs,*((unsigned int *)(&ip_address[0])));
cs=ip_add_to_checksum(cs,*((unsigned int *)(&ip_address[2])));
cs=ip_add_to_checksum(cs,*((unsigned int *)(&ip[0])));
cs=ip_add_to_checksum(cs,*((unsigned int *)(&ip[2])));
cs=ip_add_to_checksum(cs,protocol);
cs=ip_add_to_checksum(cs,len);

cs=~cs;
```

```
return cs;
}
```

PROTOCOLO HTTP

http.h

```
//Número de sesiones manejadas por el servidor HTTP
#define HTTP_SESSIONS_NUM  3

//número de puerto usado por el servidor HTTP
#define HTTP_PORT          80

//estados de las sesiones HTTP
#define HTTP_CLOSED    1
#define HTTP_OPEN      2


//información sobre cada sesion HTTP
struct session_state{

unsigned char state;
unsigned int  socket;
unsigned char *filestart;
unsigned long filelen;
unsigned long pointer;
unsigned long datasent;
};

//prototipos de funciones
char http_init_server(void);
void http_run(void);
char http_listener(char, unsigned char);
void http_close_session(unsigned int);
void http_open_session(unsigned int);
int http_find_session(char);
void http_send(unsigned int);
char search_resource(unsigned char);
unsigned char arraylen (const struct files *);
unsigned char https_read_encoded(void);
unsigned char https_calculatehash (unsigned long);
```

http.c

```
#include "ne64api.h"
#include "tcp.h"
#include "http.h"
#include "server.h"
#include "general_functions.h"
#include "index.h"
#include "hola.h"
#include "not_found.h"

struct session_state http_session[HTTP_SESSIONS_NUM];
extern struct tcp_sock tcp_socket[TCP_SOCKETS_NUM+1];

//inicializar servidor http
_____
char http_init_server(void){

unsigned char i;
char sock;

for(i=0;i<HTTP_SESSIONS_NUM;i++){
http_session[i].state = HTTP_CLOSED;
sock=tcp_get_sock(TCP_SERVER, http_listener);
if(sock<0)
reset_code(); //ya no hay sockets disponibles
http_session[i].socket = sock;
sock = tcp_sock_listen(sock,HTTP_PORT);
if(sock<0)
reset_code(); //error, no se pudo poner a escuchar el socket
} //fin del for

}

// **********  funciones de manejo de sesiones http   **********
//

//cerrar una sesión
_____
void http_close_session(unsigned int id){

if(id >=HTTP_SESSIONS_NUM)
return;
http_session[id].state = HTTP_CLOSED;
http_session[id].filestart = 0;
http_session[id].filelen = 0;
http_session[id].pointer = 0;
tcp_sock_listen(id,HTTP_PORT); // se pone a escuchar de nuevo
}

//abrir una sesión
_____
void http_open_session(unsigned int id){

if(id >=HTTP_SESSIONS_NUM)
return;
http_session[id].state = HTTP_OPEN;

}

//encontrar una sesión asociada a un socket
_____
int http_find_session(char sock){

unsigned char i;

for(i=0;i<HTTP_SESSIONS_NUM;i++)
if(http_session[i].socket == sock)
return i;

return -1;
}

//rutina http
_____
void http_run(void){

unsigned int i;
unsigned char tmp;

//checar cada una de las sesiones
for(i=0;i<HTTP_SESSIONS_NUM;i++){

if(http_session[i].state != HTTP_OPEN)
continue;

if(http_session[i].datasent!=0)
continue;

if(http_session[i].filestart == 0)
continue;

if(http_session[i].pointer >= http_session[i].filelen){
write_string("El recurso se termino de transferir, cerrar conexion");
//el archivo se ha terminado de transferir, terminar la comunicación
tcp_close(http_session[i].socket);
//tcp_abort(http_session[i].socket);
http_close_session(i);
continue;
}

write_string("enviando datos del recurso");
//todavía hay datos pendientes por enviar
http_send(i);

} // fin del for (para cada una de las sesiones)
```

```
}

//envío de datos http
_____
void http_send(unsigned int id){

unsigned int k, j=0, i=TCP_HEADER_OFFSET,len;
unsigned char message404[] = "HTTP/1.0 404 Not found";
unsigned char messageOK [] = "HTTP/1.0 200 OK";

if(id>=HTTP_SESSIONS_NUM)
return;

if(*(http_session[id].filestart)==0)
return; //la sesión no contiene un archivo a transferir

if(http_session[id].filelen == 0)
return;

if(http_session[id].pointer >= http_session[id].filelen)
return;

//si se está entregando el principio del archivo, se añade la
línea de estado
if(http_session[id].pointer == 0){
write_string("Comenzando a transferir el archivo");

if(http_session[id].filestart == not_found_file){
write_string("Recursoo no encontrado, enviar pagina de
error");
for(j=0;j<22;j++,i++)
data_buff[i] = message404[j];
}
else{
for(j=0;j<15;j++,i++)
data_buff[i] = messageOK[j];
}

data_buff[i]=0x0D;  i++; j++;
data_buff[i]=0x0A;  i++; j++;
data_buff[i]=0x0D;  i++; j++;
data_buff[i]=0x0A;  i++; j++;

}

//porción del archivo que se va a transferir
len = http_session[id].filelen - http_session[id].pointer;

if((len + j + TCP_HEADER_OFFSET ) > DATA_BUFF_LEN)
len = (DATA_BUFF_LEN - TCP_HEADER_OFFSET - j);

//terminar de llenar el buffer con los datos del objeto a
transferir
for(k=0;k<len;i++,k++)
data_buff[i]=http_session[id].filestart[http_session[id].pointer
+k];

//agregar a la longitud el tamaño de la línea de estado para
calcular la
//longitud total a transferir
len += j;

//cantidad de datos del buffer que es posible enviar
len = tcp_dlen(len);

//enviar a través de TCP
if(tcp_data_send(http_session[id].socket, data_buff, len)){
//quitar a la longitud de datos enviados la cantidad de bytes
de la primera línea
len -= j;
http_session[id].datasent = len; //datos del archivo enviados
}
else{
tcp_close(http_session[id].socket);
http_close_session(id);
}

}

// función de recepción de datos http
_____
char http_listener(char socket, unsigned char event){

int ses, i;
unsigned int dlen;

if(socket>=TCP_SOCKETS_NUM || socket < 0)
return -1;

//buscar la sesión a la que le corresponden los datos
recibidos
ses = http_find_session(socket);

if(ses < 0)
return -1;

dlen = tcp_socket[socket].dlen;

switch(event){

case TCP_EVENT_REQUEST_CONNECTION:
return 1;
break;

case TCP_EVENT_CONNECTED:
http_open_session(ses);
return 1;
break;
case TCP_EVENT_CLOSE:
http_close_session(ses);
return 1;
break;
case TCP_EVENT_ABORT:
http_close_session(ses);
return 1;
break;
case TCP_EVENT_ACK:
http_session[ses].pointer+=http_session[ses].datasent;
http_session[ses].datasent=0;
break;
case TCP_EVENT_DATA:
//verificar si es una petición de algún recurso
if(http_session[ses].filestart == 0){
if(NE64ReadByte() != 'G')
return;
if(NE64ReadByte() != 'E')
return;
if(NE64ReadByte() != 'T')
return;

//sí corresponde a una solicitud
dlen-=3;
//buscar el inicio del nombre del recurso (después del
caracter /)

for(i=0; i<dlen; i++){
if(NE64ReadByte() == '/'){
i++;
break;
}
}

dlen-=i;

//si no se especifica ningún recurso, entregar la página de
inicio
if(NE64ReadByte()==' '){
http_session[ses].filestart = index_file;
http_session[ses].filelen = INDEX_FILE_LEN;
http_session[ses].pointer = 0;
```

```
http_session[ses].datasent =0;
return;
}

NE64InitializeOffsetToReadRxBuffer(59);

//buscar el recurso solicitado
i=https_calculatehash(dlen);
if(i<0)
return -1;

i=search_resource(i);

http_session[ses].filestart = resources[i].source_file;
http_session[ses].filelen = resources[i].file_len;
http_session[ses].pointer = 0;
http_session[ses].datasent =0;
return;

}
break;
case TCP_EVENT_RESEND:
if(http_session[ses].state == HTTP_CLOSED)
return -1;
http_send(ses);
break;
default:
return -1;

} // fin switch

}


// encontrar un recurso solicitado y asociarlo a una sesión
_____
char search_resource(unsigned char hash){
unsigned char len,i;

len=arraylen(resources);
for(i=0;i<len;i++)
if(resources[i].hash == hash)
return i;

//no se encontró la llave del recurso solicitado
return 0; // índice de la página NOT FOUND
}

// cálculo de la cantidad de recursos almacenados en el
servidor _____
unsigned char arraylen (const struct files *fl){
unsigned char count=0;

while(fl->hash && fl->source_file && fl->file_len){
count++;
fl++;
}

return count;
}

/* las siguientes funciones son usadas para calcular la llave
de acceso a un recurso almacenado en el
servidor, el código se tomó del stack OpenTCP de freescale, y
corresponde a un algoritmo para realizar
el cálculo de la llave a partir del nombre del recurso,
considerando el caso en el que el cliente envie
la información codificada */

/* read two encoded bytes from HTTP URI and return
decoded byte */

unsigned char https_read_encoded(void)
{
unsigned char temp,ch;

temp = NE64ReadByte();
if ((temp>='0') && (temp<='9'))
ch = (temp-'0')<<4;
else
if ((temp>='a') && (temp<='f'))
ch = (temp-'a'+10)<<4;
else
ch = (temp-'A'+10)<<4;

temp = NE64ReadByte();
if ((temp>='0') && (temp<='9'))
ch |= (temp-'0');
else
if ((temp>='a') && (temp<='f'))
ch |= (temp-'a'+10);
else
ch |= (temp-'A'+10);

return ch;
}

// función que realiza el cálculo de la llave
_____
_____

unsigned char https_calculatehash (unsigned long len)
{
unsigned char hash=0, ch, i;


/* Read Max 60 characters */
if(len > 60)
len = 60;

for( i=0; i<len; i++){
ch = NE64ReadByte();

if ( ch ==' ')          /* End reached?                    */
break;

/* encoded HTTP URI ? */
if ( ch == '%')
{
ch = https_read_encoded();

/* is this UNICODE char encoded? (for now allow only
* one byte encoded UNICODE chars)
*/
if ( ( ch & 0xe0 ) == 0xc0 )
{
/* yes */
ch = ( ch & 0x1F ) << 6;
(void)NE64ReadByte();          /* skip first % */
ch |= (https_read_encoded() & 0x3F);
}
}

hash *= 37;
hash += ch;

} // fin for

/* Now we have hash value calculated                    */

return( hash );

}
```

Apéndice G
Estimación económica del sistema de desarrollo NE64

Tabla G.1 Costos de material por tarjeta.

Concepto	Costo [$]	Observaciones
Material de tarjeta	816.40	Revisar Tabla D-1 Listado de componentes
Cables USB Tipo A-tipo B	15.00	Adquirido en AG electronica
Cable de RED	30.00	Adquirido en AG electronica
Soldadura y pasta	25.00	Adquirido en Steren
Tablilla Fenolica	13.60	Adquirido en AG electronica
TOTAL	900.00	El costo es sujeto a variaciones según el cambio de dólar y precios de cada distribuidora Electrónica.

Tabla G.2 Costo por cada tarjeta de desarrollo

Concepto	Costo [$]	Observaciones
Material de la tarjeta de desarrollo	900.00	Según Tabla G-1 Costo de material por tarjeta.
Mano de Obra con pruebas	600.00	Armado de tarjeta, revisiones de componentes y empacado.
TOTAL	1500.00	El costo es sujeto a cambios, dependiendo de los costos en material.

Tabla G.3 Estimación económica del costo general del proyecto.

Concepto	Costo [$]	Observaciones
Material de la tarjeta de desarrollo (prototipo final)	900.00	Según Tabla D-1 Listado de componentes utilizados en la tarjeta de desarrollo, costo aproximado por tarjeta.
Diseño de tarjeta esquemático y circuito impreso.	800.00	20 horas diseño en prótel, basado en diseños previos realizados durante las pruebas.
Fabricación de tarjeta (sistema de desarrollo NE64)	800.00	20 horas armado de tarjeta y pruebas de componentes.
Tarjetas adicionales (Considerando componentes, diseño y fabricación)	500.00	Se diseñaron 3 tarjetas adicionales. 1ra.-Tarjeta de LCD (16x2) y teclado matricial 4x4 2da.-Tarjeta de sensores (sensor de temperatura, fotorresistencia) 3.-Tarjeta de control brazo mecánico (K-680, Sterem). Costo aproximado de $350.00 en material de las 3 tarjetas.
Prototipos preliminares (Considerando: componentes, diseño y fabricación)	3000.00	Se crearon 2 tarjetas previas al prototipo final "sistema de desarrollo NE64", la 1era contaba con componentes de montaje superficial y la segunda era similar al prototipo final exceptuando los relevadores y optoacopladores. Compra de circuitos varios, para pruebas de funcionalidad y mejora, como: compuertas lógicas y transformador de pulsos E2023. Costo aproximado de material de $950.00 por tarjeta. El resto del monto corresponde al tiempo de diseño y fabricación.
Material para fabricación de prototipos	1500.00	Se adquirió: cautín Weller, puntas, base de cautín con esponja, soldadura, flux, taladro pequeño con brocas para perforación de tableta fenolica, kit para creación de circuitos impresos, pinzas para cable de red, conectores RJ45, cable UTP cat5.

Investigación e implementación de software de pila TCP/IP y funciones de usuario.	72000.00	Inversión de tiempo en la investigación de temas como: - Especificaciones del microcontrolador MC9S12NE64 -Especificaciones del convertidor USB-Serie -Conceptos generales de redes -Modelo de comunicación TCP/IP -Programación de alto nivel (lenguaje C) orientado a microcontroladores. -Programación de Socket con PHP. El monto se obtiene considerando un tiempo aproximado de 1800 horas en: investigación, programación de la pila de TCP/IP, programación de aplicación web, pruebas sobre los prototipos y programación de la aplicación de usuario para el prototipo final.
TOTAL	79500.00	

- *Costo por hora considerado de 40 pesos.