An S-7800A/PIC16F877 Journey

Fred Eady

Part 2: Reviving It Up

Continuing on his journey, Fred forges a path all the way through to using the S-7800A/PIC16F877 Internet Engine as a web server. With an everyday PIC, a C compiler, a tiny firmware protocol stack, and some common components, he shows us how to put them on the Internet. As Fred says, "(There's) light at the end of the Internet tunnel!"

A second potential problem involved the boot code vector area. If for any reason the Tera Term Pro download failed, the boot code vector for the failed program would be retained in the NOPs area. If the S-7600A/PIC16F877 Internet Engine was allowed to restart, it would vector to nonexistent or corrupted code. To avoid this, I rewrite the boot code vector area with NOPs if the download fails. This allows the boot loader to loop and wait until valid code and a valid boot code vector are loaded. I also cleaned up the boot loader listing to make it easier to follow. The results can be seen in Listing 1.

TERA TERM PRO

This terminal emulator adds class and a bit of automation to the boot loader PIC code. I assembled a simple macro program to run under Tera Term Pro (see Listing 2), which guides you through the S-7600A/PIC16F877 Internet Engine file upload process. Basically, the Tera Term Pro macro [upload.ttl] connects to the S-7600A/PIC16F877 Internet Engine via a personal computer’s COM2 serial port and uploads code to the PIC16F877 by way of the boot loader code.

Let’s analyze the macro beginning with the definitions area. Tera Term Pro uses message boxes that look like their standard Windows counterparts to communicate with you. The boxtitle defines the text that resides in the title area of the message box. The

Photo 1—Note the box title. It was all done without having to write a single byte of Windows code.
rest of the definitions in this area are the wording that will appear in message boxes called from various parts of the macro. The only exception is uploadfile, which is actually the Intel hex file that is loaded and executed. In my macro, this file is called seiko.hex and is actually the web server code.

Moving on to the main macro code area, you find the label :begin followed by a connect command. The /C-2 tag says connect to the personal computer’s COM2 serial port. The connect command returns a result code. If the result code is 0, a link to Tera Term Pro has not been made. A result code of 1 means the link to Tera Term Pro is good but there is no connection to the host. You want a result code of 2, which tells you that the link to Tera Term Pro and the connection to the host are up.

Result codes of 0 and 1 trigger a messagebox event that informs you of the error and disconnects Tera Term Pro. A result code of 2 causes the macro code to branch to the :goodlink label. A yesnobox is triggered asking if you would like to upload code to the S-7600A/PIC16F877 Internet Engine.

If you glance at the boot loader listing, you’ll see that an ASCII “u” is being executed and is ready to communicate with other components of the hardware to you. and wait for the outcome of the upload process. The S-7600A/PIC16F877 Internet Engine boot loader code will send an ASCII “g” if the upload is good and an ASCII “E” if any of the checksum calculations don’t match the original line-by-line checksums.

As you can see in the macro listing, a “g” is result code 1 and the “E” is result code 2. This is determined by the order of the ASCII characters behind the wait command. A no answer to the upload question simply keeps Tera Term Pro connected and nothing else. The boot loader will time out and attempt to run the boot vector code.

**ROAD TEST**

Now that you know what the Tera Term Pro macro is supposed to do, let’s put the code to the test.

Photo 1 is the result of a good connection between Tera Term Pro and the S-7600A/PIC16F877 Internet Engine. The yesnobox command was executed and the uploadprompt message is displayed. Let’s select Yes.

The macro branches to :loadcode and immediately executes a send command that transmits an ASCII “u.” At this point, the boot loader code listens for characters coming into the PIC’s serial port. Tera Term Pro issues a sendfile command that transfers the seiko.hex file to the S-7600A/PIC16F877 Internet Engine. The “0” tag on the sendfile command tells Tera Term Pro to send the file as is without modification of the carriage return/line feed sequences.

Photo 2 is what you’ll see. Serialtest Async gives you a view of what the S-7600A/PIC16F877 Internet Engine and the Tera Term Pro components are doing in Photo 3.

When all of the data in the Intel hex file seiko.hex is transferred to the S-7600A/PIC16F877 Internet Engine, an upload complete message box is generated by Tera Term Pro. Click “OK” and Photo 4 is the result. The uploaded code is being executed and is ready to communicate with other components on the S-7600A/PIC16F877 Internet Engine. Before I move on to this phase, I’ll impart some knowledge of the hardware to you.

**THE HARDWARE**

The heart of the S-7600A/PIC16F877 Internet Engine is the S-7600A. The S-7600A contains all of the necessary hardware and firmware to implement Internet protocols such as TCP/IP, UDP, and PPP. In addition to 10 KB of on-chip RAM to support the protocol stack, the S-7600A has its own UART. Support circuitry for the S-7600A consists of a 74HC4040 that divides the incoming 7.3728-MHz processor clock by 32 and a Sipex SP3243E RS-232 converter.

Data and command information is supplied to the S-7600A by a Microchip PIC16F877 microcontroller. The S-7600A can take information from the PIC serially or with an 8-bit parallel configuration. Because the PIC16F877 has a wealth of I/O, I chose the parallel attachment method. The PIC is clocked at 7.3728 MHz and provides clocking for the S-7600A through the 74HC4040, which presents 230 KHz to the S-7600A’s clock input. A Sipex SP3243E RS-232 converter IC that allows the PIC’s UART to interface with Tera Term Pro run-
ning on a PC also supports the PIC16F877.

Boot loader code is initially loaded onto the PIC16F877 using a standard PIC programmer and a header/connector ribbon cable assembly like the one shown in Photo 5. My PIC programmer uses wide Aries ZIF sockets so I can make my programming jig using a standard 0.600 header. The connector you end up with depends on your PIC programmer setup.

The important thing is to have the necessary signals and voltages between the header and S-7600A/PIC16F877 Internet Engine for programming the PIC16F877. The business end of the in-circuit programming connector on the S-7600A/PIC16F877 Internet Engine is shown in Photo 6.

If I used the space wisely, the PIC16F877 has enough on-chip data EEPROM to accommodate the parameters needed to make a connection to the Internet. However, there wouldn’t be enough space left to put a decent HTML image in this memory area. So, a Microchip 24LC256 EEPROM is employed to hold all of the text necessary to connect to the Internet and serve an HTML-based page.

Just to make things interesting, I decided to throw in a Dallas DS1629 time and temperature IC. All the DS1629 needs to operate is a standard 32.768-KHz crystal and some room on the FC bus. The DS1629 comes hard addressed and uses the extra pins that would normally be address lines for crystal and alarm connections. The 24LC256 is addressed as 0x00 and the DS1629 is addressed at 0x07. The DS1629 clock is volatile and must be backed up with a 3-V lithium cell if you expect to lose power to your S-7600A/PIC16F877 Internet Engine after it is deployed. And, speaking of power, a National LM2937 3.3-V regulator and a standard 9-VDC power brick powers the whole thing. You can see how it all fits together by perusing the schematic shown in Figure 1. The real McCoy is shown in Photo 7.

AS A WEB SERVER

The S-7600A’s protocol stack allows the S-7600A/PIC16F877 Internet Engine to play many roles. In addition to a web server, the S-7600A/PIC16F877 Internet Engine can be configured as a TCP/IP client or an e-mail generator. In this section, I’m going to show you how the S-7600A/PIC16F877 Internet Engine can be programmed to serve a simple web page.

The process begins with the menu you see in Photo 4. After the username, password, and ISP phone number are entered, the date and time can be set and an HTML page can be loaded into the serial EEPROM. I’ve supplied a sample HTML page in Listing 3. The time and date are taken from the DS1629 and inserted into the page just before it is served.

Listing 4 is the code necessary to implement the web server application on the S-7600A/PIC16F877 Internet Engine. Let’s start at the top and work our way down. As an experienced PIC programmer, I found the PCW C package refreshing. The engineers at Microchip were good enough to provide most of the supporting include code that does the housework. All of the EEPROM, S-7600A, and PIC16F877 include files were provided by Microchip. So, building up the S-7600A/PIC16F877 Internet Engine and the accompanying software was like working with Lego blocks. The remainder of include files are the standard C includes with PIC accents that come with the PCW compiler.

I know how to write all of those PIC I/O ports. I'm using PORTA as a digital I/O port, all of the analog capability of PORTA is disabled. Timer1 is
enabled for 8-µs cycles and used as a general-purpose connection timer. The clock and data lines for the EEPROM and DS1629 are floated, and the DS1629 is instructed to keep time and wait for requests for temperature from the PIC16F877 before beginning a temperature conversion.

If the username, password, phone number, and HTML have been entered previously, a push of the Esc key performs a hard reset on the S-7600A and a DTR reset on the external US Robotics 28.8 modem. The DTR reset is performed by writing 0x06 to the S-7600A's serial port configuration register. The next task is to program the S-7600A internal clock divider to obtain a 1-KHz internal clock that is used for S-7600A internal affairs. The clock divider register contents are obtained by the following formula:

\[
\text{clock frequency} \times \frac{1}{1 \text{KHz}} - 1 = \text{divide count}
\]

The clock frequency is the actual frequency supplied to the S-7600A. In this case, that is 7.3728 MHz through a 74HC4040 that divides by 32, which equates to approximately 230 KHz. Doing the math, that gives us a divide count of 0x00E5.

**Making Changes**

In Part 1, I mentioned that I would like to eventually place a Cermetek modem module at the S-7600A UART interface. The Cermetek modem of choice is the CH1786ET, which runs at 2400 bps maximum. The reasons for choosing 2400 bps are:

- The S-7600A/PIC16F877 Internet Engine won’t be sending large chunks of data.
- 2400 bps takes little time to establish ISP connectivity compared to 56-kbps modems.
- The Cermetek modem runs at this speed.
- You can use almost any external modem for initial testing.

So, the S-7600A baud rate divisor can be calculated like this:

\[
\text{clock frequency} \times \frac{1}{1 \text{KHz}} - 1 = \text{data transfer rate} - \text{divisor value of} 0x005F
\]

Most of the time, the modem signals are rigged to simulate desired signal levels by crossing active pins with pins expecting activity on the DB shell connector. I chose to implement the S-7600A’s modem control signals correctly because I may want to control data flow in the standard manner. The S-7600A serial port configuration register allows me to directly control DTR and RTS. There’s also a bit in the serial port config register that determines who has control of the S-7600A UART and serial port.

At this point, you aren’t ready to give control over to the S-7600A. The first order of business is to dial the ISP and do the PPP thing. So, 0x00 is sent to the S-7600A serial port configuration register and DTR and RTS are activated. The US Robotics modem has status LEDs and when the WriteSeiko(Serial_Port_Config,0x00) line is executed, LEDs on the modem for DTR and RTS illuminate. Nothing fancy is needed as far as modem configuration is concerned, so an ATetF is sufficient to set the modem straight.

**Ready...Set?**

The modem is ready, but before you fire off any phone numbers, the S-7600A must be told that PPP mode will be used and the authentication method will be PAP. Then all of the data entered into the EEPROM via the Menu function in Photo 4 (with the exception of the phone number) is retrieved and placed into the appropriate S-7600A registers. The username and password are written into the S-7600A PAP string register. The length of the field precedes the actual data. For instance, if the username is Fred, a 0x04 is written to the PAP register first followed by Fred in consecutive ASCII characters. When all of the user data is entered, a final NULL character [0x00] is written to the PAP string register.

The next step is to issue the ATDT modem command followed by the ISP phone number entered earlier. The PIC16F877 has control of the S-7600A UART and is responsible for issuing the dial command. With some extra command work up front, you can configure the US Robotics modem to give you result codes during the dial process. I chose to simply wait for a connection, as the US Robotics modem has a built-in speaker and I won’t be putting this S-7600A/PIC16F877 Internet Engine out in the real world. I arbitrarily set the wait time for 20 s, which should be long enough for a connection to be established at 2400 bps. You should write some code to track modem and line status if you plan to deploy your S-7600A/PIC16F877 Internet Engine in the field.

**Go**

After the PIC16F877 is notified that the physical connection to the ISP is established, the PIC is responsible for passing the ball to the S-7600A. This is done by setting the Connection Valid bit in the S-7600A PPP control and status register. Setting this bit tells the network stack that the layer below it is up and operational. The Use PAP and PPP enabled bits also reside in the PPP control and status register and are set with the same command used to signal...
a good connection to the ISP. If you follow WWII movies, this is where the pilot of the B-17 turns over control to the guy with the Norden bombsight. The PIC sets the SCTL bit in the serial port configuration register and gives control of the S-7600A UART to the Seiko IC and its network stack mechanism.

This is the point in the process where the S-7600A performs its magic. I’ve taken a full snapshot of the process using Serialtest Async. I’m providing it to you as a readable file so you can follow through the PPP negotiation sequence frame by frame, beginning to end. The only information I’ll censor is my logon password. I’m also including an Acrobat file that contains the Serialtest Async frame decodes. Using the ASCII file in conjunction with the PDF file will illustrate the total PPP process.

Bit 0 of the PPP control and status register confirms to the PIC16F877 that PPP is up and operating. Your ISP assigned IP address is negotiated and the DS1629, the S-7600A/PIC16F877 Internet Engine’s PIC16F877 code now entered the web server loop area. The code here checks PPP up status and the condition of the modem’s DCD pin to determine if the link is up and functioning. The S-7600A is capable of supporting two sockets. This web server application uses Socket 0. Because the S-7600A/PIC16F877 Internet Engine is serving, Port 80 is loaded into the S-7600A, the server mode is activated, and Socket 0 is brought online.

If no timeouts occur and the link remains active, a request from a remote web browser can be fielded. S-7600A socket status is interrogated to ensure that a good connection exists and that the TCP state is listening. If a request is received from a web browser, an HTTP header is constructed and transmitted followed by the HTML stored in the 24LC256 EEPROM. Time and temperature are obtained from the DS1629 and inserted into the HTML text. The assembled HTML page is then sent to the S-7600A’s socket data register. The data is sent and the PIC16F877 waits for the transmit to complete. The socket is then closed and reopened making it ready to serve yet another page of HTML like the one shown in Photo 9.

I’ve shown you how to take an everyday PIC, a C compiler, a tiny firmware protocol stack, and a handful of common components and put them on the Internet. For those of you who wish to build and experiment with your own S-7600A/PIC16F877 Internet Engine, I’ll post the details of how to purchase the kit at www.edtp.com.

Fred Eady has more than 20 years of experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

Thanks to Mike Garbutt, Stephen Humberd, and Rodger Richey for the excellent PIC application notes (Microchip AN731 and AN732) that I used as models for the S-7600A/PIC16F877 Internet Engine.

Sources

SP3243E  
Sipex Corp.  
(978) 667-8700  
Fax: (978) 670-9001  
www.sipex.com

PIC16F877 and 24LC256  
Microchip Technology, Inc.  
(888) 628-6247  
(480) 786-7200  
Fax: (480) 899-9210  
www.microchip.com

DS1629  
Dallas Semiconductor  
(972) 371-4448  
Fax: (972) 371-3715  
www.dalsemi.com

LM2937  
National Semiconductor  
(408) 721-5000  
Fax: (408) 739-9803  
www.national.com

CH1786ET  
Cermetek  
(408) 752-5000  
(800) 882-6271  
Fax: (408) 752-5004  
www.cermetek.com

Serialtest Async  
Frontline Test Systems  
(804) 984-4500  
Fax: (804) 984-4505  
www.fte.com

www.circuitcellar.com/online  CIRCUIT CELLAR® ONLINE  February 2001  5
Listing 1—

/**************************************************************************
 * ASCII TO INTEGER CONVERSION ROUTINE
 */
**************************************************************************/

#include <6F877.H>
#include <877.H>
#define MS,NOWDT,NOPROTECT,NOBROWNOUT, NOLVP, PUT
#define 0x1231

/**************************************************************************
 * DEFINE END OF USER CODE SPACE
 */
**************************************************************************/

#define MAXADDR 0x1000

/**************************************************************************
 * DEFINE INTEL HEX BUFFER AREA OF 64 BYTES IN BANK 2
 */
**************************************************************************/

#define BUFFER_LEN 64
int buffaddr;
#define buffaddr = 0xA0
char buffer[BUFFER_LEN];
#define buffer = 0xA1

/**************************************************************************
 * DEFINE FLOW CONTROL PINS
 */
**************************************************************************/

#define CTSON output_low(PIN_B2)
#define CTSOFF output_high(PIN_B2)
#define RTS_input(PIN_B1)

/**************************************************************************
 * TELL COMPILER NOT TO PLACE THIS CODE INLINE
 */
**************************************************************************/

#define SEPARATE

unsigned int atoi_bl16(char *s);

/**************************************************************************
 * INLINE RECEIVE CHARACTER ROUTINE
 */
**************************************************************************/

#define inline

char rxchar(){
char serial_in;
asm
norx:
btfs RCIF
goto norx
movf RCREG,W
movwf serial_in
#endasm
return(serial_in);
}

/**************************************************************************
 * INLINE TRANSMIT CHARACTER ROUTINE
 */
**************************************************************************/

#define inline

void txchar(char s){
char serial_out;
serial_out = s;
asm
notx:
btfs TXIF
goto notx
movf serial_out,W
movwf TXREG
#endasm
}

Listing 1—continued

/* ASCII TO INTEGER CONVERSION ROUTINE */
**************************************************************************/

unsigned int atoi_bl16(char *s){
    unsigned int result = 0;
    int i;

    for (i=0; i<2; i++) {
        if (*s >= 'A' && *s <= 'F')
            result = 16*result + (*s) - 'A' + 10;
        else if (*s >= '0' && *s <= '9')
            result = 16*result + (*s) - '0';
        s++;
    }

    return(result);
}

/**************************************************************************
 * MAIN BOOT LOADER ROUTINE
 */
**************************************************************************/

void main(void) {
    char byte_in;
    short code_good, not_done, no_char, bootcodeflag;
    int i, count, line_type, cs, check_sum, addrl;
    long bootaddr, data_addr, addrh, bootcodeaddr, timel, timrh;

    // GET BOOTCODE VECTOR ADDRESS
    /**************************************************************************/
    #asm
    movphw bootcode
    movwf addrh
    movplw bootcode
    movwf addrl
    #endasm
    bootcodeaddr = addrh << 8 | addrl;
    /**************************************************************************/

    /**************************************************************************
    * INITIALIZE THE PIC UART 2400/N/8/1
    */
    /**************************************************************************/
    #asm
    TRISB = 0xFB;
    CTSOFF;
    SPBRG = 0x2F;
    TXSTA = 0x22;
    SPEN = 1;
    CREN=1;
    #endasm
    /**************************************************************************/

    /**************************************************************************
    * WAIT FOR UPLOAD COMMAND FROM TERA TERM
    */
    /**************************************************************************/
    #asm
    timr1=0x20; // DELAY VALUE
    timrh=0xFFFF; // DELAY VALUE
    no_char = TRUE;
    CTS0N;
    do(
        if(bit_test(PIR1.S)){
            byte_in = rxchar();
            no_char = FALSE;
        }
        -timrh;
        if(timrh == 0x00){
            (continued)
Listing 1—(continued)

```c
    txchar("**");
    -timr;
    timr = 0xFFFF;
  }
  }
} while ((timr != 0x000) & (no_char == TRUE));
if(timr != 0x000 & no_char == FALSE & byte_in ==
  'u'") {
  CTSOFF;
  goto download;
}

 /*******************************************************************************/
/* BOOTCODE VECTOR */
*******************************************************************************/

#define clrf PCLATH

  #asm
  
  bootloader:
  
  nop
  nop
  nop
  nop
  goto 0x000
  #endasm

 /*******************************************************************************/
/* DOWNLOAD INTEL HEX FILE FROM INTELLI TERM PRO */
 /*******************************************************************************/

  download:
    /not_done = TRUE;
    CTS0N;
    while (1){
      bufaddr = 0;
      code_good = TRUE;
      do {
        buffer[bufaddr] = txchar();
      } while ((buffer[bufaddr-1] != 0x0D) &
      (bufaddr == BUFFER_LEN));
      CTSOFF;
    
 /*******************************************************************************/
/* INTEL HEX PROCESSOR */
 /*******************************************************************************/

    /: = START OF HEX LINE */
    /: AA = NUMBER OF BYTES IN THE ASCII LINE */
    /: BB = LINE ADDRESS */
    /: CC = LINE TYPE */
 /*******************************************************************************/

 /*******************************************************************************/
/* PROCESS THE INTEL HEX LINE HEADER */
 /*******************************************************************************/

    if ((buffer[0] == ':') {
      //:AA
      count = atoi_b16 (&buffer[1]);
      //:AABB
      addr = atoi_b16 (&buffer[3]);
      //:AABBCC
      addr = (addr << 8) | atoi_b16 (&buffer[5]);
      //AABBCCDD
      line_type = atoi_b16 (&buffer[7]);
      /******************************************************************************/
/* ADJUST THE ADDRESS */
 /******************************************************************************/

    if (addr != 0)
      addr /= 2;
    /*******************************************************************************/
/* END OF HEX FILE DATA IS LINE TYPE = 0X01 */
 /******************************************************************************/

Listing 1—(continued)

 /*******************************************************************************/
if (line_type == 1)
  break;
 /*******************************************************************************/
/* COMPUTE THE HEX LINE CHECKSUM */
 /*******************************************************************************/

    else {
      cs = 0;
      for (i=1; i<buffaddr-2; i++)
        cs += atoi_b16 (&buffer[i]);
      cs = 0xFF - cs + 1;
      check_sum = atoi_b16 (&buffer[buffaddr-3]);
      if (check_sum != cs)
        code_good = FALSE;
    
 /*******************************************************************************/
/* PROCESS THE INTEL HEX DATA AREA */
 /*******************************************************************************/

    else {
      i = 9;
      while (i < buffaddr-3) {
        data = atoi_b16 (&buffer[i+2]);
        data = (data << 8) | atoi_b16 (&buffer[i]);
    /*******************************************************************************/
/* WRITE THE INTEL HEX DATA TO PROGRAM MEMORY */
 /*******************************************************************************/

    bootcodeflag = FALSE;
    if (addr < 0x0004)
      bootcodeflag = TRUE;
    bootaddr = addr + bootcodeaddr;
    write_program_eeprom (bootaddr, data);
    }
    
 /*******************************************************************************/
/* IF INTEL HEX DOWNLOAD IS SUCCESSFUL */
 /*******************************************************************************/

    CREN = 0;
    timr = 0xFFFF;
    timr = 0x40;
    CTS0N;
    do{
      -timr;
    if (timr == 0x00)
       txchar("**");
     -timr;
       timr = 0xFFFF;
    }
  } while (timr != 0x000);
  if (code_good)
    txchar("g");
(continued)
Listing 1—continued

/* IF INTEL HEX DOWNLOAD IS IN ERROR */
* PUT NOPs AT BOOTCODE VECTOR AREA *
* SEND e TO TERA TERM AND RESTART *
*******************************************************************************/

else{
  addr = 0x00;
  data = 0x00;
  while (addr < 4){
    bootaddr = addr + bootcodeaddr;
    write_program_eprom(bootaddr, data);
    addr = ++addr;
  }
  txchar('E');
}
*******************************************************************************/

/* ABSOLUTE RESET VECTOR TO LOCATION 0X0000 */
*******************************************************************************/

#define asm
MOV LW 0x00
MOV W 0x0A
GOTO 0x00
#endif
Listing 2—continued

if result = 1 goto donegood
if result = 2 goto uploadererror
:donegood
messagebox upload good boxtitle end
:noload
disable
:uploadererror
messagebox upload bad boxtitle end

Listing 3

<TITLE>EDTPS-7600A/PIC16F877 Internet Engine</TITLE>

Listing 4

#include "16f877.h"
#include "f877.h"
#include "s7600.h"
#include <ctype.h>
#include <string.h>

#define FUSES_HS, FUSE_G, FUSE_N, FUSE_B, FUSE_X
#define EEPROM_SDA PIN_C4
#define EEPROM_SCL PIN_C3
#define h(x) (*x*x+1)
#define rs232(baud=2400, xmit=PIN_C6, rcv=PIN_C7)
#define standard_io(A)
#define standard_io(B)
#define standard_io(C)
#define standard_io(D)
#define i2c(master, sda=EEPROM_SDA, scl=EEPROM_SCL)
#define EEPROM_ADDRESS 0x0B
#define EEPROM_SIZE 0x124
#define esc 0x1B

// PORTA bits
#define PORTA 0x0B
#define PORTA 0x0A
#define PORTA 0x09
#define PORTA 0x08

// PORTB bits
#define PORTB 0x0B
#define PORTB 0x0A
#define PORTB 0x09
#define PORTB 0x08

(continued)
Listing 4—continued

   #include "seiko_ct.c"  // Seiko routines use code from AN732

   /***************************************************************************/
   ** char DataAvailable(void) **
   ** Determines if there is any data available to read out of **
   ** the S-7600A. **
   ** S-7600A. **
   ** returning the value of the data available bit from the **
   /***************************************************************************/
   char DataAvailable(void)
   {
      return (ReadSeiko(Serial_Port_Config)&0x80);
   }

   /***************************************************************************/
   ** void S_Putc(char data) **
   ** Writes a byte of data to the serial port on the S-7600A. **
   /***************************************************************************/
   void S_Putc(char data)
   {
      while(!BUSY);
      CS = 1;
      RS = 0;
      WRITE = 0;
      PORTD=Serial_Port_Data;
      TRISD = 0;
      REAX = 1;
      REAX = 0;
      WRITE = 1;
      RS = 1;
      CS = 0;
      CS = 1;
      WRITE = 0;
      PORTD=data;
      REAX = 1;
      REAX = 0;
      WRITE = 1;
      CS = 0;
      TRISD = Oxff;
   }

   /***************************************************************************/
   ** void W_Putc(char data) **
   ** Writes a byte of data to the socket on the S-7600A. **
   /***************************************************************************/
   void W_Putc(char data)
   {
      // Make sure that the socket buffer is not full
      while(0x20==ReadSeiko(0x222&0x20))
      {
         WriteSeiko(TCP_Data_Send,0);
         while(ReadSeiko(Socket_Status_H));
      }
   }

   /***************************************************************************/
   ** void Get_password(void) **
   ** Requests and reads the password from the input terminal. **
   /***************************************************************************/
   void Get_password()
   {
      byte a_temp;
      i=0;
      printf("Sc[2J",esc);
Listing 4—(continued)

```c
printf("%c[12;20H 32 chars max", esc);
while(1)
{
    pass[i]=0;
    ch=getch();
    if(ch==0x0D)
        break;
    if(ch != 0x0A)
    {
        putc(ch);
        if(ch == 0x08)
        {
            pass[i]=ch;
            i++;
        }
    }
    if(i==16) break;
}
// write password to the EEPROM
a_tmp=0x20;
for(i=0;i<0x1F;i++)
{
    ch=pass[i];
    write_ext_eeprom(a_tmp,pass[i]);
    a_tmp++;
}
}

// ******************************************
** void Get_phone(void)
**
** Requests and reads the telephone number from the input
** terminal.
**
// ******************************************
void Get_phone()
{
    byte p_tmp;
    printf("%c[2J", esc);
    printf("%c[12;20H 16 chars max", esc);
    printf("%c[10;20H Enter phone number: ", esc);
    i=0;
    while(1)
    {
        phone[i]=0;
        ch=getch();
        if(ch==0x0D)
            break;
        if(ch != 0x0A)
        {
            putc(ch);
            if(ch == 0x08)
            {
                phone[i]=ch;
                i++;
            }
        }
        if(i==16) break;
    }
    // write phone number to the EEPROM
    p_tmp=0x40;
    for(i=0;i<16;i++)
    {
        ch=phone[i];
        write_ext_eeprom(p_tmp,phone[i]);
        p_tmp++;
    }
    // ******************************************
    ** void Get_time(void)
    **

    (Continued)
```

Listing 4—(continued)

```c
** Resets time and date
**
**
**
// ******************************************
void Get_Time(void)
{
    printf("%c[2J", esc);
    printf("%c[10;20H Enter minutes 00-59: ", esc);
    i=0;
    while(1)
    {
        mins[i]=0;
        ch=getch();
        if(ch < 0x3A && ch > 0x2F)
        {
            mins[i]=ch;
            i++;
        }
        else
            goto tryminagain;
        if(i==2)
            break;
    }
    if(mins[0] > 0x35)
        goto tryminagain;
    swap(mins[0]);
    mins[0] &= 0xF0;
    mins[1] &= 0xF0;
    mins[2] = mins[0] | mins[1];
    tryhoursagain:
    printf("%c[11;20H Enter hours 00-23: ", esc);
    i=0;
    while(1)
    {
        hours[i]=0;
        ch=getch();
        if(ch < 0x3A && ch > 0x2F)
        {
            hours[i]=ch;
            i++;
        }
        else
            goto tryhoursagain;
        if(i==2)
            break;
    }
    if(hours[0] == 0x32 && hours[1] > 0x33)
        goto tryhoursagain;
    if(hours[0] > 0x32)
        goto tryhoursagain;
    swap(hours[0]);
    hours[0] &= 0xF0;
    hours[1] &= 0xF0;
    hours[2] = hours[0] | hours[1];
    trydayagain:
    printf("%c[12;20H Enter day 1-7 (Sunday = 1): ", esc);
    i=0;
    while(1)
    {
        day[i]=0;
        ch=getch();
        if(ch < 0x3A && ch > 0x2F)
            day[i]=ch;
        else
```

(Continued)
Listing 4—continued

goto trydayagain;
break;
}
if(day[0] > 0x37)
goto trydayagain;
day[0] = day[0] & 0x0F;

trydateagain:
printf("%c13;20H Enter Date 01-31: ", esc);

i=0;
while(1)
{
  date[i]=0;
  ch=getc();

  putc(ch);
  if(ch < 0x3A & ch > 0x2F)
  {
    date[i]=ch;
    i++;
  }
else
  goto trydateagain;

  if(i==2)
break;
}
if(day[0] > 0x33)

trydateagain:
if(date[0] == 0x00 & date[1] == 0x00)
goto trydateagain;
if(date[0] == 0x33 & date[1] > 0x31)
goto trydateagain;

swap(date[0]);
date[0] & 0x0F;
date[1] & 0x0F;
date[2]=date[0] | date[1];

trymonthagain:
printf("%c14;20H Enter Month 01-12: ", esc);

i=0;
while(1)
{
  month[i]=0;
  ch=getc();

  putc(ch);
  if(ch < 0x3A & ch > 0x2F)
  {
    month[i]=ch;
    i++;
  }
else
  goto trymonthagain;

  if(i==2)
break;
}
if(month[0] > 0x31)
goto trymonthagain;
if(month[0] == 0x00 & month[1] == 0x00)
goto trymonthagain;
if(month[0] == 0x31 & month[1] > 0x32)
goto trymonthagain;

swap(month[0]);
month[0] & 0xF0;
month[1] & 0x0F;
month[2]=month[0] | month[1];

tryyearagain:
printf("%c15;20H Enter Year 00-99: ", esc);

i=0;
while(1)
{

(continued)
Listing 4—continued

```c
} write_ext_eeprom(index, 0); // Write terminating NULL

CTS = 0;

// Print status of download to EEPROM
index = index - 80;
printf("%c\20", esc);
printf("%c%2H Received %1ub bytes", esc, index);
if (index < 32688)
printf("%c\16\20H Error maximum bytes is 32688", esc);
printf("%c\18\25H Press any key to continue", esc);
ch = getc();
CTS = 1;
}

/ ******************************************
** void init_temp
**
** initialize the DS1629
**
******************************************/
void init_temp(){
12c_start();
12c_write(0x9E);
12c_write(0x0A);
12c_write(0x00);
12c_stop();
}

/ ******************************************
** void Menu(void)
**
** Displays menu on user’s terminal screen. Allows changes
**
** to username, password, phone number and web page.
**
******************************************/
void Menu(void)
{
        i = 0;
        while(ch != 0x1B)
        {
                CTS = 0;
                printf("%c\20", esc);
                printf("%c\31;25H 1 Enter user name", esc);
                printf("%c\31;25H 2 Enter password", esc);
                printf("%c\31;25H 3 Enter phone number", esc);
                printf("%c\31;25H 4 Set Time and Date", esc);
                printf("%c\31;25H 5 Down load HTML file", esc);
                printf("%c\31;30H ESC exit", esc);
                ch = getc(); // Get input and process
                switch(ch)
                {
                        case 0x31: // ‘1’ -> change username
                                Get_username();
                                break;
                        case 0x32: // ‘2’ -> change password
                                Get_password();
                                break;
                        case 0x33: // ‘3’ -> change phone #
                                Get_phone();
                                break;
                        case 0x34: // ‘4’ -> change time and date
                                Get_Time();
                                break;
                        case 0x35: // ‘5’ -> new web page
                                Read_file();
                                break;
                        default:
                                printf("Error command!");
                        }
        }
}
```

Listing 4—continued

```c
)}

CTS = 0;

/ ******************************************
** void main(void)
**
** Displays menu on user’s terminal screen. Allows changes
**
** to username, password, phone number and web page.
**
******************************************/
void main()
{
        // Initialize PORTs & TRISs
        // BIT 1 REAX
        // / 2 CS
        // / 3 RS
        PORTA = 0xF9; //1111001
        // BIT 2 CTS
        // / 3 BUSY
        // / 4 INT1
        // / 5 WRITE
        PORTB = 0x38; //0011000
        // BIT 0 RESET
        PORTC = 0x80; //1000000
        PORTD = 0x00; //00000000
        // BIT 0 LED1
        // / 1 LED2
        PORTE = 0xFA; //1111010
        TRISA = 0x01; //11010001
        TRISB = 0x18; //00011000
        TRISC = 0x80; //10000000
        TRISD = 0xFF; //11111111
        TRISE = 0x00; //00000000
        ADCON1 = 0x06; //00000010 all digital
        ADCON0 = 0;
        TICON = 0x31; //00110001 Timer1
        CTS = 1;
        init_ext_eeprom();
        init_temp();
        Menu();
        restart:
        }
WriteSeiko(Serial_Port_Config, 0x06);
        delay_ms(10);
        WriteSeiko(Clock_Div_L, 0x5E);
        while(ReadSeiko(Clock_Div_L) != 0x5E)
WriteSeiko(Clock_Div_H, 0x00);
WriteSeiko(BAUD_Rate_Div_L, 0x5F);
WriteSeiko(BAUD_Rate_Div_H, 0x00);
WriteSeiko(Serial_Port_Config, 0x00);
        printf(S_Putc,"AT&TFR");
        delay_ms(10);
        WriteSeiko(PPP_Control_Status, 0x01);
        WriteSeiko(PPP_Control_Status, 0x00);
        WriteSeiko(PPP_Control_Status, 0x20);
        delay_ms(5);
```

(continued)
ch=1;
i=0;
while(ch)
{
    ch = read_ext_eeprom(i);
    i++;
}
i=0;
WriteSeiko(PAP_String,i);
for(j=0;j<i;j++)
{
    ch = read_ext_eeprom(j);
    WriteSeiko(PAP_String,ch);
}
ch=1;
i=0x20;
while(ch)
{
    ch = read_ext_eeprom(i);
    i++;
}
i=0x20;
WriteSeiko(PAP_String,i);
for(j=0x20;j<(1+0x20);j++)
{
    ch = read_ext_eeprom(j);
    WriteSeiko(PAP_String,ch);
}
WriteSeiko(PAP_String,0x00);
printf(S_Putc,"ATDT");
ch=1;
index=0x40;
while(1)
{
    ch = read_ext_eeprom(index);
    if(ch == 0)
        break;
    S_Putc(ch);
    index++;
}
printf(S_Putc,"\r");
delay_ms(5);
printf("%c[2j",esc);
printf("\rDialing");
ch=1;
i=0x40;
while(1)
{
    ch = read_ext_eeprom(i);
    if(ch == 0)
        break;
    printf("%c",ch);
    i++;
}
printf("\r");
delay_ms(20000);
WriteSeiko(PPP_Control_Status,0x62);
WriteSeiko(Serial_Port_Config,0x04);
delay_ms(5);

while((!(ReadSeiko(PPP_Control_Status)&0x01))
delay_ms(5);

(Continued) —

while(ReadSeiko(Our_IP_Address_L) == 0);
MyIPAddr[0] = ReadSeiko(Our_IP_Address_L);
MyIPAddr[1] = ReadSeiko(Our_IP_Address_M);
MyIPAddr[2] = ReadSeiko(Our_IP_Address_H);
MyIPAddr[3] = ReadSeiko(Our_IP_Address_U);

printf("\nMy address is \
 192.168.1.1",MyIPAddr[3],MyIPAddr[2],MyIPAddr[1],MyIPAddr[0]);

while(1)
{
    while(1)
    {
        delay_ms(1);
        if(!((ReadSeiko(PPP_Control_Status)&0x01))
goto restart;
        if((ReadSeiko(Serial_Port_Config)&0x40))
goto restart;
        WriteSeiko(Socket_Index,0x00);
        WriteSeiko(Socket_Config_Status_L,0x10);
delay_ms(10);
        WriteSeiko(Our_Port_L,80);
        WriteSeiko(Our_Port_H,0);
        WriteSeiko(Socket_Config_Status_L,0x06);
        WriteSeiko(Socket_Activate,0x01);
delay_ms(5);
        printf("\nSocket open\n\r");
        i = 2;
        while(1)
        {
            delay_ms(1);
            if(!((ReadSeiko(PPP_Control_Status)&0x01))
goto restart;
            if((ReadSeiko(Serial_Port_Config)&0x40))
goto restart;
            temp =
            ReadSeiko(Socket_Status_M);
            if(temp&0x10)
            {
                i = 0;
                break;
            }
            else if(temp&0x00)
            break;
            if(temp == 0x09)
            continue;
delay_ms(5);
            i++;
(continued)
Listing 4—continued

if(i == 255) break;
} if(!i) break;

if(i == 1) break;
printf("\n\nWaiting for data\n");
WriteSeiko(Socket_Interrupt_H,0xF0);
i=0;
printf("\n\nReading data\n\n");
while(ReadSeiko(Socket_Config_Status_L)&0x10) {
  temp = ReadSeiko(Socket_Data);
putc(temp);

  WriteSeiko(Socket_Data,0x0A);
  WriteSeiko(Socket_Data,0x0D);
  WriteSeiko(Socket_Data,0x0A);
  WriteSeiko(Socket_Data,0x0D);

  byt_cnt=0;
  index=0x30;
  ch="
  while(ch != 0) {
    ch =
    read_ext_eeprom(index);
    if(ch == 0) break;
    if(ch == 0x25) {
      index++;
      ch =
    }
    read_ext_eeprom(index);
    switch(ch) {
      case 'a':
        printf(W_putchar, "%u.%u.%u",MyIPAddr[3],MyIPAddr[2],MyIPAddr[1],MyIPAddr[0]);
        break;
      case 'c':
        i2c_start();
        i2c_write(0x0E);
        i2c_write(0x0E);
        delay_ms(500);
        i2c_start();
        i2c_write(0x9E);
        i2c_write(0x9E);
        i2c_start();
        i2c_write(0xAA);
        i2c_start();
        i2c_write(0x9F);
        ctmp = i2c_read();
        i2c_stop();
        negtemp = FALSE;
        if(ctmp >= 0x80) {
          ctmp = ctmp +1;
          negtemp = TRUE;
        }
        if(negtemp==TRUE) printf(W_putchar,-"-");
        printf(W_putchar, "%u",ctmp);
        break;
      case 't':
        i2c_start();
        i2c_write(0x9E);
        (continued)

Listing 4—continued

i2c_write(0x00);
i2c_write(0x01);
i2c_start();
i2c_write(0x0F);
//i = i2c_read();
mins[0] = i2c_read();
hours[0] = i2c_read();
day[0] = i2c_read();
date[0] = i2c_read();
month[0] = i2c_read();
year[0] = i2c_read();
i2c_stop();

negtemp = FALSE;
if(ctmp >= 0x80) {
  ctmp = ctmp +1;
  negtemp = TRUE;
}
//secs[2] = secs[0];
//secs[0] &= 0xF0;
//secs[0] >>= 4;
//secs[0] &= 0xF0;
//secs[0] += 0x30;
//secs[1] = secs[2];
//secs[1] &= 0xF0;
//secs[1] += 0x30;

mins[2] = mins[0];
mins[0] &= 0xF0;
mins[0] >>= 4;
mins[0] &= 0xF0;
mins[0] += 0x30;
mins[1] = mins[2];
mins[1] &= 0xF0;
mins[1] += 0x30;

hours[2] = hours[0];
hours[0] &= 0xF0;
hours[0] >>= 4;
hours[0] &= 0xF0;
hours[0] += 0x30;
hours[1] = hours[2];
hours[1] &= 0xF0;
hours[1] += 0x30;

//day[0] &= 0xF0;
//day[0] += 0x30;

date[2] = date[0];
date[0] &= 0xF0;
date[0] >>= 4;
date[0] &= 0xF0;
date[0] += 0x30;
date[1] = date[2];
date[1] &= 0xF0;
date[1] += 0x30;

month[2] = month[0];
month[0] &= 0xF0;
month[0] >>= 4;
month[0] &= 0xF0;
month[0] += 0x30;
month[1] = month[2];
month[1] &= 0xF0;
month[1] += 0x30;

year[2] = year[0];
year[0] &= 0xF0;
year[0] >>= 4;
year[0] &= 0xF0;
year[0] += 0x30;
year[1] = year[2];
year[1] &= 0xF0;
year[1] += 0x30;

printf(W_putchar, "%c%c%c%c%c%c-20%c%c",hours[0],hours[1],(continued)
Listing 4—continued

```c
mins[0],mins[1],month[0],month[1],date[0],date[1],year[0],year[1]);
    break;
  }
}
else
{
    count=0;
    count1=0;

    bit_clear(P1P1,TMR1IF);
    while(0x0200=(ReadSeiko(0x22)&0x20))
    {

    WriteSeiko(TCP_Data_Send,0);
    while(ReadSeiko(Socket_Status_H));
    
    if(ReadSeiko(Serial_Port_Config)&0x40)
        goto restart;
        if(bit_test(P1P1,TMR1IF))
    {
        count++;
        if(count > 0xf8)
        {
            count1++;
            count=0;
        }
        if(count1 > 0x00)
            goto restart;
            bit_clear(P1P1,TMR1IF);
    }
}

WriteSeiko(Socket_Data,ch);
} index++;
}

WriteSeiko(TCP_Data_Send,0);
    count=0;
    bit_clear(P1P1,TMR1IF);
    while(0x0a00=(ReadSeiko(0x22)&0x40))
    {
        if(bit_test(P1P1,TMR1IF))
        {
            count++;
            bit_clear(P1P1,TMR1IF);
        }
        
        printf("Close socket\n");
        WriteSeiko(Socket_Activate,0);
        WriteSeiko(TCP_Data_Send,0);
        for(i=0;i<255;i++)
        {
            delay_ms(10);
            temp = ReadSeiko(Socket_Status_H);  
            if((temp & 0x0f)==0x08)
            break;
            if((temp&0xe0))
            break;
            printf("\n\r final socket wait\n");
        }
    while(ReadSeiko(Socket_Status_H));
    delay_ms(5000);
    }
```

(continued)