AN731

Embedding PICmicro® Microcontrollers in the Internet

Author: Rodger Richey/Steve Humberd
Microchip Technology Inc.
Chandler, AZ

INTRODUCTION

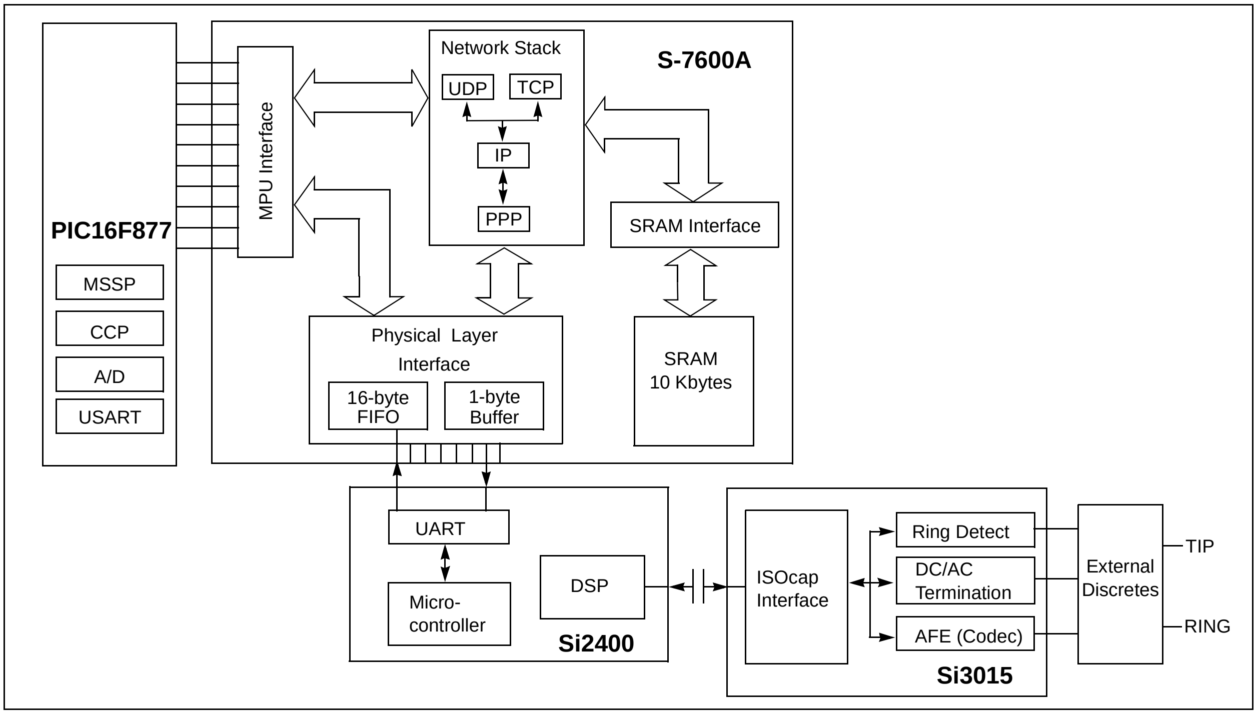
The market for products or services that are Internet related is HOT! Increased amounts of money and design resources are being thrown at these products and services. One significant portion of this trend is to embed the Internet or, in other words, make embedded products have the capability to connect to the Internet. It is estimated that by the year 2005, the number of embedded applications with the ability to connect to the Internet will be larger than the number of PCs by a factor of 100 or more.

The latest in PDAs and cellular telephones allow the user to access stock quotes, sports scores, E-mail, and more. The other advantage of the movement to embed the Internet is enabling client devices, such as appliances and vending machines, to connect to the Internet and upload status information. The price for such products is being reduced dramatically due to the introduction of new technology.

The application note presented here is based on the block diagram shown in Figure 1. It consists of the PIC16F87X microcontroller, the Seiko iChip™ S-7600A TCP/IP stack IC, and the ISOmodem™ Si2400 and Si3015 DAA from Silicon Labs. Each of these components was specifically selected for this application because of the feature set it provides. Each of the following sections will present the reasons for selection. The sample applications include both a web server and a client. The web server stores the HTML page in an external serial EEPROM and dynamically inserts temperature and potentiometer settings into the HTML file as it is being sent. The client application mimics a vending machine that uploads status information at predefined intervals.

This application note was written with the assumption that the reader has a basic level of knowledge of Internet protocols. There are many good books on the market that describe the Internet and all the various protocols used. It is suggested that the reader obtain one of these books to understand what the protocols are used for and how they relate to other protocols. Appendix A is a dictionary of the terms used in this application note.

FIGURE 1: BLOCK DIAGRAM



EMBEDDED WEB SERVER VS. CLIENT

When most people think of the Internet, they think about viewing web pages filled with information. A computer of some sort provides the necessary resources required to support a web server. The computer has significant hard disk space to hold the web pages and ancillary data files such as data sheets, application notes, etc. It also has a high-speed communications interface such as a T1 line, a 56K modem, etc., that can serve this information to a user quickly.

Embedded systems are quite different from standard web servers. They have limited resources in terms of memory space and operating speed. Embedded systems usually work with a few kilobytes, up to several megabytes of memory and operate up to 40 MHz. Typical PCs have several gigabytes of memory storage and are starting to operate in the gigahertz range. Therefore, web servers are not really practical for embedded applications, but the embedded web client is very practical.

One of the most viable embedded web client applications is a vending machine. The microcontroller keeps track of all functions:

- Price for each item
- Number of items left
- Amount of deposited money
- Amount of money available for change

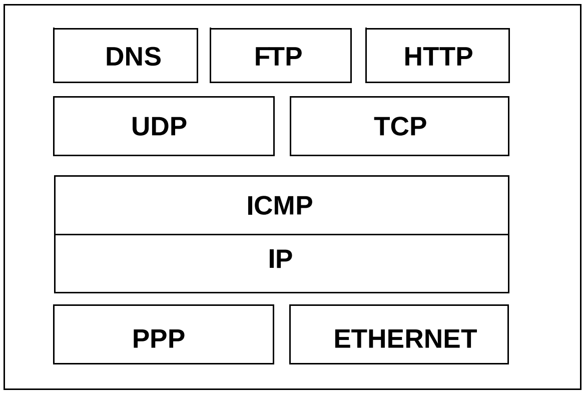
One could also argue that the machine could keep track of the name or type of items that it is dispensing.

On a preset interval (such as once a night between 11 PM and 4 AM), the vending machine would dial out and make a connection to a local Internet Service Provider (ISP). The machine would then connect to a server and upload all the information. If the machine is out of change, full of deposited money, or needs restocking, a report can be generated detailing exactly what is required. The service person can now make one trip to the machine because they know exactly what that particular machine needs. The machine can also call out if an operating error has been detected. This may include the vending machine is tilted to improperly dispense items, or lights have burned out. A service report is generated and the service person knows exactly what is needed to fix the problems. Since we have established a full duplex link, the embedded Internet connection can provide additional benefits, such as reprogramming the microcontroller with new firmware to fix bugs, or new pricing structure for items.

MICROCONTROLLER

The microcontroller is the primary component of the application, not necessarily a part of the Internet communications. In some cases, it may be advantageous to offload the Internet communications to an external device, so the microcontroller can focus on the details of the application, as well as the application layer of the Internet Protocol Stack (see Figure 2).

FIGURE 2: INTERNET PROTOCOL STACK



One feature that provides the most flexibility for an embedded application is FLASH-based program memory. FLASH program memory provides the capability to reprogram portions of the firmware remotely over the Internet connection. The sections of firmware that can be reprogrammed not only include the instructions, but also sensor calibration tables, IDs, and application lookup tables. For instance, the pricing for the vending machine items may be stored in a lookup table that can be reprogrammed by the server when a connection is made.

When choosing a FLASH-based microcontroller, it is important to select one with the features that can facilitate this type of remote reprogramming operations. The microcontroller must be able to modify its own firmware without the need for external circuitry. It must also have the capability to reprogram without interrupting the operation of the application. In other words, the peripherals and internal clocks must continue to operate and queue events, while the microcontroller is reprogramming memory locations.

One example of this type of microcontroller is the PIC16F87X family of products from Microchip Technology. Table 1 shows the current family of products. These products provide a migration path for memory size and feature set within a given footprint (28 or 40 pins).

TABLE 1: PIC16F87X PRODUCT FAMILY

Features	PIC16F870	PIC16F871	PIC16F872	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz						
Resets	Power-on Reset, Brown-out Reset, Power-up Timer, Oscillator Start-up Timer						
FLASH Program Memory	2K	2K	2K	4K	4K	8K	8K
Data Memory	128	128	128	192	192	368	368
EEPROM Data Memory	64	64	64	128	128	256	256
Interrupts	10	10	11	13	14	13	14
I/O Pins	22	33	22	22	33	22	33
Timers	3	3	3	3	3	3	3
Capture/Compare/PWM	1	1	1	2	2	2	2
Serial Communication	USART	USART	MSSP	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communication		PSP			PSP		PSP
10-bit A/D Channels	5	8	5	5	8	5	8
Instruction Set	35 Instructions						
Packages	28-pin DIP, SOIC, SSOP	40-pin DIP 44-pin PLCC, TQFP	28-pin DIP, SOIC, SSOP	28-pin DIP, SOIC	40-pin DIP 44-pin PLCC, TQFP	28-pin DIP SOIC	40-pin DIP 44-pin PLCC, TQFP

These devices have the standard complement of peripherals including USARTs, Master Mode I^{2}C^{TM}, SPI^{TM}, A/D, Capture/Compare/PWM and Timers. It also has up to 8K words of FLASH program memory, up to 368 bytes of RAM and 256 bytes of data EEPROM memory. By combining both a data EEPROM and FLASH program memory, the designer has the flexibility to store data in the appropriate place. Information that is updated frequently can be stored in the data EEPROM, which has increased endurance over the FLASH program memory. This data can be modified without interrupting the operation of the application. Other information that is rarely changed can be stored in FLASH program memory. When updating these locations, the microcontroller ceases to execute instructions, but continues to clock the peripherals and queue interrupts.

The programming operation is self-timed, i.e., an internal clock source applies the high voltage to the memory cells to erase and/or write the data for a predetermined amount of time. This frees the designer from the task of timing the event. It also reduces the risk of the high voltage being applied too long, which damages or reduces the endurance of the cell, or not long enough, which does not fully program the memory location. The actual programming code is shown in Example 1. This simple piece of code assumes the addresses of the memory location and data have been loaded before being called. The only difference between modifying FLASH program memory and data EEPROM memory is the setting of a single bit. This bit needs to be configured only once and all subsequent programming operations will be directed to the desired memory.

EXAMPLE 1: SELF-TIMING CODE

```
bsf     STATUS,RP1     ;Switch to Bank 3
bsf     STATUS,RP0
bsf     EECON1,WREN    ;Enable write
movlw   55h            ;5 instruction
mowf    EECON2         ;sequence to
movlw   AAh            ;initiate the
movwf   EECON2         ;write operation
bsf     EECON1,WR      ;to memory
nop
nop
```

The programming sequence contains a five-instruction sequence that must be executed exactly to initiate the programming cycle. If this sequence is not followed, the programming operation will not start. Also included is a write enable bit. If this bit is not set, then writes will not take place. These two features provide a measure of protection against inadvertent writes.

The other part of the remote programming operation handled by the microcontroller is the bootloader. This portion of the firmware accepts data from the outside world and controls the erase/program operations. It may also include security features for entering the programming mode. One feature that is a must in all bootloaders is the capability to recover when the communications medium is interrupted. If the Internet connection, or telephone connection is interrupted, the microcontroller must be smart enough to recover. The application note AN732 provides one method for recovery. The security portion is not implemented because requirements are heavily dependent on the application.

TCP/IP STACK

The Internet protocol used varies depending on the application. Figure 2 shows the standard Internet protocol stack. In most embedded systems, a dial up connection will be used and, therefore, the Network Access Layer will be Point-to-Point Protocol (PPP). This protocol encapsulates the other protocols for transmission over serial links such as modems. This is the most commonly used protocol for dial-up networks such as America Online Inc. or Compuserve®. PPP performs three main functions. It encapsulates IP and TCP/UDP packets with a header and checksum and inserts escape characters to distinguish data bytes from flag bytes. It also configures the link between the two peers, including compression and maximum receive units (MRUs). Finally it configures the network connection allowing multiple network protocols and negotiation of an IP address. PPP does not specify a server or a client. It assumes there are two peers, both of which have equivalent authority in the connection. PPP also handles user name and password authentication. Two protocols are supported: Password Authentication Protocol (PAP) and Challenge Handshake Authentication Protocol (CHAP). Once the link parameters are negotiated, PPP is used to transfer data packets between the two hosts and terminate the connection, once all the data has been sent.

The transport layer provides the basic communication between a server and a client. User Datagram Protocol (UDP) is not based on making connections between the hosts. In other words, the host receiving a packet from another host does not acknowledge the reception. Therefore, the transmitting host has no indication of whether the packet made it to its destination. The acknowledgment can be a return UDP packet from the receiving host. However, this is not included in the UDP specification and requires additional overhead for both hosts. UDP is the best protocol for use when transmitting small amounts of data or for regular transmissions of small data packets. Each UDP packet is encapsulated with a header and a checksum to provide some error checking.

Unlike UDP, Transmission Control Protocol (TCP) provides a flow controlled, connection based host-to-host transfer of data. TCP provides connections based on IP address and ports. Each of the source and destination devices have different IP addresses, but both devices must be using the same port to be able to communicate to each other. The number of the port used usually determines what type of application layer is used for communication. These port numbers may be from 0 to 65536, but some are already assigned to specific applications. A port number of 80 is usually assigned to HTTP. Therefore an HTTP server will listen to port 80 for any incoming data. The HTTP server will accept data packets, if the destination IP address and the port number match. A large data file will be broken up into many packets of data. TCP is capable of receiving these packets in any order and then reconstructing the original file. TCP also has the same type of checksums that are involved with UDP.

TCP has been deemed to be the more reliable method of transferring data over the Internet, but it really depends on the type of data being transferred between the two devices. One factor is the relative importance of the data. If a device is transmitting temperature measurements every couple of minutes, losing one reading may not be as severe as losing one frame of data in an image file. Temperature changes very slowly and the fact that another reading will be sent in a couple of minutes would allow extraction of the missing reading from the previously received reading and the current reading. UDP can come close to achieving some of the features of TCP, but it involves a lot of work to provide the handshaking necessary to create a reliable connection.

Many microcontroller manufacturers have software libraries that support TCP and UDP. However, most of these implementations have documented limitations that may cause increased overhead to transmit and receive data using Internet protocols. The following bullets list some of these limitations documented by a manufacturer for their UDP and TCP software implementations.

- The UDP & TCP headers contain a checksum field. This checksum is computed for the data in the packet, yet it is transmitted in the header. This implementation transmits the packet with an incorrect checksum, calculates the checksum as the packet is sent, and sends the packet again with the correct checksum.
- The implementation does not handle fragmented packets meaning that packets that arrive out of sequence are handled as sequential packets. This causes the resulting data to be corrupted. The proposed method assumes that if the packet size is minimized to 256 bytes, then fragmentation is not likely to occur.
- The implementation does not send a terminate acknowledge packet, which forces the host on the other end to time-out in order to recognize that the connection was terminated.

Rather than implement a software protocol stack that has these type of limitations, the designer may choose to use an external TCP/IP stack device. This device should relieve the designer of working around limitations and allowing more capabilities to be built into the product. The TCP/IP stack device should minimally implement the Network Access Layer, Internet Layer, and the Transport Layer as shown in Figure 2, freeing up program memory to implement protocols for the Application Layer. A minimal software implementation of the Point-to-Point Protocol (PPP), Internet Protocol (IP), ICMP, and UDP/TCP would require approximately 5 Kbytes of program memory and at least 4 Kbytes of data memory. The now vacant program memory could be used for Simple Mail Transfer Protocol (SMTP), Post Office Protocol version 3 (POP3), or Hyper Text Transfer Protocol (HTTP).

The S-7600A TCP/IP Stack IC from Seiko Instruments was designed specifically for this type of market. It integrates the TCP/IP Stack engine, 10 Kbytes of RAM, microcontroller interface and UART into a single chip. Once configured, it acts like a data buffer. Data to be transmitted, up to 1024 bytes, is stored in the internal RAM buffer and the TCP/IP engine appends the various headers and checksums. It then transmits this packet from the UART. When packets are received, the TCP/IP engine determines if the IP address and port number match those set during configuration, calculates and verifies the checksums, and transfers the data contents of the packet to a buffer. It then uses interrupt lines to indicate there is data available to the microcontroller.

The complete list of features for the S-7600A are:

- Implements PPP, IP, TCP, UDP and PAP
- Two general purpose sockets
- Two parallel interfaces (68K/x80 Motorola/Intel MPU bus) or synchronous serial interface
- On-chip UART physical layer transport interface
- 256 kHz typical operating frequency
- Low power consumption (0.9 mA typical, 1.0 µA standby)
- 2.4 V to 3.6 V operating voltage range

The designer now has full Internet capabilities without any of the limitations of the software implementations. The bulk of the program memory on the microcontroller can now be used for the main application and also for implementing some of the Application Layer protocols previously described. The size of the program memory is now dependent on the application and can be scaled accordingly. The S-7600A delivers Internet capability to the bulk of microcontrollers that were previously constrained due to program and data memory size.

The data sheet for the S-7600A can be downloaded from their website at http://www.seiko-usa-ecd.com. There is also a designer's kit available in the form of a PC card. It includes some software library routines to control and interact with the S-7600A. Besides the data sheet, there are manuals for exact protocol implementations, software API for the developer's kit, and example source code available for download.

MODEM

The last key ingredient is the communications medium used to connect to the Internet. The type of medium selected is highly application dependent but can include:

- Wireless
- Cellular
- Power Line Carrier
- POTS (Standard telephone lines)

All of these mediums require some infrastructure to be in place before the embedded device can communicate. Both wireless and cellular transceivers require antennas to be placed in the surrounding area to provide the communication channel. While most areas have some of this infrastructure in place, there are areas that are not completely covered. Everyone has probably experienced this with a cellular telephone at one time or another. Power line carrier also requires infrastructure to be in place. There has to be some sort of transceiver at the other end of the power lines that communicates with the embedded application. This infrastructure for this technology does not currently have the widespread use that wireless or cellular offer and therefore, the costs to build this infrastructure would be substantial. The standard telephone lines are everywhere. The telephone poles, wiring, relay stations, etc., are already in place. The cost of building the infrastructure is zero and therefore, makes the most sense for the bulk of embedded applications. There will be applications where the other mediums are needed, but the application will be able to justify and therefore, absorb the additional costs associated with using the respective mediums.

Usually, the telephone modem technology is significantly less expensive than that of other mediums. It also fits better into embedded applications due to size and power consumption. The key to modem selection is to find one that is highly integrated, i.e., smaller in size. This is important due to the fact that most embedded applications are small. This modem should be easy to use and provide all the necessary features in a single package. Fortunately, the folks at Silicon Laboratories have developed an embedded modem that may be one of the best designs ever to hit the streets. The first feature that stands out is the size. Figure 3 shows the size of the Silicon Labs design compared to a standard modem design. NO relays. NO optoisolators. NO transformers. This design has the Si2400 modem and the Si3015 DAA chip and some passive devices (resistors, capacitors, diodes and transistors). The secret is in the ISOcap^{TM} Technology, used to transfer control and data over a capacitive interface as shown in Figure 4. This interface provides a low cost solution based off standard high voltage capacitors and over 2400 V of isolation. The cost to implement this design in volume should be less than $9.00.

After recovering from the size shock, the impressive list of features makes this product a winner.

The list includes:

- AT command set support
- V.21/Bell 103, V.22/Bell 212A, V.22bis, V.23, & SIA support
- Caller ID detection and decode
- Call progress support
- Voice codec
- Globally programmable DAA (FCC, CTR21, NET4, JATE, etc.)
- Parallel phone detect
- Wake-up on ring
- Overcurrent detection

Never before has the embedded control electronics industry seen a modem design this integrated AND this small AND this CHEAP! But one question remains; why 2400 baud? Isn't that baud rate a little slow to use for Internet applications, even embedded Internet applications? The answer is quite simple. If the application was a web server, then yes, a 2400 baud modem is not practical. But it was already established that a web server was not practical for the embedded world. A typical embedded application will only transfer several hundred bytes of data. When looking at the complete connect and transfer time of one session, a 2400 baud modem will connect in approximately three seconds and upload 200 bytes of data in 0.833 seconds (200 bytes x 10 bits/byte x 1s/2400 bits) for a total of 3.833 seconds. A 56K modem will connect in approximately 15 seconds and transfer 200 bytes in 0.036 seconds (200 bytes x 10 bits/byte x 1s/56000 bits). This calculation shows that a 2400 baud modem can connect to the ISP, dump the data to the server and disconnect before the 56K modem even establishes a connection to the modem on the other end of the line. It just doesn't make sense, especially when you consider the price of the 2400 baud modem versus the 56K modem.

Another feature of a telephone-based system is choosing the ISP to make the Internet connection. Everyone hears about the high speed Internet links such as cable modems. Most providers are targeting customers that want high-speed access for web browsing. According to the estimates, this market which itself is very large, will be dwarfed by the embedded devices. Some companies are starting to realize this fact and are catering towards these embedded applications with low speed modems. One such company is iReady with their iready.net service. It caters to all facets of Internet connectivity, but includes the service for embedded low speed modem applications.

The data sheet for the Si2400/Si3015 can be downloaded from the Silicon Laboratories website at http://www.silabs.com. They also make a modem evaluation board that has a complete modem implementation and an RS-232 interface for use with a PC. The user's manual for the evaluation board is also available from the website and provides some suggested layout guidelines.

FIGURE 3: MODEM SIZE COMPARISON

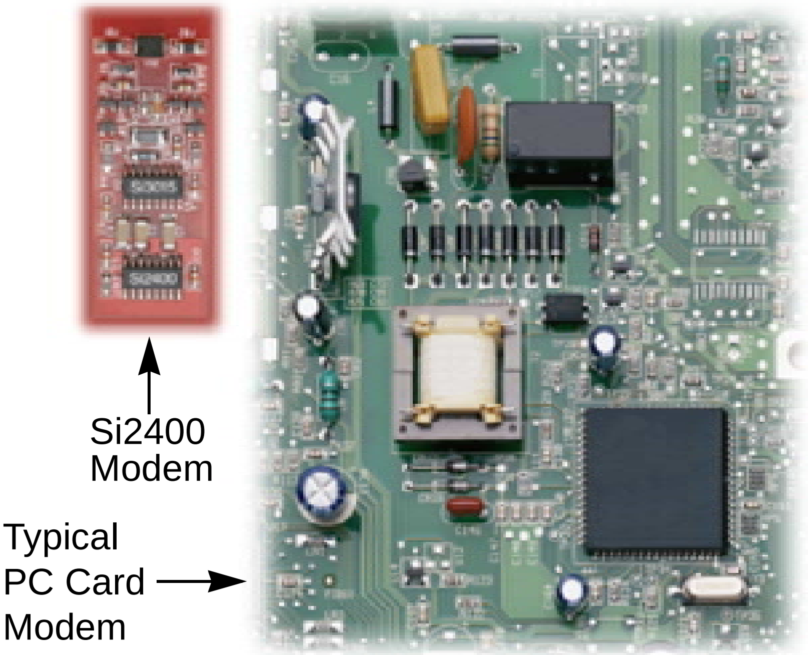
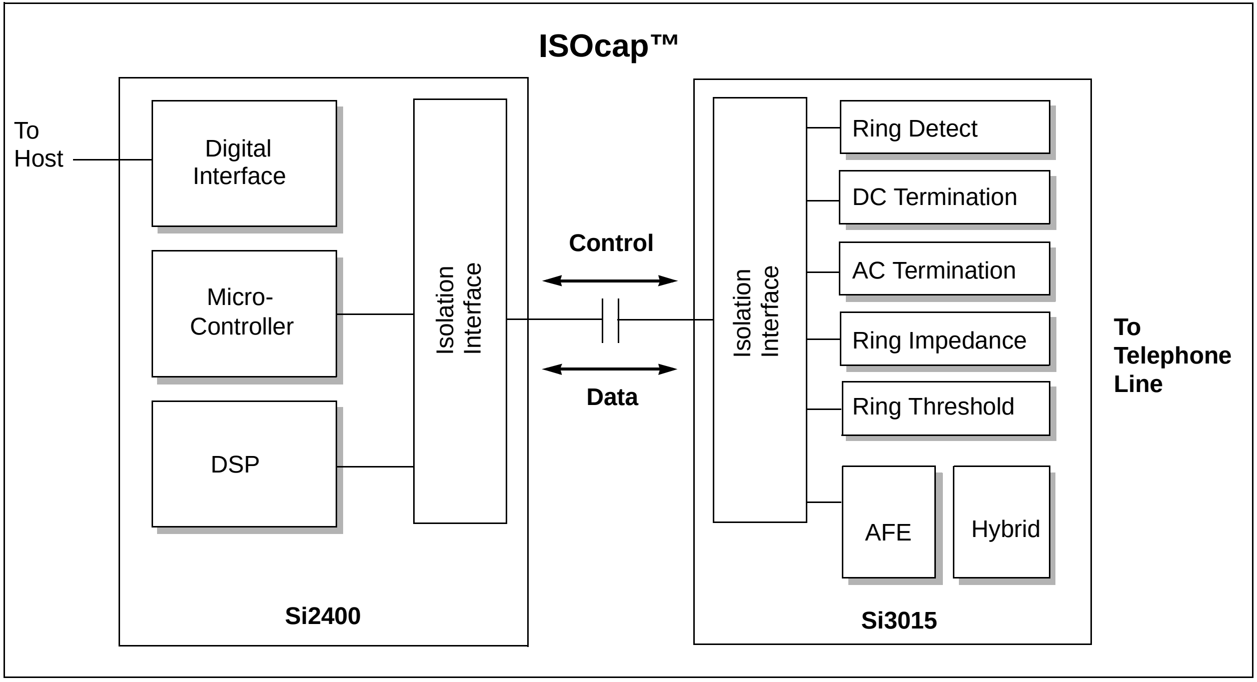


FIGURE 4: ISOcap™ INTERFACE



WEB SERVER APPLICATION

The embedded web server application is more for show and tell. As mentioned before, it is not really a practical use of the hardware. The memory sizes required to serve web pages and data files far outweighs that which can be found on a typical microcontroller. In fact, if the price of non-volatile semiconductor memory and that of hard drives were compared, the results would show that the average price per megabyte of FLASH memory is approximately $1.00 – $2.00 and approximately $0.01 – $0.05 for hard drives. That equates to a ratio of 40:1 favoring hard drives.

Demonstrations of embedded web servers are just that, demonstrations of Internet connectivity. They are easy to design and require nothing more than a web browser and a phone line to demonstrate the capabilities. Demonstrating a client application such as a vending machine is more difficult. Toting around a vending machine in your car for product demonstrations really impacts your gas mileage. It's heavy, too.

Appendix B shows the schematic for the embedded web server. It uses the PIC16F877, the S-7600A TCP/IP stack IC, the Si2400 modem and Si3015 DAA. The design uses 24LCxx serial EEPROM that comes in sizes from 16 bytes up to 32 Kbytes. It holds the ISP phone number, user name and password, and the HTML web page. Also included are a potentiometer and a DS1721 temperature IC. This design can be found using the link: http://198.175.253.101. It takes up to 30 seconds or more for the web page to load. Remember that we are transmitting several thousand bytes of information over the Internet at 2400 baud.

The schematic for everything but the modem and serial port interface is shown on the schematic in Appendix B. This design was actually built around a modem evaluation board from Silicon Laboratories. Since the modem must meet FCC or other governing body regulations, the schematics are not provided for the modem evaluation board. The schematics and layout considerations for the Si2400/Si3015 can be obtained from the Silicon Laboratories website.

The source code and flowchart for the web server are in Appendix C. The source code was written using the C-Compiler from Custom Computer Services Inc. (CCS). More information about the compiler can be found on their website at http://www.ccsinfo.com. The web server has two modes of operation. One is the standard web server mode where the device makes a phone call to the local ISP and establishes an IP address. The user may access the web page by typing in the IP address displayed on the LCD display into any web browser. It takes 30 to 40 seconds for the web page to load. The web page shows some variable information, such as number of hits, temperature where the web server is, IP address, and a potentiometer setting. This information is dynamically inserted into the web page as it is transmitted.

The other mode is a configuration mode, that allows the ISP information and web page to be downloaded into the serial EEPROM. The ISP information includes the phone number, user name and password. The size of the serial EEPROM is application dependent. It can range from 16 bytes (24LC00) to 32 Kbytes (24LC256). The board is currently using a 16 Kbyte device, the 24LC64. To configure the web server, the RS-232 interface on the modem evaluation board is connected to the USART module on the PIC16F877.

Any terminal program will work, as the PIC16F877 displays a text menu that allows the user several options:

1. Enter user name
2. Enter password
3. Enter phone number
4. Download HTML file
5. Or exit configuration mode

Each piece of information has specific memory locations reserved in the serial EEPROM. 32 bytes are reserved for both the user name and password. The ISP phone number takes an additional 16 bytes. Finally, 32688 bytes are available to store the HTML file. The serial interface must use hardware flow control, otherwise the USART buffer will be overrun, due to the programming time required by the serial EEPROM.

The application provides lots of status information. Messages are displayed when a call is being made, a dial tone is detected, a ring occurs, a busy phone line is detected, or when the modem finally makes a connection. The display will show the IP address once the modem has made the connection. There are also a couple of LEDs that indicate the status of the web server. The first LED shows if the modem is connected. The second LED flickers when the web server is being accessed.

Special characters are used to allow the web server to insert variable data into the web page. The following information can be displayed when these characters are found in the HTML web page:

- %a displays the IP address for web page reload function
- %c inserts the current temperature in degrees Celsius
- %f inserts the current temperature in degrees Fahrenheit
- %h displays the number of times the web site has been accessed
- %p inserts the current value of the potentiometer

Several modifications had to be made to the modem evaluation board. It was originally designed to interface the serial port on the PC directly to the modem. Using a terminal program, the user could make or receive phone calls. This interface was hacked to allow stacking of the control board on top of the modem evaluation board. These modifications include:

1. Drill holes through the traces from U3 pins 24 and 25 and remove residual traces
2. Cut the copper away from the holes on the bottom
3. Cut the traces on the bottom of the board going to JP3 pins 5 and 7
4. Lift U1 pin 7
5. Connect a jumper wire from U3 pin 24 through hole to JP3 pin 7
6. Connect a jumper wire from U3 pin 25 through hole to JP3 pin 5
7. Remove shorting jumpers from JPIO 1 and 3 and JP4

The controller board is then bolted directly on top of the evaluation board and makes the necessary connections from the S-7600A to the modem, the modem to the microcontroller, and the microcontroller to the RS-232 interface IC.

CLIENT APPLICATION

This application represents a typical embedded Internet application, where the embedded device is the client and is capable of connecting to a server to upload information and download new information, or firmware. In this case, a vending machine that receives new information. The vending machine application has an LCD display that shows the current items in the machine and the price for each. It will “dispense” items until the machine is empty. At any time, a push-button switch may be pressed to start the connection to the server via the Internet. The modem dials an ISP, makes a connection to the server, and receives new names and prices of items to be dispensed.

The hardware design is a subset of the web server application. Appendix D shows the schematic for the client application. This design removes the serial EEPROM, potentiometer and temperature IC. It adds some push-button switches for the additional user interface required. It uses the same modem evaluation board as the web server with all the same modifications.

The source code and flowchart for the client is given in Appendix E. This source code was also written using the CCS C-Compiler. The vending machine has two modes of operation. It has the standard operating mode of reading the push-button switches and “dispensing” items based on which button is pressed. It tracks the number of items remaining in the vending machine and the total amount of money collected. The second mode of operation is the Internet connection. Most of the code to interface with the S-7600A is the same as the web server, with the exception that it is now a TCP client instead of a TCP web server. It must also know what the IP address and port number of the server is before it can make a connection to the server. This means that more than a web browser is required to complete the connection on the server side. There must be a program running on the server that listens to a port. Once connected, the transfer of information may take place between the client and the server.

In this application, the Internet connection provides the names and prices of new items. Every connection to the server downloads two new item names and prices that are then programmed into data EEPROM. Since these are values that could change frequently, the data EEPROM was used for nonvolatile storage of the information, due to the higher endurance. The same methods presented here can be used in conjunction with the bootloader of AN732 to download new source code into FLASH program memory.

The data that is transferred between the client and the server has some handshaking built in. Once the connection is established, the client waits for a response from the server. The value of the data is not important, only the response from the server, so the buffer is emptied without any processing of the information. The client now responds with an index number between 0 and 9. This index is used by the server to extract the next vending machine item names and prices out of a database. The format of this data is ~ ~ # ~ ~ where # is the index value 0-9. The server will then respond with the new names and prices in the following format:

~ <name1>; <name2>; <price1>; <price2>;

The tilde character is used to denote the start of the string. Each of the names and prices is a null terminated ASCII string and they are delimited using semicolons. Once the client receives this information and updates the data EEPROM, the connection with the server is terminated. At this point, the client must be reset through a $\overline{\text{MCLR}}$ Reset, or by cycling the power. It now switches back to the normal mode of operation, using the new names and prices provided by the web server. Other information, such as total amount of money collected and the number of remaining items, could have been transmitted back to the server, but the application was kept simple for both the client and the server. This interaction is highly application dependent and can be easily adapted based on the system requirements.

CONCLUSION

The move to embed the Internet is creating many new and fascinating devices for all different types of markets. Cellular phones and PDAs are the latest devices to add Internet capability. Soon many household appliances, such as refrigerators, will have Internet capability and these embedded applications will dominate the Internet. These devices can Internet-enable any application that has already used most of the available microcontroller resources to control the application. In most cases, a microcontroller cannot afford to dedicate 5 Kbytes of program memory and a significant portion of data memory for Internet connections. This need has created devices such as the S-7600A and Si2400 specifically for the embedded Internet market.

FURTHER REFERENCE

RFC1332, The PPP Internet Protocol Control Protocol

RFC1334, PPP Authentication Protocols

RFC1661, The Point-to-Point Protocol

RFC1662, PPP in HDLC-like Framing

RFC1700, Assigned Numbers

RFC793, TCP

RFC821, SMTP

RFC1725, POP3

RFC959 FTP

RFC792 ICMP

RFC791 IP

RFC768 UDP

"Internetworking with TCP/IP", Prentice-Hall, 1995, Douglas E. Comer

APPENDIX A: GLOSSARY

ARP Address Resolution Protocol: The TCP/IP protocol that translates an Internet address into the hardware address of the network interface hardware.

client A program that requests services from a server.

client/server A style of computing that allows work to be distributed across hosts.

DNS Domain Name System: The name/address resolution service that uses a distributed database containing address. DNS makes it easier to refer to computers by name rather than numeric address (www.microchip.com -- instead of 198.175.253.68)

FTP File Transfer Protocol: A TCP/IP application, service, and protocol for copying files from one computer to another.

HTML HyperText Markup Language: The language used to write pages for the Internet.

HTTP HyperText Transfer Protocol: The TCP/IP protocol for transferring pages across the Internet.

ICMP Internet Control Message Protocol: The TCP/IP protocol used to report network errors and to determine whether a computer is available on the network.

Internet The international collection of internets that use TCP/IP to work together as one immense logical network.

IP One of the two main parts of the TCP/IP protocol suite. IP delivers TCP and UDP packets across a network.

IP Address A 32-bit unique numeric address used by a computer on a TCP/IP network.

LCP Link Control Protocol: The protocol that negotiates the parameters used by the link between two computers and is protocol within PPP.

NCP Network Control Protocol: The protocol within PPP that negotiates the type of network connection made by the two computers.

POP3 Post Office Protocol version 3: The protocol that you use to download e-mail from a POP3 mail server to your computer.

port A number used by TCP and UDP to indicate which application is sending or receiving data.

PPP Point-to-Point Protocol: A protocol that provides a serial line connectivity (that is, a dial-up with a modem) between two computers, between a computer and a network, or between two networks. PPP can handle several protocols simultaneously.

protocol Rules and message formats for communication between computers in a network.

protocol layers The divisions of a hierarchical network model. Each layer performs a service on behalf of the layer directly above it. Each layer receives services from the layer directly below it.

protocol stack A group of protocols that work together across network layers.

server A computer program that provides services to clients, and/or the computer that runs the server program.

SMTP Simple Mail Transfer Protocol: The TCP/IP protocol for sending and receiving e-mail across a network.

socket A data structure that allows programs on an internet to communicate. It works as a pipeline between the communicating programs and consists of an address and a port number.

TCP Transmission Control Protocol: One of the two principal components of a TCP/IP protocol suite. TCP puts data into packets and provides reliable packet delivery across a network (packets arrive in order and are not lost).

UDP User Datagram Protocol: A TCP/IP protocol found at the network (internet) layer, along with the TCP protocol. UDP sends data down to the internet layer and to the IP protocol. Unlike TCP, UDP does not guarantee reliable, sequenced packet delivery. If data does not reach its destination, UDP does not retransmit as TCP does.

Most definitions provided by the book TCP/IP for Dummies 3rd Edition by Candace Leiden and Marshall Wilensky and published by IDG Books Worldwide, Inc. ISBN 0-7645-0473-8

APPENDIX B: EMBEDDED WEB SERVER SCHEMATICS

FIGURE B-1: TCP/IP SERVER MODEM (SHEET 1 OF 3)

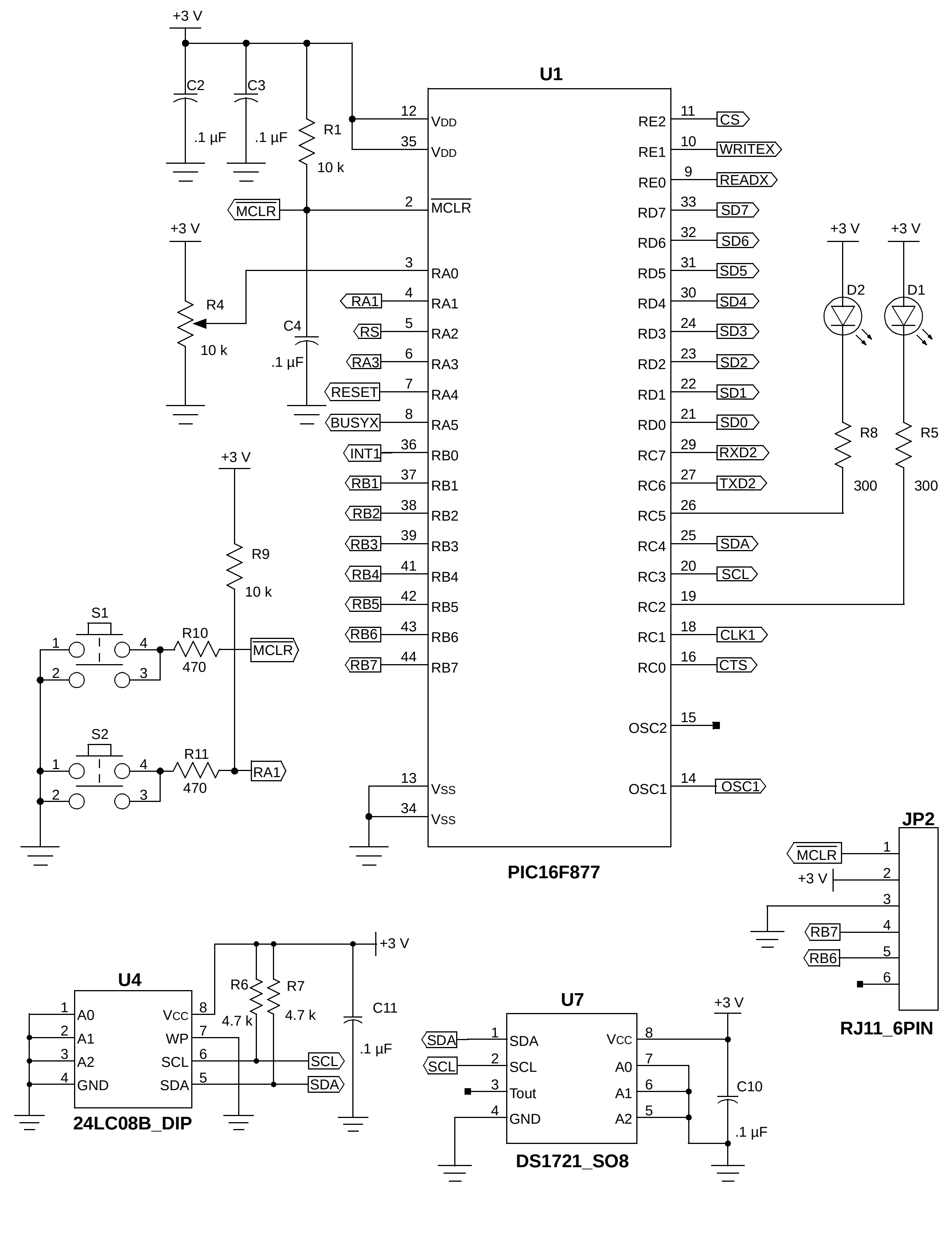


FIGURE B-2: TCP/IP SERVER MODEM (SHEET 2 OF 3)

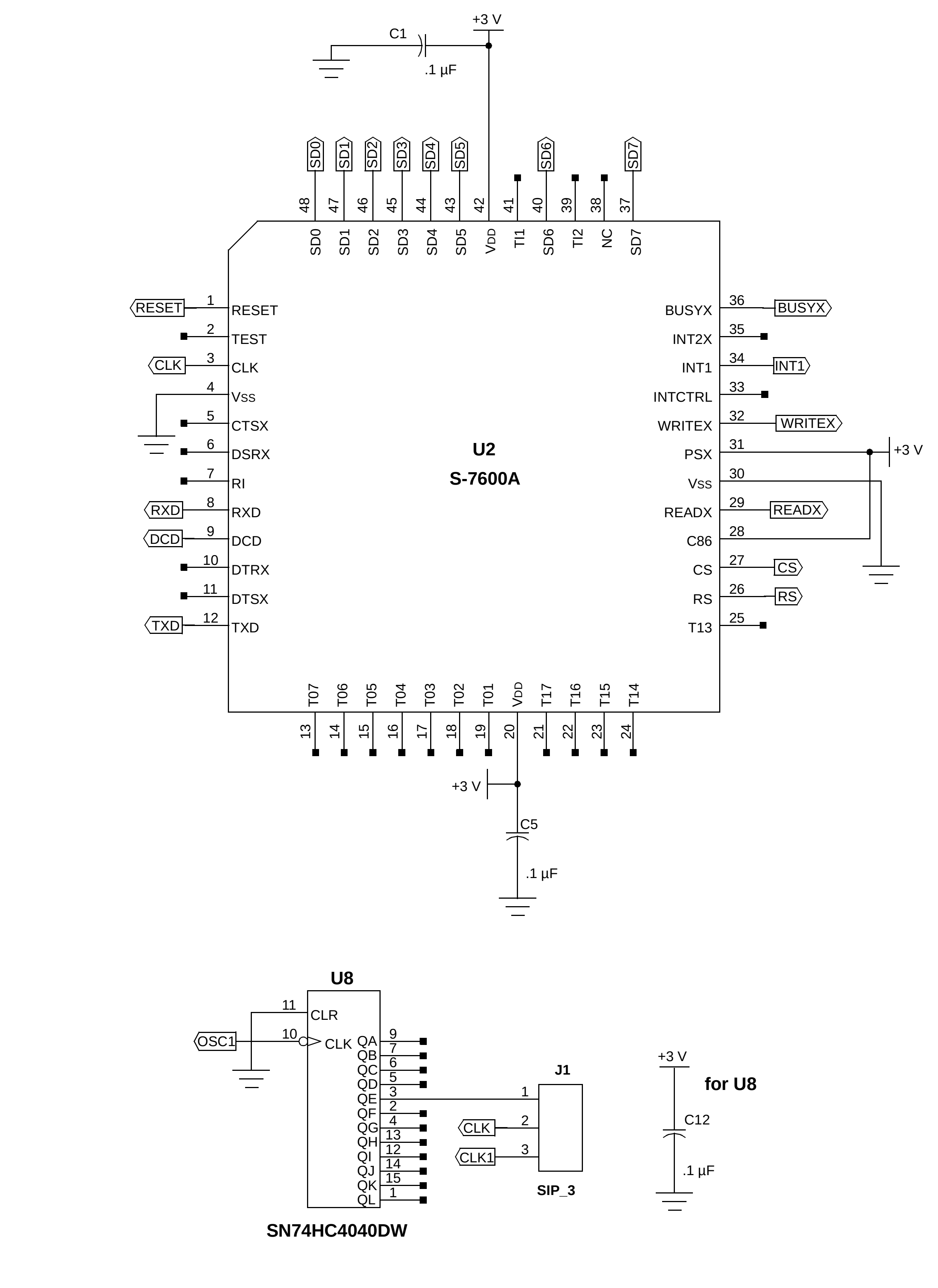


FIGURE B-3: TCP/IP SERVER MODEM (SHEET 3 OF 3)

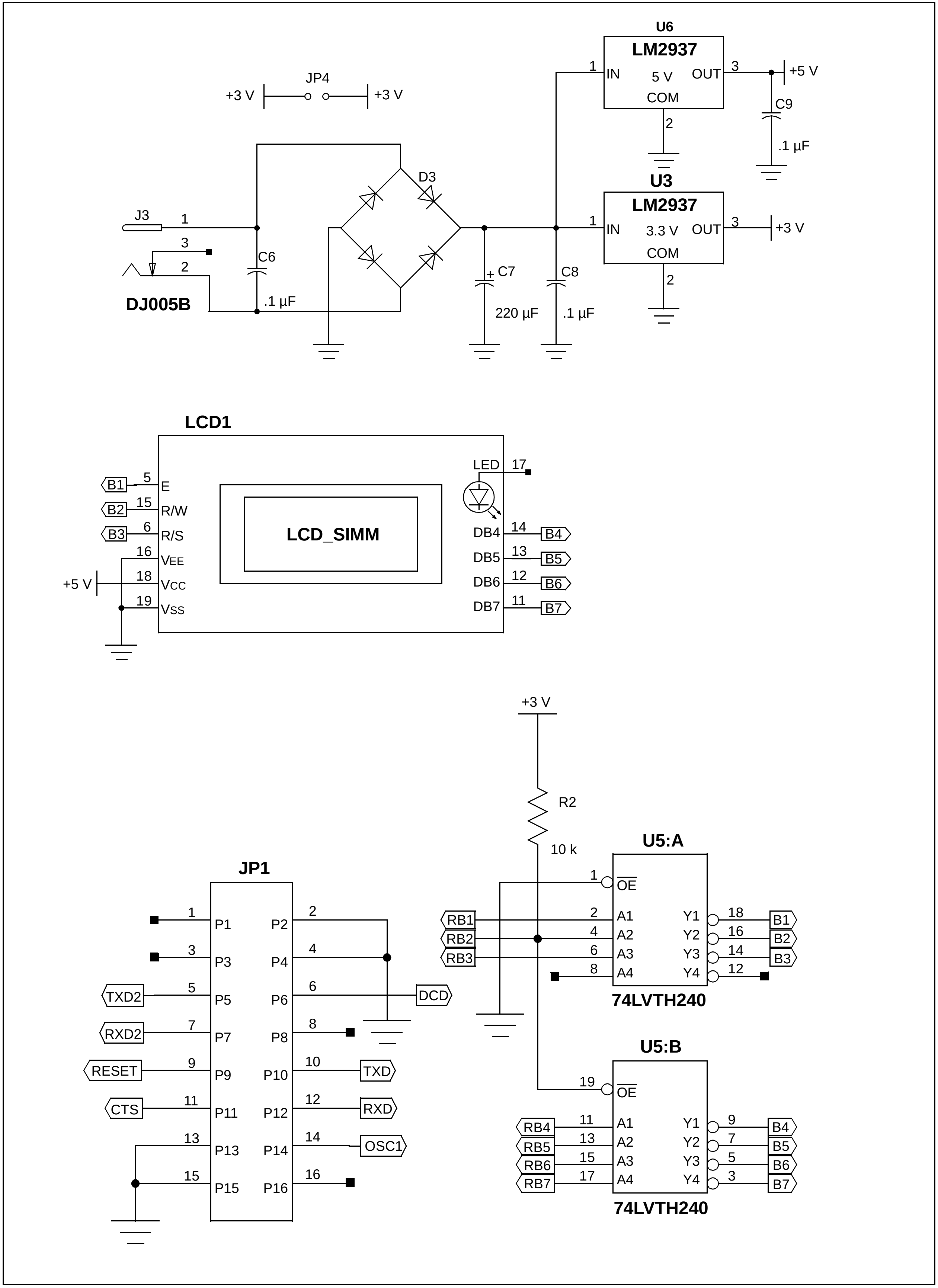


FIGURE B-4: WEB SERVER FLOWCHART (SHEET 1 OF 3)

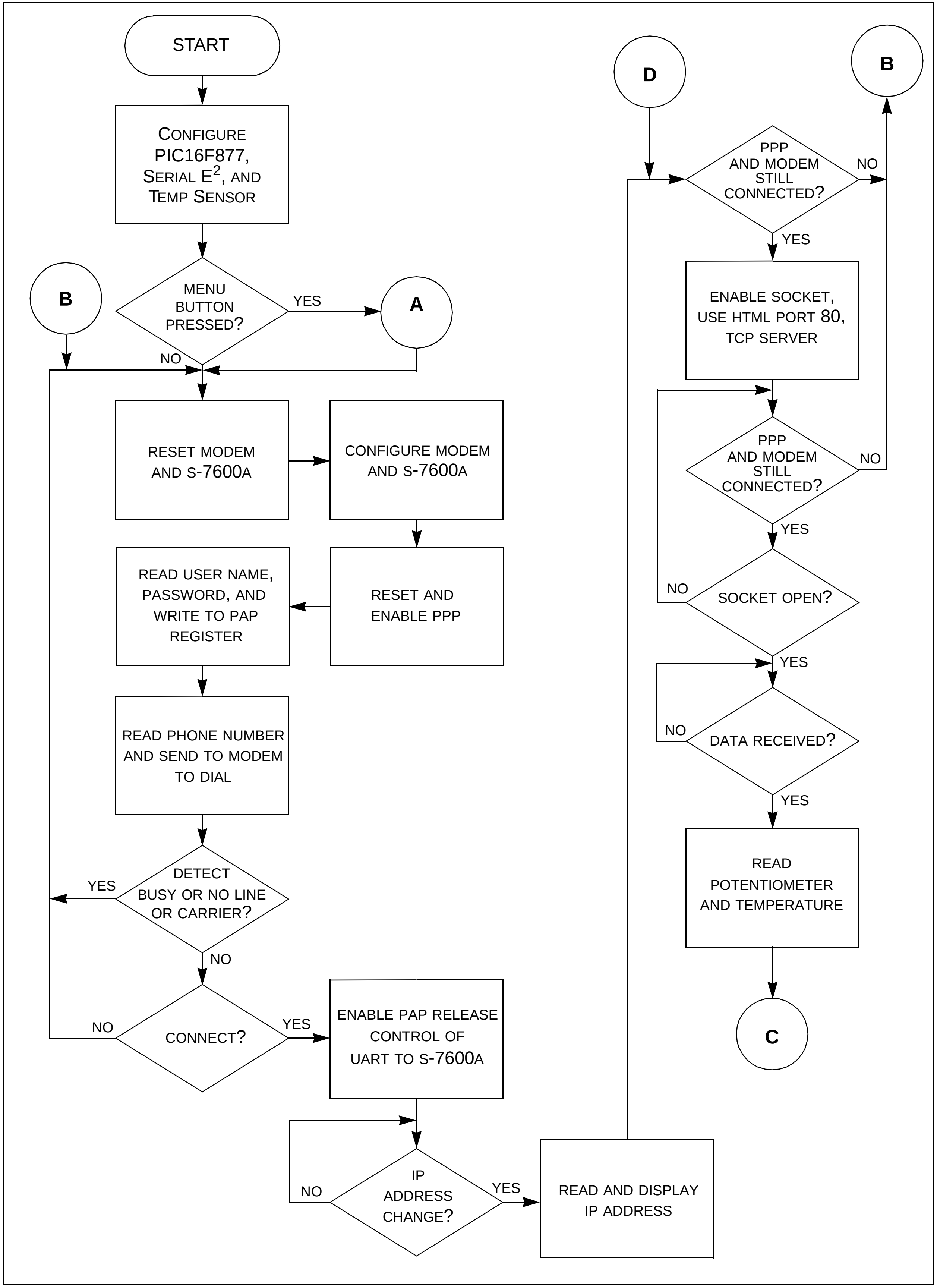

START
CONFIGURE PIC16F877, SERIAL E2, AND TEMP SENSOR
B
MENU BUTTON PRESSED?
YES
A
NO
RESET MODEM AND S-7600A
CONFIGURE MODEM AND S-7600A
READ USER NAME, PASSWORD, AND WRITE TO PAP REGISTER
RESET AND ENABLE PPP
READ PHONE NUMBER AND SEND TO MODEM TO DIAL
YES
DETECT BUSY OR NO LINE OR CARRIER?
NO
NO
CONNECT?
YES
ENABLE PAP RELEASE CONTROL OF UART TO S-7600A
NO
IP ADDRESS CHANGE?
YES
READ AND DISPLAY IP ADDRESS
D
B
PPP AND MODEM STILL CONNECTED?
NO
YES
ENABLE SOCKET, USE HTML PORT 80, TCP SERVER
PPP AND MODEM STILL CONNECTED?
NO
YES
NO
SOCKET OPEN?
YES
NO
DATA RECEIVED?
YES
READ POTENTIOMETER AND TEMPERATURE
C

FIGURE B-5: WEB SERVER FLOWCHART (SHEET 2 OF 3)

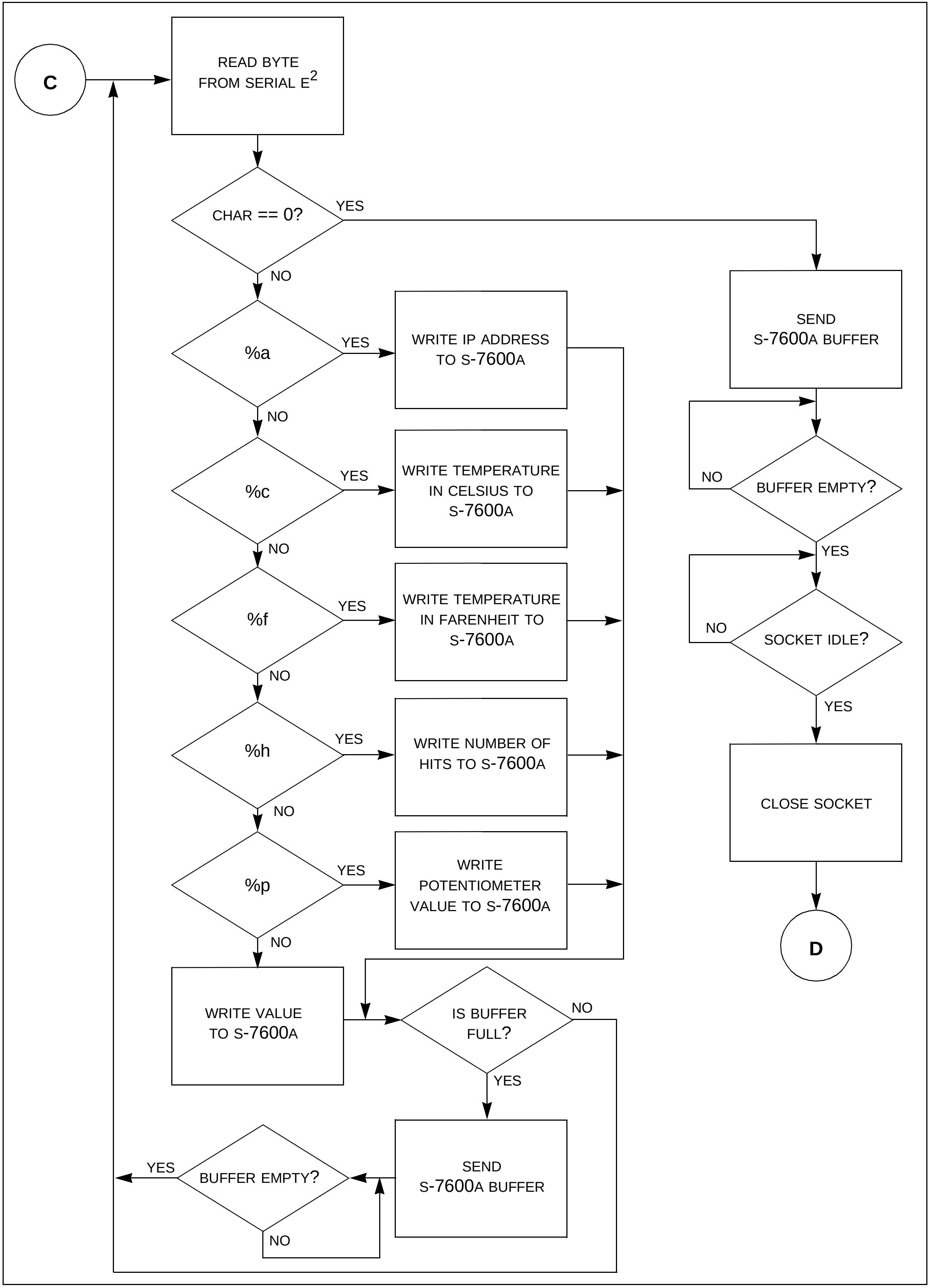
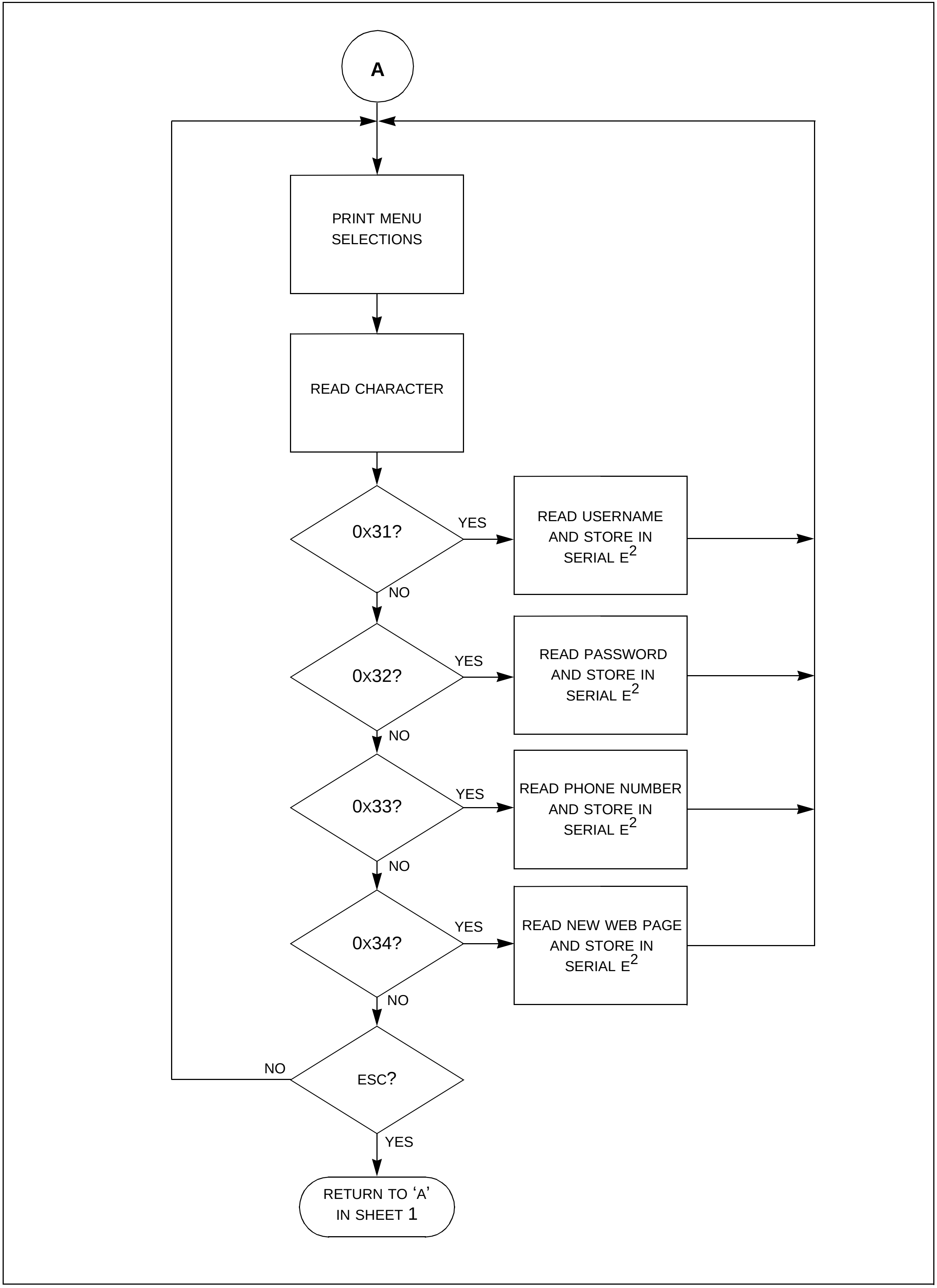


FIGURE B-6: WEB SERVER FLOWCHART (SHEET 3 OF 3)


A
PRINT MENU SELECTIONS
READ CHARACTER
0x31?
YES
READ USERNAME AND STORE IN SERIAL E2
NO
0x32?
YES
READ PASSWORD AND STORE IN SERIAL E2
NO
0x33?
YES
READ PHONE NUMBER AND STORE IN SERIAL E2
NO
0x34?
YES
READ NEW WEB PAGE AND STORE IN SERIAL E2
NO
ESC?
NO
YES
RETURN TO 'A' IN SHEET 1

APPENDIX C: EMBEDDED WEB SERVER SOURCE CODE

```
/***************************************************************************
*  Filename: MODEM_C.C
****************************************************************************
*   Author: Stephen Humberd/Rodger Richey
*   Company:Microchip Technology
*   Revision:RevA0
*   Date:  5-24-00
*   Compiled using CCS PICC
****************************************************************************
*   Include files:
*       16f877.h
*       f877.h
*       s7600.h
*       ctype.h
*       string.h
*       lcd_cut.c
*       seiko_ct.c
*       eeprom.c
****************************************************************************
*
*  This program demonstrates a simple WEB severer using a Microchip PIC16C74B
*  microcontroller and a Seiko S-7600 TCP/IP Network Protocol chip.  This
*  is the main file that includes all other header files and source code files.
*
*  External Clock provided by Si2400 modem = 9.8304MHz
*
***************************************************************************/

#include <16c77.h>
#include “f877.h”
#include “s7600.h”
#include <ctype.h>
#include <string.h>

#fuses HS,NOWDT,NOPROTECT,NOBROWNOUT

#define EEPROM_SDA  PIN_C4
#define EEPROM_SCL  PIN_C3
#define hi(x) (*(&x+1))///

#use delay(clock=9828606)
#use rs232(baud=2400, xmit=PIN_C6, rcv=PIN_C7)
#use standard_io(C)
#use i2c(master,sda=EEPROM_SDA, scl=EEPROM_SCL)

#define EEPROM_ADDRESS long int
#define EEPROM_SIZE    1024
#define esc 0x1b// ESC char
```

```
// PORTA bits
#bit RS     = PORTA.2
#bit RESET  = PORTA.4
#bit BUSY   = PORTA.5

// PORTB bits
//#bit INT1 = PORTB.0
// As defined in the following structure the pin connection is as follows:
//     B1  enable
//     B3  reg_sel
//     B2  rd_w
//     B4  D4
//     B5  D5
//     B6  D6
//     B7  D7

#define lcd_type 2           // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40    // LCD RAM address for the second line

byte lcd_cmd, cmd_tmp;
byte CONST LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6};
                             // These bytes need to be sent to the LCD
                             // to start it up.

// PORTC bits
#bit CTS    = PORTC.0// When CTS = 0 send is enabled
#bit LED1   = PORTC.2// When LED = 0 the LED is ON
#bit LED2   = PORTC.5// When LED = 0 the LED is ON

// PORTE bits
#bit READX  = PORTE.0
#bit WRITEX = PORTE.1
#bit CS     = PORTE.2

char i,j,ch,addr,temp,pot,ftmp,ctmp,count,count1,page;
long index, byt_cnt, hits;
char Login[10];
char MyIPAddr[4];
char user[33];
char pass[33];
char phone[17];
char read_data[5];
char test_str[7];
char read_Q[7];

struct lcd_pin_map {                 // This structure is overlayed
           boolean unused;           // on to an I/O port to gain
           boolean enable;           // access to the LCD pins.
           boolean rd_w;             // The bits are allocated from
           boolean reg_sel;       // low order up.  UNUSED will
           int   data : 4;        // be pin B0.
        } lcd;

#byte lcd = 6                        // This puts the entire structure
                                     // on to port B (at address 6)

STRUCT lcd_pin_map const LCD_WRITE = {1,0,0,0,0}; // For write mode all pins are out except RB0
(int1)

#include “lcd_cut.c”// LCD routines
#include “seiko_ct.c”// Seiko routines
```

```
/**************************************************************
**  char DataAvailable(void)                                 **
**  Determines if there is any data available to read out of **
**  the S-7600A.                                             **
**  Returns the value of the data available bit from the     **
**  S-7600A.                                                 **
**************************************************************/
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);  // Check if S-7600A is busy
    CS = 1;        // 1st cycle sets register
    RS = 0;        // address
    WRITEX = 0;
    PORTD = Serial_Port_Data;// Write to serial port
    TRISD = 0;
    READX = 1;
    READX = 0;
    WRITEX = 1;
    RS = 1;
    CS = 0;

    CS = 1;        // 2nd cycle writes the data
    WRITEX = 0;    // to the register
    PORTD = data;  // Data to write
    READX = 1;
    READX = 0;
    WRITEX = 1;
    CS = 0;

    TRISD = 0xff;
}


/**************************************************************
**  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(0x22)&0x20))
    {
        WriteSeiko(TCP_Data_Send,0);     // If full send data
        while(ReadSeiko(Socket_Status_H));// Wait until done
    }

    while(!BUSY);                        // Check if S-7600A is busy
    CS = 1;                              // 1st cycle sets register
    RS = 0;                              // address
    WRITEX = 0;
    PORTD = Socket_Data;                 // Write to socket
    TRISD = 0;
    READX = 1;
    READX = 0;
    WRITEX = 1;
    RS = 1;
    CS = 0;
    CS = 1;                              // 2nd writes the data to
    WRITEX = 0;                          // the register
    PORTD = data;                        // Data to write
    READX = 1;
    READX = 0;
    WRITEX = 1;
    CS = 0;

    TRISD = 0xff;
}

#include “eeprom.c”                    // external serial EEPROM routines


/**************************************************************
**  void Get_username(void)                                  **
**  Requests and reads the user name from the input terminal.**
**************************************************************/
void Get_username(void)
{
    char n_tmp;
    i=0;
    printf(“%c[2J”,esc);// Print request to termainal
    printf(“%c[12;20H 32 chars max”,esc);
    printf(“%c[10;20H Enter user name: “,esc);

    while(1)   // Read characaters until a
    {              // CR is read or 32 chars
        user[i]=0;// have been read
        ch=GETC();
        if(ch==0x0D)
            break;
        putc(ch);
        if(ch != 0x08)
        {
            user[i]=ch;
            i++;
        }
        if(i==32) break;
    }

    // write user name to the EEPROM
    n_tmp=0;
    for(i=0;i<0x1F;i++)
    {
        ch=user[i];
        write_ext_eeprom(n_tmp, user[i]);
        n_tmp++;
    }
}


/**************************************************************
**  void Get_password(void)                                  **
**  Requests and reads the password from the input terminal. **
**************************************************************/
void Get_password()
{
    byte a_tmp;
    i=0;
    printf(“%c[2J”,esc);// Print request to terminal
    printf(“%c[12;20H 32 chars max”,esc);
    printf(“%c[10;20H Enter password: “,esc);

    while(1)   // Read characters until a
    {              // CR is read or 16 chars
        pass[i]=0;// have been read
        ch=getc();
        if(ch==0x0D)
            break;
        if(ch != 0x0A)// line feed
        {
            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);// Print request to terminal
    printf(“%c[12;20H 16 chars max”,esc);
    printf(“%c[10;20H Enter phone number: “,esc);

    i=0;
    while(1)   // Read characters until a
    {              // CR is read or 16 chars
        phone[i]=0;// have been read
        ch=getc();
        if(ch==0x0D)
            break;
        if(ch != 0x0A)// line feed
        {
            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 Read_file(void)                                     **
**  Requests and reads the HTML web page that is sent when   **
**  requested by a web browser.                              **
**  This routine strips out all carriage returns and line-   **
**  feeds found in the file.  It also lookds for a semi-     **
**  colon to end the file.                                   **
**************************************************************/
void Read_file()
{
    printf(“%c[2J”,esc);// Print request to the terminal
    printf(“%c[10;20H Ready to receive file”,esc);
    printf(“%c[12;20H 32688 bytes max”,esc);
    printf(“%c[14;20H End file with ;”,esc);
    printf(“%c[16;20H Set your terminal for Hardware Flow Control”,esc);
    ch=1;
    i=0;
    index=0x50;

    while(1)
    {
        CTS = 0;// enable send
        ch=getc();// get character from USART
        CTS = 1;// disable send
        if(index == 32767)// Stop if EEPROM is full
            break;
        if(ch==00)// Stop if NULL is read
            break;
        if(ch==’;’)// Stop if semicolon is read
            break;

        // Otherwise write character to EEPROM
        write_ext_eeprom(index, ch);

        // FLASH status LED to indicate data transfer
        if(bit_test(index,4))// flash LEDs
        {
            LED1 = 1;
            LED2 = 0;
        }
        else
        {
            LED1 = 0;
            LED2 = 1;
        }
        index++;
    }
    write_ext_eeprom(index, 0);// Write terminating NULL
    CTS = 0;   // enable send
    LED1=1;        // turn off LEDs
    LED2=1;

    // Print status of download to EEPROM
    index = index - 80;
    printf(“%c[2J”,esc); // clear screen
    printf(“%c[12;28H Received %lu bytes”,esc,index);
    if(index > 32688)
        printf(“%c[18:20H Error maximum bytes is 32688”,esc);
    printf(“%c[18;25H Press any key to continue”,esc);
    ch=getc();
    CTS = 1;   // disable send
}
```

```
/**************************************************************
**  void Write_data_to_rs232(void)                           **
**  Debugging routine that dumps the contents of the EEPROM  **
**  to the terminal.  Must uncomment line in main().         **
**************************************************************/
void Write_data_to_rs232(void)
{
    // Read and print username, password and phone number
    for(index=0;index<0x50;index++)
    {
        ch = read_ext_eeprom(index);
        temp=isalnum(ch);
        if(temp)
            putc(ch);
        else
            putc(‘.’);
    }

    // Read out web page until a null is reached
    index=0x50;
    ch=1;
    while(ch)
    {
        ch = read_ext_eeprom(index);
        putc(ch);
        temp=ch;
        index++;
    }
    temp = (index>>8);
    ch=1;
}

/**************************************************************
**  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;
    CTS=0;      // enable send
    // Print main menu to terminal, escape terminates
    while(ch != 0x1b)
    {
        lcd_putc(“\fPC Terminal menu”);
        printf(“%c[2J”,esc);
        printf(“%c[8;25H 1   Enter user name”,esc);
        printf(“%c[10;25H 2   Enter password”,esc);
        printf(“%c[12;25H 3   Enter phone number”,esc);
        printf(“%c[14;25H 4   Down load HTML file”,esc);
        printf(“%c[17;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’ -> new web page
            Read_file();
            break;
        }
    }
    CTS = 1;   // disable send
}
/**************************************************************
**  void temp_config(byte data)                              **
**  Configures DS1721 temperature measurement IC using the   **
**  I2C module.                                              **
**************************************************************/
void temp_config(byte data)
{
    i2c_start();// Send start bit
    i2c_write(0x90);// Send address byte
    i2c_write(0xac);// Send command byte
    i2c_write(data);// Send data
    i2c_stop();// Send stop bit
}
/**************************************************************
**  void init_temp(void)                                     **
**  Initializes DS1721 temperature measurement IC using      **
**  the I2C module.                                          **
**************************************************************/
void init_temp(void)
{
    output_high(PIN_B7);// Configure I/O pins
    output_high(PIN_B6);
    i2c_start();// Send start bit
    i2c_write(0x90);// Send address byte
    i2c_write(0xee);// Send command byte
    i2c_stop();// Send stop bit
    temp_config(8);// 9 bit mode was 8
}
/**************************************************************
**  byte read_temp(void)                                     **
**  Reads the temperature from the DS1721.  Value returned   **
**  is degrees F from 0 to 255.                              **
**************************************************************/
byte read_temp(void)
{
    byte datah,datal;
    long data;

    i2c_start();// Send start bit
    i2c_write(0x90);// Send address byte
    i2c_write(0xaa);// Send command byte
    i2c_start();// Send start bit (restart)
    i2c_write(0x91);// Send address byte (w/read)
    datah=i2c_read();// Read a byte
    datal=i2c_read(0);// Read a byte
    i2c_stop();// Send stop bit
    data=datah;// Format received data
    ctmp=data;
    data=data*9;
    if((datal&0x80)!=0)
        data=data+4;
    data=(data/5)+32;
    datal=data;
    return(datal);// Return temperature
}
```

```
/**************************************************************
**  void main(void)                                          **
**************************************************************/
void main(void)
{
    // Intialize PORTs & TRISs
    PORTA = 0x24;//00100100
    PORTB = 0xff;//00000000
    PORTD = 0;//00000000
    PORTE = 0xFA;//11111010

    TRISA = 0xE3;//11100011
    TRISB = 0xff;
    TRISC = 0x98;//10011000
    TRISD = 0xff;//11111111
    TRISE = 0;//00000000

    ADCON1 = 0x06;//00000110all digital to start
    ADCON0 = 0;

    PR2 = 0x07;// Enable PWM
    CCPR2L = 0x03;
    CCP2CON = 0x0C;//00001100
    T2CON = 0x04;//00000100
    T1CON = 0x31;//00110001Timer1

    LED1 = 1;// LED OFF
    LED2 = 1;
    CTS = 1;// disable send
    hits = 0;

//////////////////////////////////////////////////
    init_ext_eeprom();// initalize I2C
    init_temp();// initalize DS1721 temperature chip
    lcd_init();// initalize the LCD

    ch = bit_test(PORTA,1);// test for menu button
    if(ch == 0)
        Menu(); // Show enter menu

    ADCON1 = 0x04;//00000100
    ADCON0 = 0x81;//10000001

//  Write_data_to_rs232();// This routine dumps all data stored in the EEPROM
               // to the PC terminal it is used for debug only

// Program starts here.  Modem is hard reset, configured, and waits
// for a request on the HTTP port #80.
restart:
        LED1 = 1;  // LED OFF
        LED2 = 1;
        RESET = 0; // Reset modem and S-7600A
        delay_ms(1);
        RESET = 1;

        lcd_putc(“\fReseting Modem\n”);
        printf(“\n\rReseting Modem”);
        delay_ms(10000);

        // Set clock divider for 1KHz clock
        WriteSeiko(Clock_Div_L,0x32);
        while(ReadSeiko(Clock_Div_L) != 0x32)// Detects when
            WriteSeiko(Clock_Div_L,0x32);// reset done
        WriteSeiko(Clock_Div_H,0x01);
```

```
// Set Baud Rate Divisor for 2400 baud @ 9.828606
WriteSeiko(BAUD_Rate_Div_L,0x7e);
WriteSeiko(BAUD_Rate_Div_H,0x00);

// Set up MODEM
printf(S_Putc,”ATS07=06SE2=C0\r”);// Send V22bis to modem
delay_ms(10);
printf(S_Putc,”ATE0\r”);// Send Local echo off
delay_ms(10);

WriteSeiko(PPP_Control_Status,0x01);// enable PPP
WriteSeiko(PPP_Control_Status,0x00);// disable PPP
WriteSeiko(PPP_Control_Status,0x20);// set PAP mode
delay_ms(5);

// Determine length of “Username” and write it to the PAP register
// in the S-7600A
ch=1;
i=0;               // Beginning of Username string in EEPROM
while(ch)      // Read until a NULL is reached
{
    ch = read_ext_eeprom(i);
    i++;
}
i--;
WriteSeiko(PAP_String,i);// Write # of chars in username

for(j=0; j<i; j++)// Write “Username” to PAP register
{
    ch = read_ext_eeprom(j);
    WriteSeiko(PAP_String,ch);
}

//Determine length of “Password” and write it to the PAP register
ch=1;
i=0x20;    // Beginning of Password string in EEPROM
while(ch)
{
    ch = read_ext_eeprom(i);
    i++;
}
i--;
i=(i-0x20);
WriteSeiko(PAP_String,i);// Write # of chars in password

for(j=0x20; j<(i + 0x20); j++)// Write “Password” to PAP register
{
    ch = read_ext_eeprom(j);
    WriteSeiko(PAP_String,ch);
}

WriteSeiko(PAP_String,0x00);// Write NULL to PAP register

printf(S_Putc,”ATDT”);
ch=1;          // Read phone # out of EEPROM & write to modem
index=0x40;// beginning of phone number in EEPROM
while(1)
{
    ch = read_ext_eeprom(index);
    if(ch == 0)// end of string
        break;

    S_Putc(ch);
    index++;
}
```

```
printf(S_Putc,”\r”);// end with CR / LF

delay_ms(5);

printf(“%c[2J”,esc);// clear VT100 screen
lcd_putc(“\fCall\n”);// Print message that modem
printf(“\rDialing “);// is dialing

ch=1;          // Write phone # to LCD and terminal
i=0x40;        // beginning of phone number in EEPROM
while(1)
{
    ch = read_ext_eeprom(i);
    if(ch == 0)// end of string
        break;
    printf(“%c”,ch);// write number to RS-232Terminal
    printf(lcd_putc,”%c”,ch);// write number to LCD display
    i++;
}
printf(“\r”);// send with CR / LF

// Read status bytes from modem will dialing
while(1)
{
    i = ReadSeiko(Serial_Port_Config);// Read UART status

    lcd_send_byte(0,0x87);// goto first line char 8
    if(bit_test(i,7))// If data available
        ch = ReadSeiko(Serial_Port_Data);// read
    if(ch == ‘t’)      // dial tone detected
    {
        lcd_putc(“dial tone\n”);
        printf(“\ndial tone\r”);
    }
    if(ch == ‘r’)  // ring detected
    {
        lcd_putc(“ring     “);
        printf(“\nring\r”);
    }
    if(ch == ‘c’)  // modem connected
        break;         // done
    if(ch == ‘b’)  // busy tone detected
    {
        lcd_putc(“busy     “);// reset & start over
        printf(“\nbusy\r”);
        gotorestart;
    }
    if(ch == ‘l’)  // No phone line detected
        gotorestart;
    if(ch == ‘N’)  // No carrier detected
        gotorestart;
    ch = 0;
}

lcd_putc(“\fConnect  “);
printf(“\nConnect\r”);
LED1 = 0;                             // LED1 ON

WriteSeiko(PPP_Control_Status,0x62);// Use PAP, enable PPP
WriteSeiko(Serial_Port_Config,0x01);// turn serial port over to S-7600
delay_ms(5);

// (Register 0x60) wait for PPP to be Up
while(!(ReadSeiko(PPP_Control_Status)&0x01))
    delay_ms(5);
```

```
while(ReadSeiko(Our_IP_Address_L) == 0);// detect when ready to proceed

MyIPAddr[0] = ReadSeiko(Our_IP_Address_L);// Read the assigned IP address
MyIPAddr[1] = ReadSeiko(Our_IP_Address_M);// of the web server
MyIPAddr[2] = ReadSeiko(Our_IP_Address_H);
MyIPAddr[3] = ReadSeiko(Our_IP_Address_U);


// Print address to the terminal and LCD
printf(“\r\nMy address is %u.%u.%u.%u”,MyIPAddr[3],MyIPAddr[2],MyIPAddr[1],MyIPAddr[0]);
lcd_putc(“\fMy address is\n”);
printf(lcd_putc,”%u.%u.%u.%u”,MyIPAddr[3],MyIPAddr[2],MyIPAddr[1],MyIPAddr[0]);


// Main Webserver Loop
while(1)
{
    while(1)
    {
        delay_ms(1);
        // Check to see if PPP is still up or modem has disconnected
        if(!(ReadSeiko(PPP_Control_Status)&0x01))
            gotorestart;
        if(ReadSeiko(Serial_Port_Config)&0x40)
            gotorestart;

        WriteSeiko(Socket_Index,0x00);// Use socket 0
        WriteSeiko(Socket_Config_Status_L,0x10);// Reset socket
        delay_ms(10);

        WriteSeiko(Our_Port_L,80);// open a socket to listen on port 80
        WriteSeiko(Our_Port_H,0);

        // Reg 0x22 TCP server mode
        WriteSeiko(Socket_Config_Status_L,0x06);

        // Reg 0x24 General socket 0 active
        WriteSeiko(Socket_Activate,0x01);
        delay_ms(5);

        // Socket open, wait for connection
        printf(“\n\rSocket open\n\r”);
        i = 2;
        while(1)
        {
            // Check to see if PPP is still up or modem has disconnected
            delay_ms(1);
            if(!(ReadSeiko(PPP_Control_Status)&0x01))
                gotorestart;
            if(ReadSeiko(Serial_Port_Config)&0x40)
                gotorestart;

            // Read socket status and process
            temp = ReadSeiko(Socket_Status_M);
            if(temp&0x10)// set = “Connection established”
            {
                i = 0;
                break;
            }
            else if(temp&0xe0)// Error detected
                break;
            if(temp == 0x09)// Waiting for connection
                continue;// by web browser
            delay_ms(5);
            i++;
            if(i == 255)// Timeout
```

```
                        break;
                }
                if(!i)
                    break;
            }

            if(i == 1)     // Timeout detected
                break;

            printf(“\n\rWaiting for data”);
            WriteSeiko(Socket_Interrupt_H,0xf0);// Clear interrupt
            i=0;

            // Read data send by web browser and dump to terminal
            printf(“\n\rReading data\r\n”);
            while(ReadSeiko(Socket_Config_Status_L)&0x10)
            {
                temp = ReadSeiko(Socket_Data);
                putc(temp);
            }

            // Write HTTP Header (which is nothing, just termination, for httpd/0.9)
            WriteSeiko(Socket_Data,0X0A);
            WriteSeiko(Socket_Data,0X0D);
            WriteSeiko(Socket_Data,0X0A);
            WriteSeiko(Socket_Data,0X0D);

            // Send the main HTML page.
            LED2 = 0;      // LED2 ON
            hits++;            // increment web site hits

            set_adc_channel(1);// Read potentiometer
            pot = read_adc();
            ftmp = read_temp();// Read temperature

            // Dump web page in EEPROM to S-7600A, parse data to insert
            // #hits, temperature, IP address, potentiometer value, etc.
            byt_cnt=0;
            index=0x50;    // beginning of html in EEPROM
            ch=1;
            while(ch != 0) // break on NULL
            {
                ch = read_ext_eeprom(index);// Read a byte
                if(ch == 0)// If NULL, quit
                    break;
                if(ch == 0x25)// look for % replaceable paramaters
                {
                    LED2 ^= 1;// toggle LED to indicate activity
                    index++;
                    ch = read_ext_eeprom(index);// get the next char
                    switch(ch)
                    {
                    case ‘a’:// address
                        printf(W_putc,”%u.%u.%u.%u”,MyIPAddr[3],MyIPAddr[2],MyIPAddr[1],MyI-
PAddr[0]);
                        break;
                    case ‘c’:// temperature C
                        printf(W_putc,”%u”,ctmp);
                        break;
                    case ‘f’:// temperature F
                        printf(W_putc,”%u”,ftmp);
                        break;
                    case ‘h’:// # of hits
                        printf(W_putc,”%lu”,hits);
                        break;
                    case ‘p’:// potentiometer
                        printf(W_putc,”%03u”,pot);
                        break;
                    }
                }
                else                // data not replaceable
                {                   // send to S-7600A
                    count=0;
                    count1=0;
                    bit_clear(PIR1,TMR1IF);// clear interrupt flag
                    // If buffer is full
                    while(0x20==(ReadSeiko(0x22)&0x20))
                    {
                        // Send current buffer and wait
                        WriteSeiko(TCP_Data_Send,0);
                        while(ReadSeiko(Socket_Status_H));
                        {   // check if modem has disconnected
                            if(ReadSeiko(Serial_Port_Config)&0x40)
                                gotorestart;

                            if(bit_test(PIR1,TMR1IF))// if this loops
                            {                            // times out then
                                count++;          // the socket has
                                if(count > 0xf8)// hung or the user
                                {                        // has hit stop
                                   count1++;
                                   count=0;
                                }
                                if(count1 > 0xC0)
                                   goto restart;
                                bit_clear(PIR1,TMR1IF);// clear interrupt flag
                            }
                        } ///
                        LED2 = 0;
                    }
                    // Otherwise write data to socket
                    WriteSeiko(Socket_Data,ch);
                }
                index++;
            }

            // We are done sending, send the buffer of data
            WriteSeiko(TCP_Data_Send,0);
            LED2 = 1;                         // turn LED2 OFF

            // Wait for socket to finish sending data
            count=0;
            bit_clear(PIR1,TMR1IF);// clear interrupt flag
            while(0x40!=(ReadSeiko(0x22) & 0x40))
            {
                if(bit_test(PIR1,TMR1IF))
                {
                    count++;
                    bit_clear(PIR1,TMR1IF);// clear interrupt flag
                }
            }

            // Close Socket
            printf(“close socket\n”);
            LED2 = 1;      // turn LED2 OFF
            WriteSeiko(Socket_Activate,0);// close the socket
            WriteSeiko(TCP_Data_Send,0);

            // Wait for Socket To Close, 2.5 seconds Max, can shorten in most
            //  cases if we’ve waited for the buffer to empty above.
            for(i=0;i<255;i++)
            {
```

```
                delay_ms(10);
                temp = ReadSeiko(Socket_Status_M);
                if((temp & 0x0f)==0x08)// Socket closed
                    break;
                if((temp&0xe0))// Error detected
                    break;
            }
            printf("\n\rfinal socket wait\n");
            while(ReadSeiko(Socket_Status_H));
            delay_ms(5000);   // wait five more seconds to be sure socket is closed
        }
}

// EEPROM addresses:
// username00 - 1F32 bytes
// password20 - 3F32 bytes
// Phone#40 - 4F16 bytes

/***************************************************************************
*  Filename: SEIKO_CT.C
****************************************************************************
*   Author: Stephen Humberd/Rodger Richey
*   Company:Microchip Technology
*   Revision:RevA0
*   Date:   5-24-00
*   Compiled using CCS PICC

****************************************************************************
*
*  This file contains the routines to read and write data to the S-7600A.
*
*  External Clock provided by Si2400 modem = 9.8304MHz
*
***************************************************************************/

/**************************************************************
**  void WriteSeiko(char address, char data)                 **
**  Writes a byte of data to a register on the S-7600A.      **
**************************************************************/
void WriteSeiko(char address, char data)
{
    while(!BUSY);// Wait for S-7600A
    CS = 1;     // 1st cycle sets register address
    RS = 0;
    WRITEX = 0;
    PORTD = address;// Send address
    TRISD = 0;
    READX = 1;
    READX = 0;
    WRITEX = 1;
    RS = 1;
    CS = 0;

    CS = 1;     // 2nd cycle sends data
    WRITEX = 0;
    PORTD = data;// Send data
    READX = 1;
    READX = 0;
    WRITEX = 1;
    CS = 0;

    TRISD = 0xff;
}
```

```
/**************************************************************
**  char ReadSeiko(char address)                             **
**  Reads a byte of data from a register on the S-7600A.     **
**************************************************************/
char ReadSeiko(char address)
{
    char data;

    while(!BUSY);// Wait for S-7600A
    CS = 1;    // 1st cycle sets register address
    RS = 0;
    WRITEX = 0;
    PORTD = address;// Write address
    TRISD = 0;
    READX = 1;
    READX = 0;
    WRITEX = 1;
    RS = 1;
    CS = 0;

    TRISD = 0xff;// 2nd cycle
    CS = 1;
    RS = 0;
    READX = 1;
    data = PORTD;
    READX = 0;
    RS = 1;
    CS = 0;

    while(!BUSY);// Wait for S-7600A
    CS = 1;    // to get data
    READX = 1;
    data = PORTD;// Read data
    READX = 0;
    CS = 0;

    return (data);
}

/***************************************************************************
*  Filename: EEPROM.C
****************************************************************************
*   Author: Stephen Humberd/Rodger Richey
*   Company:Microchip Technology
*   Revision:RevA0
*   Date:   5-24-00
*   Compiled using CCS PICC
****************************************************************************
*
*  This file contains external EEPROM access routines.  Initialization, write,
*  and read routines are provided.
*
*  External Clock provided by Si2400 modem = 9.8304MHz
*
***************************************************************************/

/**************************************************************
**  void init_ext_eeprom(void)                               **
**  Initializes I/O pins for use as Master Mode I2C          **
**************************************************************/
void init_ext_eeprom(void)
{
    output_float(EEPROM_SCL);
    output_float(EEPROM_SDA);
}
```

```
/***********************************************************
**  void write_ext_eeprom(long int address, char data)       **
**  Writes the value in data to the address in address.      **
***********************************************************/
void write_ext_eeprom(long int address, char data)
{              // set for 256Kbyte EEPROM
    char w_tmp;
    i2c_start();// start bit
    i2c_write(0xa0);// control byte
    i2c_write(hi(address));// Address high byte
    i2c_write(address);// Address low byte
    i2c_write(data);// data byte
    i2c_stop();// stop bit
    delay_ms(11);// delay to allow write
}

/***********************************************************
**  void read_ext_eeprom(long int address)                   **
**  Returns the 8-bit value in address.                      **
***********************************************************/
char read_ext_eeprom(long int address)
{              // set for 256Kbyte EEPROM
    char data, r_tmp;

    i2c_start();// start bit
    i2c_write(0xa0);// control byte
    i2c_write(hi(address));// address high byte
    i2c_write(address);// address low byte
    i2c_start();// restart
    i2c_write(0xa1);// control byte
    data=i2c_read(0);// read byte
    i2c_stop();// stop bit
    return(data);
}

/************************************************************************
*  Filename: LCD_CUT.C
*************************************************************************
*   Author: Stephen Humberd/Rodger Richey
*   Company:Microchip Technology
*   Revision:RevA0
*   Date:   5-24-00
*   Compiled using CCS PICC
*************************************************************************
*
*  This file contains the LCD interface routines to send a nibble, byte,
*  initialize the LCD, goto an x,y coordinate and write a byte.
*
*  External Clock provided by Si2400 modem = 9.8304MHz
*
************************************************************************/

/***********************************************************
**  void lcd_send_nibble( byte n )                           **
**  Writes 4-bits of information to the LCD display.         **
***********************************************************/
void lcd_send_nibble( byte n )
{
    lcd.data = n;// Write nibble to port
    delay_cycles(1);// delay
    lcd.enable = 1;// clock data in
    delay_us(2);
    lcd.enable = 0;
}
```

```
/**************************************************************
**  void lcd_send_byte( byte address, byte n )               **
**  Writes the byte n to the address in address.             **
**************************************************************/
void lcd_send_byte( byte address, byte n )
{
    set_tris_b(LCD_WRITE);// set TRIS bits for output
    lcd.reg_sel = 0;
    delay_us(50);
    lcd.reg_sel = address;// select register to write
    delay_cycles(1);
    lcd.rd_w = 0;// set for writes
    delay_cycles(1);
    lcd.enable = 0;
    lcd_send_nibble(n >> 4);// write data byte in nibbles
    lcd_send_nibble(n & 0xf);
    TRISB=0xfd;
}


/**************************************************************
**  void lcd_init(void)                                      **
**  Initializes the LCD display.                             **
**************************************************************/
void lcd_init()
{
    byte i;

    set_tris_b(LCD_WRITE);// set tris bits
    lcd.reg_sel = 0;// select configuration
    lcd.enable = 0;
    delay_ms(15);

    lcd_send_byte(0,0x28);// Write config info
    lcd_send_byte(0,0x0C);
    lcd_send_byte(0,0x01);
    TRISB=0xfd;
}


/**************************************************************
**  void lcd_gotoxy(byte x,byte y)                           **
**  Changes the cursor position to x,y.                      **
**************************************************************/
void lcd_gotoxy( byte x, byte y)
{
    byte address;

    if(y!=1)   // Check for line 1 or 2
        address=lcd_line_two;
    else
        address=0;
    address+=x-1;
    lcd_send_byte(0,0x80|address);// Write cursor position
}

/**************************************************************
**  void lcd_putc(char c)                                    **
**  Writes the byte c to the current cursor position. Routine**
**  detects form feeds, returns, and backspaces.             **
**************************************************************/
void lcd_putc( char c)
{
    switch(c)
    {
        case ‘\f’:     // form feed
```

```
           lcd_send_byte(0,1);
           delay_ms(2);
           break;
       case ‘\n’: // new line
           lcd_gotoxy(1,2);
           break;
       case ‘\b’: // backspace
           lcd_send_byte(0,0x10);
           break;
       default:   // character
           lcd_send_byte(1,c);
       break;
   }
}

/****************************************************************************
*  Filename: F877.H
*****************************************************************************
*   Author: Stephen Humberd/Rodger Richey
*   Company:Microchip Technology
*   Revision:RevA0
*   Date:   5-24-00
*   Compiled using CCS PICC
*****************************************************************************
*
*  This is a header file containing register and bit definitions for the
*  PIC16F877.
*
*  External Clock provided by Si2400 modem = 9.8304MHz
*
****************************************************************************/

//----- Registers --------------------------------------------------------
#byte INDF=0x000
#byte TMR0=0x001
#byte PCL=0x002
#byte STATUS=0x003
#byte FSR=0x004
#byte PORTA=0x005
#byte PORTB=0x006
#byte PORTC=0x007
#byte PORTD=0x008
#byte PORTE=0x009
#byte PCLATH=0x00A
#byte INTCON=0x00B
#byte PIR1=0x00C
#byte PIR2=0x00D
#byte TMR1L=0x00E
#byte TMR1H=0x00F
#byte T1CON=0x010
#byte TMR2=0x011
#byte T2CON=0x012
#byte SSPBUF=0x013
#byte SSPCON=0x014
#byte CCPR1L=0x015
#byte CCPR1H=0x016
#byte CCP1CON=0x017
#byte RCSTA=0x018
#byte TXREG=0x019
#byte RCREG=0x01A
#byte CCPR2L=0x01B
#byte CCPR2H=0x01C
#byte CCP2CON=0x01D
#byte ADRESH=0x01E
#byte ADCON0=0x01F
```

```
#byte OPTION_REG=0x081
#byte TRISA=0x085
#byte TRISB=0x086
#byte TRISC=0x087
#byte TRISD=0x088
#byte TRISE=0x089
#byte PIE1=0x08C
#byte PIE2=0x08D
#byte PCON=0x08E
#byte SSPCON2=0x091
#byte PR2=0x092
#byte SSPADD=0x093
#byte SSPSTAT=0x094
#byte TXSTA=0x098
#byte SPBRG=0x099
#byte ADRESL=0x09E
#byte ADCON1=0x09F

#byte EEDATA=0x10C
#byte EEADR=0x10D
#byte EEDATH=0x10E
#byte EEADRH=0x10F

#byte EECON1=0x18C
#byte EECON2=0x18D

//----- STATUS Bits --------------------------------------------------------

#bit IRP =STATUS.7
#bit RP1 =STATUS.6
#bit RP0 =STATUS.5
#bit NOT_TO = STATUS.4
#bit NOT_PD = STATUS.3
#bit Z =STATUS.2
#bit DC =STATUS.1
#bit C = STATUS.0

//----- INTCON Bits --------------------------------------------------------

#bit GIE =  INTCON.7
#bit PEIE = INTCON.6
#bit T0IE = INTCON.5
#bit INTE = INTCON.4
#bit RBIE = INTCON.3
#bit T0IF = INTCON.2
#bit INTF = INTCON.1
#bit RBIF = INTCON.0

//----- PIR1 Bits ----------------------------------------------------------

#bit PSPIF =PIR1.7
#bit ADIF =PIR1.6
#bit RCIF =PIR1.5
#bit TXIF =PIR1.4
#bit SSPIF =PIR1.3
#bit CCP1IF =PIR1.2
#bit TMR2IF =PIR1.1
#bit TMR1IF =PIR1.0

//----- PIR2 Bits ----------------------------------------------------------

#bit EEIF =PIR2.4
#bit BCLIF =PIR2.3
#bit CCP2IF =PIR2.0
```

```
//----- T1CON Bits ---------------------------------------------------------

#bit T1CKPS1 =T1CON.5
#bit T1CKPS0 =T1CON.4
#bit T1OSCEN =T1CON.3
#bit NOT_T1SYNC =T1CON.2
#bit T1INSYNC =T1CON.2
#bit TMR1CS =T1CON.1
#bit TMR1ON =T1CON.0

//----- T2CON Bits ---------------------------------------------------------

#bit TOUTPS3 =T2CON.6
#bit TOUTPS2 =T2CON.5
#bit TOUTPS1 =T2CON.4
#bit TOUTPS0 =T2CON.3
#bit TMR2ON =T2CON.2
#bit T2CKPS1 =T2CON.1
#bit T2CKPS0 =T2CON.0

//----- SSPCON Bits --------------------------------------------------------

#bit WCOL =SSPCON.7
#bit SSPOV =SSPCON.6
#bit SSPEN =SSPCON.5
#bit CKP =SSPCON.4
#bit SSPM3 =SSPCON.3
#bit SSPM2 =SSPCON.2
#bit SSPM1 =SSPCON.1
#bit SSPM0 =SSPCON.0

//----- CCP1CON Bits -------------------------------------------------------

#bit CCP1X =CCP1CON.5
#bit CCP1Y =CCP1CON.4
#bit CCP1M3 =CCP1CON.3
#bit CCP1M2 =CCP1CON.2
#bit CCP1M1 =CCP1CON.1
#bit CCP1M0 =CCP1CON.0

//----- RCSTA Bits ---------------------------------------------------------

#bit SPEN =RCSTA.7
#bit RX9 =RCSTA.6
#bit SREN =RCSTA.5
#bit CREN =RCSTA.4
#bit ADDEN = RCSTA.3
#bit FERR =RCSTA.2
#bit OERR =RCSTA.1
#bit RX9D =RCSTA.0

//----- CCP2CON Bits -------------------------------------------------------

#bit CCP2X =CCP2CON.5
#bit CCP2Y =CCP2CON.4
#bit CCP2M3 =CCP2CON.3
#bit CCP2M2 =CCP2CON.2
#bit CCP2M1 =CCP2CON.1
#bit CCP2M0 =CCP2CON.0

//----- ADCON0 Bits --------------------------------------------------------

#bit ADCS1 =ADCON0.7
#bit ADCS0 =ADCON0.6
#bit CHS2 =ADCON0.5
#bit CHS1 =ADCON0.4
#bit CHS0 =ADCON0.3
```

```
#bit GO =ADCON0.2
#bit NOT_DONE =ADCON0.2
#bit CHS3 =ADCON0.1
#bit ADON =ADCON0.0

//----- OPTION Bits --------------------------------------------------------

#bit NOT_RBPU =OPTION_REG.7
#bit INTEDG =OPTION_REG.6
#bit T0CS =OPTION_REG.5
#bit T0SE =OPTION_REG.4
#bit PSA =OPTION_REG.3
#bit PS2 =OPTION_REG.2
#bit PS1 =OPTION_REG.1
#bit PS0 =OPTION_REG.0

//----- TRISE Bits ---------------------------------------------------------

#bit IBF = TRISE.7
#bit OBF = TRISE.6
#bit IBOV = TRISE.5
#bit PSPMODE = TRISE.4

//----- PIE1 Bits ----------------------------------------------------------

#bit PSPIE =PIE1.7
#bit ADIE =PIE1.6
#bit RCIE =PIE1.5
#bit TXIE =PIE1.4
#bit SSPIE =PIE1.3
#bit CCP1IE =PIE1.2
#bit TMR2IE =PIE1.1
#bit TMR1IE =PIE1.0

//----- PIE2 Bits ----------------------------------------------------------

#bit EEIE =PIE2.4
#bit BCLIE =PIE2.3
#bit CCP2IE =PIE2.0

//----- PCON Bits ----------------------------------------------------------

#bit NOT_POR = PCON.1
#bit NOT_BOR = PCON.0

//----- SSPCON2 Bits --------------------------------------------------------

#bit GCEN =SSPCON2.7
#bit ACKSTAT =SSPCON2.6
#bit ACKDT =SSPCON2.5
#bit ACKEN =SSPCON2.4
#bit RCEN =SSPCON2.3
#bit PEN =SSPCON2.2
#bit RSEN =SSPCON2.1
#bit SEN =SSPCON2.0

//----- SSPSTAT Bits -------------------------------------------------------

#bit SMP =SSPSTAT.7
#bit CKE =SSPSTAT.6
#bit D_A =SSPSTAT.5
#bit P =SSPSTAT.4
#bit S =SSPSTAT.3
#bit R_W =SSPSTAT.2
#bit UA =SSPSTAT.1
#bit BF =SSPSTAT.0
```

```
//----- TXSTA Bits ---------------------------------------------------------

#bit CSRC =TXSTA.7
#bit TX9 =TXSTA.6
#bit TXEN =TXSTA.5
#bit SYNC =TXSTA.4
#bit BRGH =TXSTA.2
#bit TRMT =TXSTA.1
#bit TX9D =TXSTA.0

//----- ADCON1 Bits --------------------------------------------------------

#bit ADFM =ADCON1.5
#bit PCFG3 =ADCON1.3
#bit PCFG2 =ADCON1.2
#bit PCFG1 =ADCON1.1
#bit PCFG0 =ADCON1.0

//----- EECON1 Bits --------------------------------------------------------

#bit EEPGD = EECON1.7
#bit WRERR = EECON1.3
#bit WREN = EECON1.2
#bit EEWR = EECON1.1
#bit EERD = EECON1.0

/***************************************************************************
*  Filename: S7600.H
****************************************************************************
*   Author:    Stephen Humberd/Rodger Richey
*   Company:   Microchip Technology
*   Revision:  RevA0
*   Date:      5-24-00
*   Compiled using CCS PICC

****************************************************************************
*
*  This file contains the register address definitions of the S-7600A.
*
*  External Clock provided by Si2400 modem = 9.8304 MHz
*
***************************************************************************/

// Seiko S-7600A TCP/IP Stack IC
// Header File

// #case
#define  Revision                0x00
#define  General_Control         0x01
#define  General_Socket_Location 0x02
#define  Master_Interrupt        0x04
#define  Serial_Port_Config      0x08
#define  Serial_Port_Int         0x09
#define  Serial_Port_Int_Mask    0x0a
#define  Serial_Port_Data        0x0b
#define  BAUD_Rate_Div_L         0x0c
#define  BAUD_Rate_Div_H         0x0d
#define  Our_IP_Address_L        0x10
#define  Our_IP_Address_M        0x11
#define  Our_IP_Address_H        0x12
#define  Our_IP_Address_U        0x13
#define  Clock_Div_L             0x1c
#define  Clock_Div_H             0x1d
#define  Socket_Index            0x20
#define  Socket_TOS              0x21
```

```
#define  Socket_Config_Status_L  0x22
#define  Socket_Status_M         0x23
#define  Socket_Activate         0x24
#define  Socket_Interrupt        0x26
#define  Socket_Data_Avail       0x28
#define  Socket_Interrupt_Mask_L 0x2a
#define  Socket_Interrupt_Mask_H 0x2b
#define  Socket_Interrupt_L      0x2c
#define  Socket_Interrupt_H      0x2d
#define  Socket_Data             0x2e
#define  TCP_Data_Send           0x30
#define  Buffer_Out_L            0x30
#define  Buffer_Out_H            0x31
#define  Buffer_In_L             0x32
#define  Buffer_In_H             0x33
#define  Urgent_Data_Pointer_L   0x34
#define  Urgent_Data_Pointer_H   0x35
#define  Their_Port_L            0x36
#define  Their_Port_H            0x37
#define  Our_Port_L              0x38
#define  Our_Port_H              0x39
#define  Socket_Status_H         0x3a
#define  Their_IP_Address_L      0x3c
#define  Their_IP_Address_M      0x3d
#define  Their_IP_Address_H      0x3e
#define  Their_IP_Address_U      0x3f
#define  PPP_Control_Status      0x60
#define  PPP_Interrupt_Code      0x61
#define  PPP_Max_Retry           0x62
#define  PAP_String              0x64
```

APPENDIX D: WEB CLIENT SCHEMATICS

FIGURE D-1: TCP/IP CLIENT – SODA MACHINE (SHEET 1 OF 3)

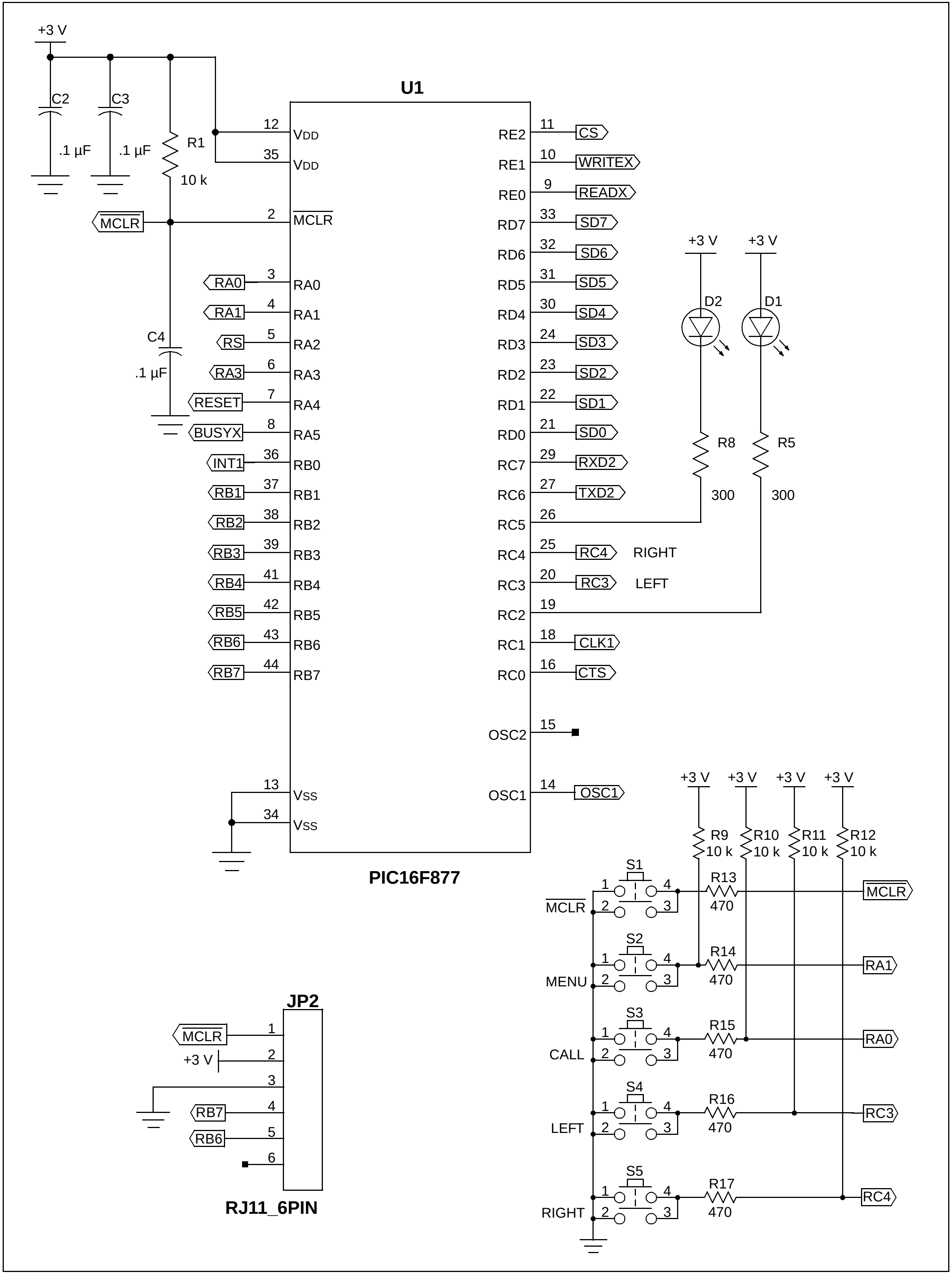


FIGURE D-2: TCP/IP CLIENT – SODA MACHINE (SHEET 2 OF 3)

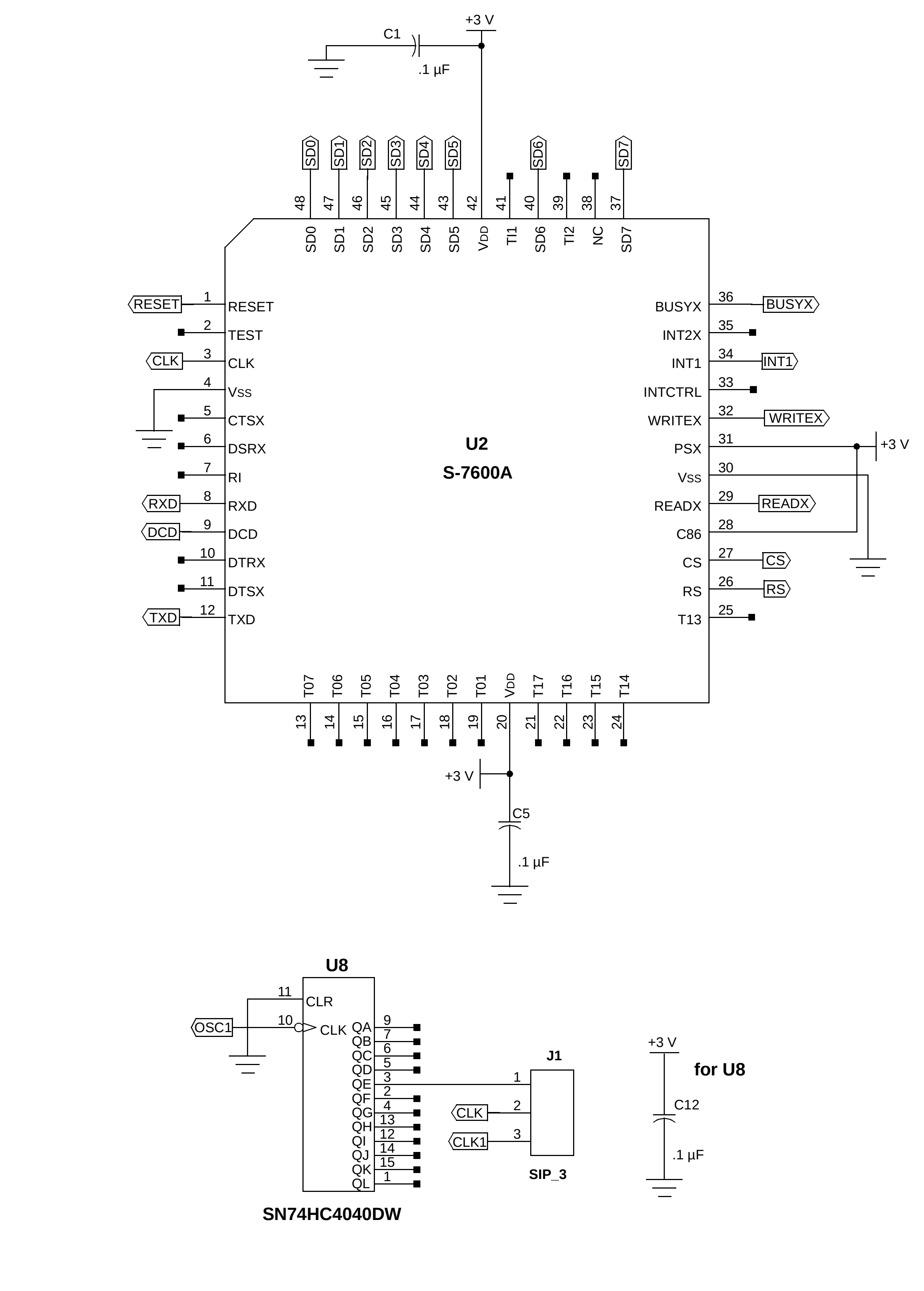


FIGURE D-3: TCP/IP CLIENT – SODA MACHINE (SHEET 3 OF 3)

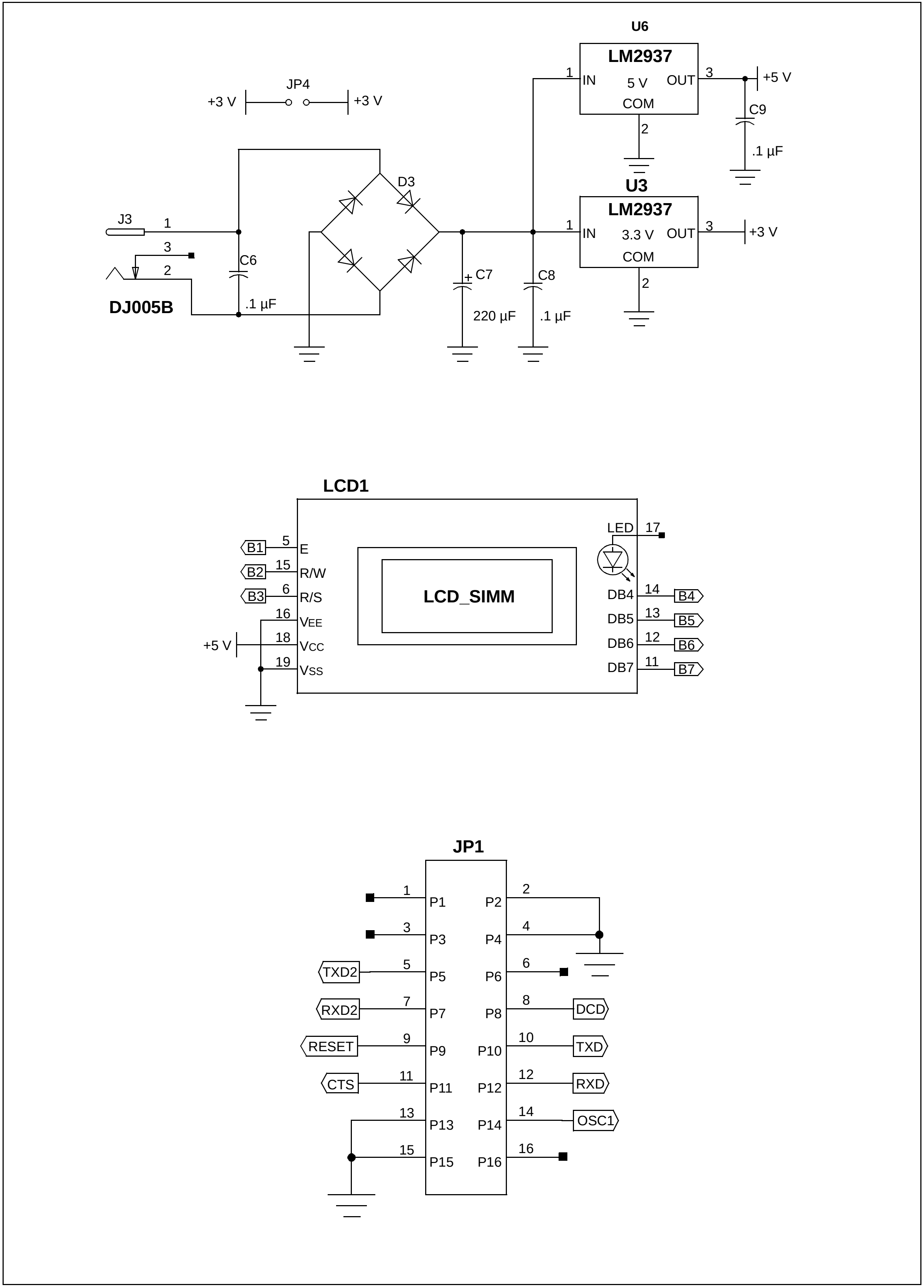


FIGURE D-4: WEB CLIENT FLOWCHART (SHEET 1 OF 3)

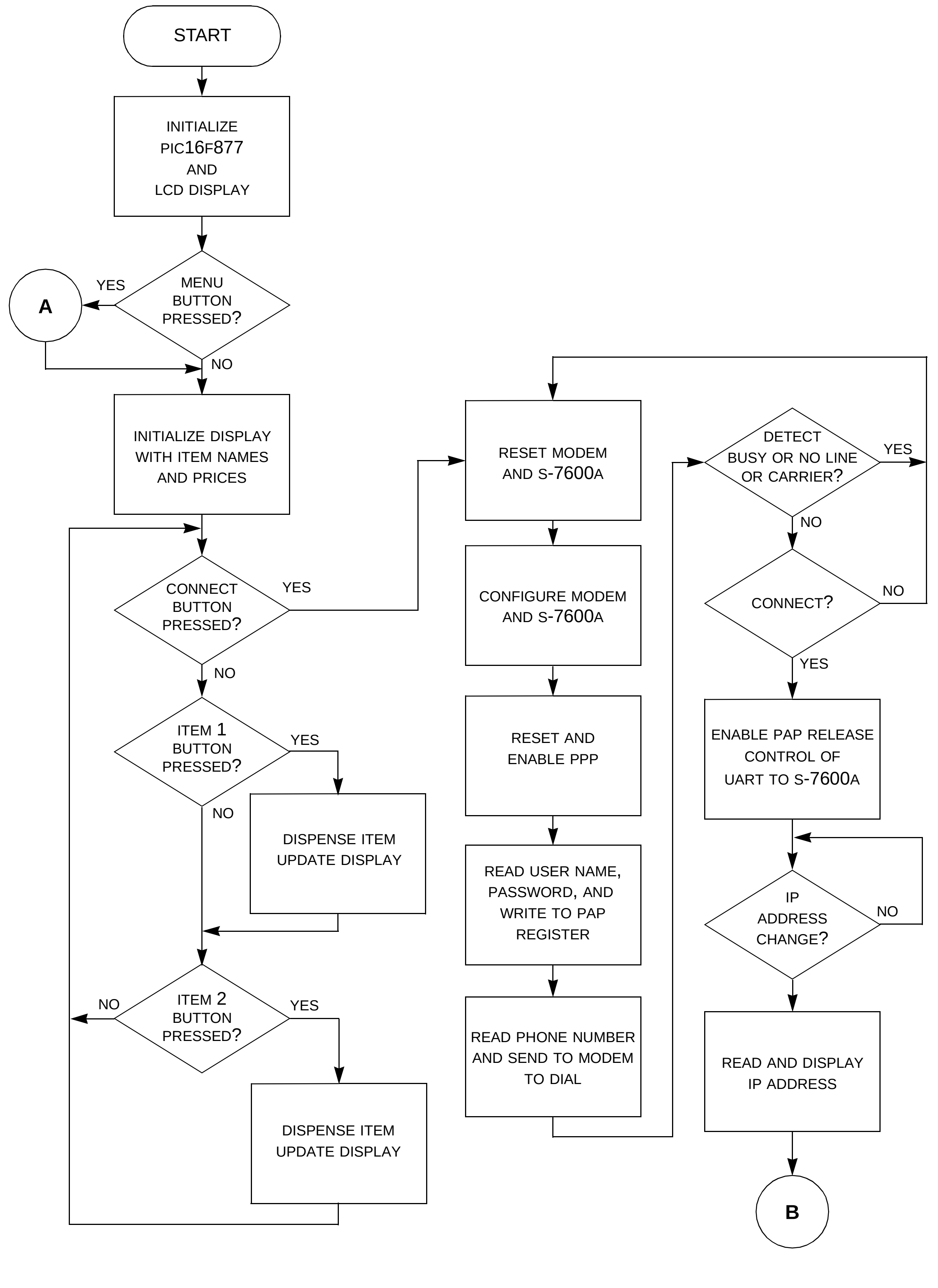

START
INITIALIZE PIC16F877 AND LCD DISPLAY
MENU BUTTON PRESSED?
YES
A
NO
INITIALIZE DISPLAY WITH ITEM NAMES AND PRICES
CONNECT BUTTON PRESSED?
YES
NO
ITEM 1 BUTTON PRESSED?
YES
NO
DISPENSE ITEM UPDATE DISPLAY
ITEM 2 BUTTON PRESSED?
NO
YES
DISPENSE ITEM UPDATE DISPLAY
RESET MODEM AND S-7600A
CONFIGURE MODEM AND S-7600A
RESET AND ENABLE PPP
READ USER NAME, PASSWORD, AND WRITE TO PAP REGISTER
READ PHONE NUMBER AND SEND TO MODEM TO DIAL
DETECT BUSY OR NO LINE OR CARRIER?
YES
NO
CONNECT?
NO
YES
ENABLE PAP RELEASE CONTROL OF UART TO S-7600A
IP ADDRESS CHANGE?
NO
READ AND DISPLAY IP ADDRESS
B

FIGURE D-5: WEB CLIENT FLOWCHART (SHEET 2 OF 3)

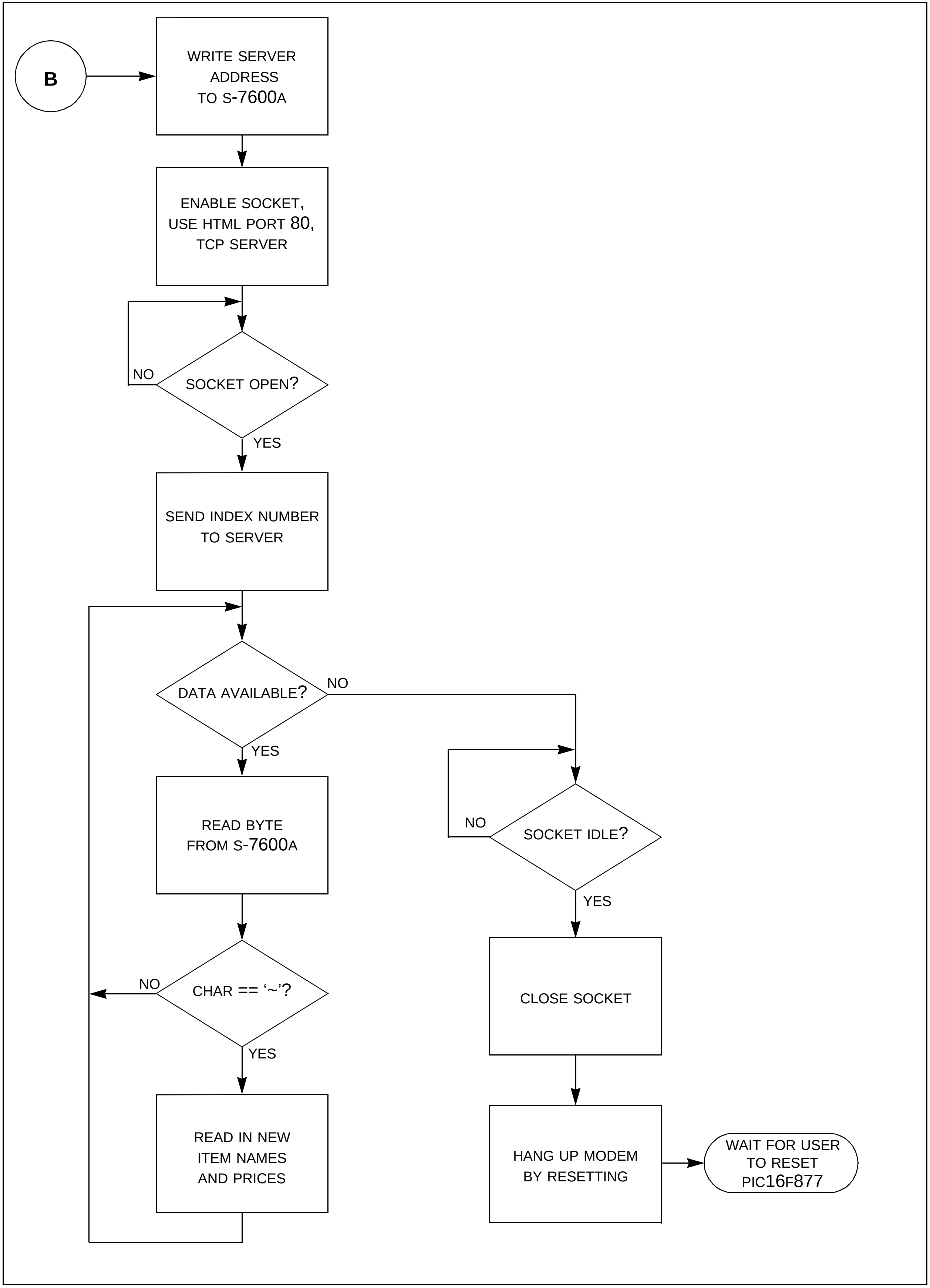
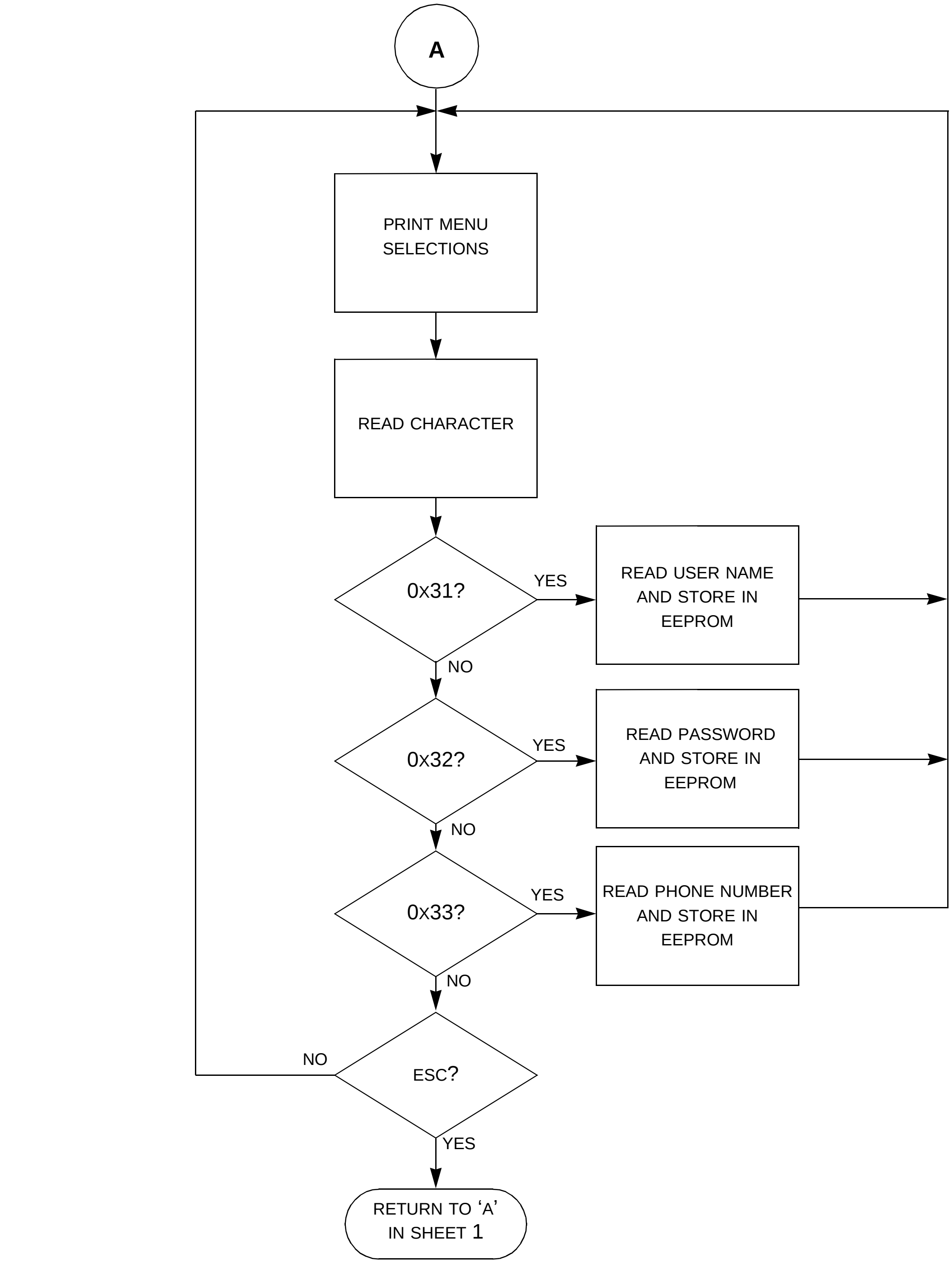

B
WRITE SERVER ADDRESS TO S-7600A
ENABLE SOCKET, USE HTML PORT 80, TCP SERVER
NO
SOCKET OPEN?
YES
SEND INDEX NUMBER TO SERVER
DATA AVAILABLE?
NO
YES
READ BYTE FROM S-7600A
NO
CHAR == '~'?
YES
READ IN NEW ITEM NAMES AND PRICES
NO
SOCKET IDLE?
YES
CLOSE SOCKET
HANG UP MODEM BY RESETTING
WAIT FOR USER TO RESET PIC16F877

FIGURE D-6: WEB CLIENT FLOWCHART (SHEET 1 OF 3)


A
PRINT MENU SELECTIONS
READ CHARACTER
0x31?
YES
NO
READ USER NAME AND STORE IN EEPROM
0x32?
YES
NO
READ PASSWORD AND STORE IN EEPROM
0x33?
YES
NO
READ PHONE NUMBER AND STORE IN EEPROM
ESC?
NO
YES
RETURN TO 'A' IN SHEET 1

APPENDIX E: WEB CLIENT SOURCE CODE

```
/***************************************************************************
*  Filename: SODA_TCP.C
****************************************************************************
*   Author:    Stephen Humberd/Rodger Richey
*   Company:   Microchip Technology
*   Revision:  RevA0
*   Date:      5-30-00
*   Compiled using CCS PICC
****************************************************************************
*   Include files:
*       16f877.h
*       f877.h
*       s7600.h
*       ctype.h
*       string.h
*       lcd_cut.c
*       seiko_ct.c
*       int_ee.c
*       get_info.c
****************************************************************************
*
*  This program demonstrates a simple WEB client using a Microchip PIC16F877
*  microcontroller and a Seiko S-7600 TCP/IP Network Protocol chip.  This
*  is the main file that includes all other header files and source code files.
*
*  External Clock provided by Si2400 modem = 9.8304 MHz
*
****************************************************************************
*  Internal EEPROM Addresses
*    User Name        0x00 to 0x1F
*    Password         0x20 to 0x3F
*    Phone Number     0x40 to 0x4F
*    Item_1           0x50 to 0x58
*    Item_2           0x60 to 0x68
*    Price_1          0x70 to 0x74
*    Price_2          0x78 to 0x7C
*
*  Edit Line 52 to change Default Quantity
*  Edit Line 53-54 to change Port #
*  Edit Line 55-58 to change Their IP Address
***************************************************************************/

#include “f877.h”
#include <16f877.h>
#include “s7600.h”
#include <ctype.h>
#include <string.h>

#fuses HS,NOWDT,NOPROTECT,NOBROWNOUT
```

```
#use delay(clock=9828606)
#use rs232(baud=2400, xmit=PIN_C6, rcv=PIN_C7)

#define esc 0x1b                         // ESC char (for VT-100 emulation)

// PORTA bits
#bit RS     = PORTA.2
#bit RESET  = PORTA.4
#bit BUSY   = PORTA.5

// PORTB bits
//#bit INT1 = PORTB.0
// As defined in the following structure the pin connection is as follows:
//     B1  enable
//     B3  reg_sel
//     B2  rd_w
//     B4  D4
//     B5  D5
//     B6  D6
//     B7  D7

#define QUAN 5                           // Default quantity of Items in Vending Machine
#define PORT_H 0x1F                      // IP Port #
#define PORT_L 0x90
#define T_Address_163                    // Their IP Address
#define T_Address_2 224
#define T_Address_3 145
#define T_Address_4 51
#define lcd_type 2                       // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40                // LCD RAM address for the second line

char Flags;                              // Flag register
#bit CONNECTED = Flags.0                 // Established TCP/IP connection
#bit INTRUSION = Flags.1                 // Parallel phone detect
#bit WRITEREADING = Flags.2              // Write count to serial EEPROM
#bit SECONDS = Flags.3                   // 60 second flag
#bit UPLOAD = Flags.4                    // Make TCP/IP connection and upload
#bit ONHOOK = Flags.5                    // Parallel Phone Detect
#bit WRITETS = Flags.6
#bit ATTEMPT = Flags.7

byte lcd_cmd, cmd_tmp;
byte CONST LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6};
                                         // These bytes need to be sent to the LCD
                                         // to start it up.

// PORTC bits
#bit CTS    = PORTC.0// When CTS = 0 send is enabled
#bit LED1   = PORTC.2// When LED = 0 the LED is ON
#bit LED2   = PORTC.5// When LED = 0 the LED is ON

// PORTE bits
#bit READX  = PORTE.0
#bit WRITEX = PORTE.1
#bit CS     = PORTE.2

// Buttons
// #bit Call= PORTA.0
// #bit T_Menu= PORTA.1
// #bit Left= PORTC.3
// #bit Right= PORTC.4

char i,j,ch,addr,temp,count,count1,page;
long index, byt_cnt;
char MyIPAddr[4];
char TheirIPAddr[4];
char user[33];
char pass[33];
char phone[17];

// vending machine variables
char Item_1[9];
char Item_2[9];
char Price_1[5], Price_2[5];
int  Quan_1, Quan_2, Item_Num;

struct lcd_pin_map {                     // This structure is overlayed
           boolean unused;               // on to an I/O port to gain
           boolean enable;               // access to the LCD pins.
           boolean rd_w;                 // The bits are allocated from
           boolean reg_sel;              // low order up.  UNUSED will
           int   data : 4;               // be pin B0.
        } lcd;

#byte lcd = 6                            // This puts the entire structure
                                         // on to port B (at address 6)

STRUCT lcd_pin_map const LCD_WRITE = {1,0,0,0,0};// For write mode all pins are
                                                 // out except RB0 (int1)

#include “lcd_cut.c”                     // LCD routines
#include “seiko_ct.c”                    // Seiko routines
#include “Int_EE.c”                      // Internal EEPROM
#include “get_info.c”                    // Get Username, Password, and Phone Number routines

/***************************************************************
**  void Write_LCD_1(void)                                    **
**  Initializes text on LCD display.                          **
***************************************************************/
void Write_LCD_1(void)
{
    lcd_putc(“\f”);                      // LCD line 1

    i = 0;
    while(Item_1[i])                     // Write text of Item 1 to LCD
    {
        lcd_putc(Item_1[i]);
        i++;
    }

    lcd_gotoxy(9,1);                     // LCD Line 1 pos 9
    i = 0;
    while(Item_2[i])                     // Write text of Item 2 to LCD
    {
        lcd_putc(Item_2[i]);
        i++;
    }
    lcd_gotoxy(3,2);                     // LCD Line 2 pos 3
    i = strlen(Price_2);

    i = 0;
    while(Price_1[i])                    // Write price of Item 1 to LCD
    {
        lcd_putc(Price_1[i]);
        i++;
    }
    lcd_gotoxy(11,2);                    // LCD Line 2 pos 11
    i = 0;
    while(Price_2[i])                    // Write price of Item 2 to LCD
    {
        lcd_putc(Price_2[i]);
        i++;
    }
}

/**************************************************************
**  void disp_left(void)                                     **
**  Dispenses from Item 1 on left side of LCD.  Shows item   **
**  being dispensed and shows quantities.                    **
**************************************************************/
void disp_left(void)
{
    Write_LCD_1();                       // Display Item names on LCD
    lcd_putc(“\n  “);
    lcd_putc(0x00);
    lcd_putc(0x00);
    lcd_putc(0x00);
    lcd_putc(“  KerPlunk”);              // Show item dispensed
    if(Quan_1)                           // If still have Item 1
    {
        Quan_1--;                        // decrement quantity
        if(Quan_1 == 0)                  // Show out if quantity = 0
            strcpy(Price_1, “Out”);
    }
    delay_ms(1500);                      // short delay
    Write_LCD_1();                       // Display Item names on LCD
}

/**************************************************************
**  void disp_right(void)                                    **
**  Dispenses from Item 2 on right side of LCD.  Shows item  **
**  being dispensed and shows quantities.                    **
**************************************************************/
void disp_right(void)
{
    Write_LCD_1();                       // Display Item names on LCD
    lcd_putc(“\n KerPlunk  “);           // Show item dispensed
    lcd_putc(0x00);
    lcd_putc(0x00);
    lcd_putc(0x00);
    if(Quan_2)                           // If still have Item 2
    {
        Quan_2--;                        // decrement quantity
        if(Quan_2 == 0)                  // Show out if quantity = 0
            strcpy(Price_2, “Out”);
    }
    delay_ms(1500);                      // short delay
    Write_LCD_1();                       // Display Item names on LCD
}

/**************************************************************
**  void Make_arrow(void)                                    **
**  Creates a down arrow in User RAM on the LCD module.      **
**************************************************************/
void Make_arrow(void)
{
    set_tris_b(LCD_WRITE);               // Configure TRIS bits
    lcd.reg_sel = 0;                     // Select command registers
    lcd.enable = 0;
    delay_ms(15);

    lcd_send_byte(0,0x40);               // send start address
    lcd.reg_sel = 1;                     // Select data registers
    lcd_putc(4);                         // Write arrow data
    lcd_putc(4);
    lcd_putc(4);
    lcd_putc(4);
    lcd_putc(21);
    lcd_putc(14);
    lcd_putc(4);
    lcd.reg_sel = 0;                        // Select command registers
    lcd_send_byte(0,0x80);
}

/***************************************************************
**  void main(void)                                           **
***************************************************************/
void main(void)
{
    // Initialize PORTs & TRISs
    PORTA = 0x24;                           //00100100
    PORTB = 0xff;                           //00000000
    PORTD = 0;                              //00000000
    PORTE = 0xFA;                           //11111010

    TRISA = 0xE3;                           //11100011
    TRISB = 0xff;
    TRISC = 0x98;                           //10011000
    TRISD = 0xff;                           //11111111
    TRISE = 0;                              //00000000

    ADCON1 = 0x06;                          //00000110 all digital to start
    ADCON0 = 0;

    PR2 = 0x07;                             // Enable & use PWM
    CCPR2L = 0x03;
    CCP2CON = 0x0C;                         //00001100
    T2CON = 0x04;                           //00000100
    T1CON = 0x31;                           //00110001 Timer1

    LED1 = 1;                               // LED OFF
    LED2 = 1;
    CTS = 1;                                // disable send
    Quan_1 = QUAN;
    Quan_2 = QUAN;

    lcd_init();                             // initalize the LCD
    Make_arrow();                           // Create a down arrow in LCD RAM

    ch = bit_test(PORTA,1);                 // test for menu button
    if(ch == 0)
        Menu();                             // Show enter menu

//  Write_data_to_rs232();                  // This routine dumps all data stored in the EEPROM
                                            // to the PC terminal it is used for debug only

// Main program starts here.  Waits for user input which is dispense one of the
// two items and update quantities or connect to server and get new information.
Main_loop:
    LED1 = 1;                               // LED OFF
    LED2 = 1;
    RESET = 0;                              // Reset modem and S-7600A
    delay_ms(1);
    RESET = 1;

    Read_data();                        // Get data from Internal EEPROM and store it in variables
    Write_LCD_1();                      // Display Item names on LCD

    // Wait for user input
    while(1)
    {
        ch = bit_test(PORTA,0);             // rest for call button
        if(ch == 0)                         // Call button pressed
            break;
        if(Quan_1)                        // If still have Item 1
        {
            ch = bit_test(PORTC,3);       // Left button pressed
            if(ch == 0)
                disp_left();              // Dispense Item 1
        }

        if(Quan_2)                        // If still have Item 2
        {
            ch = bit_test(PORTC,4);       // Right buttonpressed
            if(ch == 0)
                disp_right();             // Dispense Item 2
        }
    }

// This part of the program makes the Internet connection to the server
// and retrieves the new information to store in the EEPROM
restart:
    LED1 = 1;                             // LED OFF
    LED2 = 1;
    RESET = 0;                            // Reset modem and S-7600A
    delay_ms(1);
    RESET = 1;

    lcd_putc(“\fReseting Modem\n”);
    printf(“\n\rReseting Modem”);
    delay_ms(10000);

    // Set clock divider for 1KHz clock
    WriteSeiko(Clock_Div_L,0x32);
    while(ReadSeiko(Clock_Div_L) != 0x32) // Detects when reset
        WriteSeiko(Clock_Div_L,0x32);     // is complete
    WriteSeiko(Clock_Div_H,0x01);

    // Set Baud Rate Divisor for 2400 baud @ 9.828606
    WriteSeiko(BAUD_Rate_Div_L,0x7e);
    WriteSeiko(BAUD_Rate_Div_H,0x00);

    // Set up MODEM
    printf(S_Putc,”ATS07=06SE2=C0\r”);    // Send V22bis to modem
    delay_ms(10);
    printf(S_Putc,”ATE0\r”);              // Send Local echo off
    delay_ms(10);

    WriteSeiko(PPP_Control_Status,0x01);  // reset PPP
    WriteSeiko(PPP_Control_Status,0x00);
    WriteSeiko(PPP_Control_Status,0x20);  // set PAP mode
    delay_ms(5);

    // Determine length of “Username” and write it to the PAP register
    // in the S-7600A
    ReadIntEE(0, user, 32);               // read 32 byte password from EEPROM address 0x00
    i = strlen(user);                     // Find actual string length
    WriteSeiko(PAP_String,i);             // Write string length to PAP register

    // Write “Username” to PAP register in S-7600A
    for(j=0; j < i; j++)
        WriteSeiko(PAP_String,user[j]);

    //Determine length of “Password” and write it to the PAP register
    ReadIntEE(0x20, pass, 32);                // read 32 byte username from EEPROM address 0x20
    i = strlen(pass);                         // Find actual string length
    WriteSeiko(PAP_String,i);                 // Write string length to PAP register

    // Write “Password” to PAP register in S-7600A
```

```
for(j=0; j < i; j++)
    WriteSeiko(PAP_String,pass[j]);

WriteSeiko(PAP_String,0x00);             // Write final NULL character to PAP register


// Read Phone Number from EEPROM
ReadIntEE(0x40, phone, 16);              // read 16 byte Phone Number from EEPROM address 0x40

// Send Phone Number to the MODEM
printf(S_Putc,”ATDT”);                   // Send initial ATDT to indicate tone dial
i = strlen(phone);
for(j=0; j < i; j++)                     // Write phone number to modem
    S_Putc(phone[j]);
printf(S_Putc,”\r”);                     // end with CR
delay_ms(5);

// Also Write Phone Number to LCD
lcd_putc(“\fCall\n”);                    // Calling
i = 0;
while(phone[i])                          // Write phone number
{
    lcd_putc(phone[i]);
    i++;
}

// Also Write Phone Number to CRT
printf(“%c[2J”,esc);                     // clear VT100 screen
printf(“\rDialing “);                    // Write dialing
printf(phone);                           // Write phone number
printf(“\r”);
LED1 = 0;                                // LED ON

// Read status bytes from modem while dialing
while(1)
{
    i = ReadSeiko(Serial_Port_Config);   // Read UART status

    lcd_send_byte(0,0x87);               // goto first line char 8
    if(bit_test(i,7))                    // If data available
        ch = ReadSeiko(Serial_Port_Data); // read
    if(ch == ‘t’)                        // dial tone detected
    {
        lcd_putc(“dial tone\n”);
        printf(“\ndial tone\r”);
    }
    if(ch == ‘r’)                        // ring detected
    {
        lcd_putc(“ring     “);
        printf(“\nring\r”);
    }
    if(ch == ‘c’)                        // modem connected
        break;                           // done
    if(ch == ‘b’)                        // busy tone detected
    {
        lcd_putc(“busy     “);           // reset & start over
        printf(“\nbusy\r”);
        gotorestart;
    }
    if(ch == ‘l’)                        // No phone line detected
    {
        lcd_putc(“\fNo Phone Line”);
        printf(“\nNo Phone Line\r”);
        lcd_putc(“\nPRESS MCLR”);
        printf(“\nPRESS MCLR\r”);
        LED1 = 1;// LED OFF
        while(1);
    }
    if(ch == ‘N’)                        // No carrier detected
        gotorestart;
    ch = 0;
}

lcd_putc(“\fConnect  “);
printf(“\nConnect\r”);
LED2 = 0;                                // turn connected LED2 ON

WriteSeiko(PPP_Control_Status,0x62);     // Use PAP, enable PPP
WriteSeiko(Serial_Port_Config,0x01);     // turn serial port over to S-7600
delay_ms(5);

// (Register 0x60) wait for PPP to be Up
while(!(ReadSeiko(PPP_Control_Status)&0x01))
    delay_ms(5);

while(ReadSeiko(Our_IP_Address_L) == 0); // detect when ready to proceed

MyIPAddr[0] = ReadSeiko(Our_IP_Address_L);// Read our assigned IP address
MyIPAddr[1] = ReadSeiko(Our_IP_Address_M);
MyIPAddr[2] = ReadSeiko(Our_IP_Address_H);
MyIPAddr[3] = ReadSeiko(Our_IP_Address_U);


// Print address to the terminal and LCD
printf(“\r\nMy address is %u.%u.%u.%u”,MyIPAddr[3],MyIPAddr[2],MyIPAddr[1],MyIPAddr[0]);
lcd_putc(“\fMy address is\n”);
printf(lcd_putc,”%u.%u.%u.%u”,MyIPAddr[3],MyIPAddr[2],MyIPAddr[1],MyIPAddr[0]);

// Write the preconfigured server IP address to the S-7600A
WriteSeiko(T_Address_1,TheirIPAddr[0]);
WriteSeiko(T_Address_2,TheirIPAddr[1]);
WriteSeiko(T_Address_3,TheirIPAddr[2]);
WriteSeiko(t_Address_4,TheirIPAddr[3]);

for(temp=0;temp<3;temp++)
{
    WriteSeiko(Socket_Index,0x00);           // Use Socket 0
    WriteSeiko(Socket_Config_Status_L,0x10); // Reset socket
    delay_ms(5);
    WriteSeiko(Our_Port_L,PORT_L);           // Write the port address
    WriteSeiko(Their_Port_L,PORT_L);         // for both the server
    WriteSeiko(Our_Port_H,PORT_H);           // and the client
    WriteSeiko(Their_Port_H,PORT_H);

    WriteSeiko(Socket_Config_Status_L,0x02); // Use TCP client mode
    WriteSeiko(Socket_Activate,0x01);        // Activate socket

    // Loop to wait for socket to be connected with the server
    while(1)
    {
        delay_ms(5);
        j = ReadSeiko(Socket_Status_M);

        if(j&0x10)
        {
            i = 0;
            break;
        }
        else if(j&0xe0)
            break;
    }
}
```

```
CONNECTED = 1;

// The client must respond with the current index into the array to get
// the next set of item names and prices.  Use “ ~~ Item_Number ~~ “ where
// Item_Number is between 0 and 9.
Item_Num++;                               // Increment item number
if(Item_Num > 9)                          // Check for overflow
    Item_Num = 0;
WriteSeiko(Socket_Data,’~’);              // Write the string to the S-7600A
WriteSeiko(Socket_Data,’~’);
WriteSeiko(Socket_Data,Item_Num);
WriteSeiko(Socket_Data,’~’);
WriteSeiko(Socket_Data,’~’);
WriteSeiko(Socket_Data,0X0A);
WriteSeiko(Socket_Data,0X0D);

WriteSeiko(TCP_Data_Send,0);              // send data
LED2 = 1;                                 // turn LED2 OFF

// Server should respond with data that has the following sequence
// ~<Item1>;<Item2>;<Price1>;<Price2>
// These are strings that have names and prices.
lcd_putc(“\fReading Data”);
while(ReadSeiko(Socket_Config_Status_L)&0x10)
{
    temp = ReadSeiko(Socket_Data);        // Read characters until a ~
    putc(temp);                           // is read to indicate start
    if(temp == ‘~’)                       // of data
    {
        temp = ReadSeiko(Socket_Data);
        putc(temp);

        i = 0;
        while(temp != ‘;’)                // Read until a ; is reached
        {                                 // to delimit Item1 from Item2
            temp = ReadSeiko(Socket_Data);
            putc(temp);
            Item_1[i] = temp;
            i++;
            Item_1[i] = 0;
        }

        i = 0;
        while(temp != ‘;’) // Read until a ; is reached
        {                                 // to delimit Item2 from Price1
            temp = ReadSeiko(Socket_Data);
            putc(temp);
            Item_2[i] = temp;
            i++;
            Item_2[i] = 0;
        }

        i = 0;
        while(temp != ‘;’) // Read until a ; is reached
        {                                 // to delimit Price1 from Price2
            temp = ReadSeiko(Socket_Data);
            putc(temp);
            Price_1[i] = temp;
            i++;
            Price_1[i] = 0;
        }

        i = 0;
        while(temp != ‘;’) // Read until a ; is reached
        {                                 // to read all of Price2
            temp = ReadSeiko(Socket_Data);
```

```
                putc(temp);
                Price_2[i] = temp;
                i++;
                Price_2[i] = 0;
            }
        }
    }


    // Ensure that socket is inactive
    count=0;
    bit_clear(PIR1,TMR1IF);// clear interrupt flag
    while(0x40!=(ReadSeiko(0x22) & 0x40))
    {
        if(bit_test(PIR1,TMR1IF))
        {
            count++;
            bit_clear(PIR1,TMR1IF);// clear interrupt flag
        }
    }

    // Close Socket
    printf(“close socket\n”);
    LED2 = 1;                 // turn LED2 OFF
    WriteSeiko(Socket_Activate,0);// close the socket
    WriteSeiko(TCP_Data_Send,0);

    // Wait for Socket To Close, 2.5 seconds Max, can shorten in most
    //cases if we’ve waited for the buffer to empty above.
    for(i=0;i<255;i++)
    {
        delay_ms(10);
        temp = ReadSeiko(Socket_Status_M);
        if((temp & 0x0f)==0x08)// Socket closed
            break;
        if((temp&0xe0))    // Error detected
            break;
    }
    printf(“\n\rfinal socket wait\n”);
    LED2 = 1;// LED OFF
    while(ReadSeiko(Socket_Status_H));
    delay_ms(5000);   // wait five more seconds to be sure socket is closed

    // Hang Up Modem by reseting everything
    RESET = 0;
    LED1 = 1;// LED OFF
    delay_ms(1);
    RESET = 1;
    lcd_putc(“\fUpdate Complete\n”);

    // At this point the program freezes until the user removes power.
    // The new downloaded values are saved in EEPROM and when power is
    // applied, these new values will be used for item names and prices.
    lcd_putc(“Power Down”);
    while(1);
}
```

```
/***************************************************************************
*  Filename: SEIKO_CT.C
****************************************************************************
*   Author: Stephen Humberd/Rodger Richey
*   Company:Microchip Technology
*   Revision:RevA0
*   Date:  5-31-00
*   Compiled using CCS PICC

****************************************************************************
*
*  This file contains the routines to read and write data to the S-7600A.
*
***************************************************************************/

/**************************************************************
**  void WriteSeiko(char address, char data)                **
**  Writes a byte of data to a register on the S-7600A.      **
*************************************************************/
void WriteSeiko(char address, char data)
{
    while(!BUSY);// Wait for S-7600A
    CS = 1;    // 1st cycle sets register address
    RS = 0;
    WRITEX = 0;
    PORTD = address;// Send address
    TRISD = 0;
    READX = 1;
    READX = 0;
    WRITEX = 1;
    RS = 1;
    CS = 0;

    CS = 1;    // 2nd cycle sends data
    WRITEX = 0;
    PORTD = data;// Send data
    READX = 1;
    READX = 0;
    WRITEX = 1;
    CS = 0;

    TRISD = 0xff;
}


/**************************************************************
**  char ReadSeiko(char address)                            **
**  Reads a byte of data from a register on the S-7600A.     **
*************************************************************/
char ReadSeiko(char address)
{
    char data;

    while(!BUSY);// Wait for S-7600A
    CS = 1;    // 1st cycle sets register address
    RS = 0;
    WRITEX = 0;
    PORTD = address;// Write address
    TRISD = 0;
    READX = 1;
    READX = 0;
    WRITEX = 1;
    RS = 1;
    CS = 0;
    TRISD = 0xff;// 2nd cycle
    CS = 1;
    RS = 0;
    READX = 1;
    data = PORTD;
    READX = 0;
    RS = 1;
    CS = 0;

    while(!BUSY);// Wait for S-7600A
    CS = 1;    // to get data
    READX = 1;
    data = PORTD;// Read data
    READX = 0;
    CS = 0;

    return (data);
}

/***************************************************************
**  char DataAvailable(void)                                     **
**  Determines if there is any data available to read out of **
**  the S-7600A.                                              **
**  Returns the value of the data available bit from the      **
**  S-7600A.                                                  **
***************************************************************/
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);// Check if S-7600A is busy
    CS = 1;        // 1st cycle sets register
    RS = 0;        // address
    WRITEX = 0;
    PORTD = Serial_Port_Data;// Write to serial port
    TRISD = 0;
    READX = 1;
    READX = 0;
    WRITEX = 1;
    RS = 1;
    CS = 0;

    CS = 1;        // 2nd cycle writes the data
    WRITEX = 0;// to the register
    PORTD = data;// Data to write
    READX = 1;
    READX = 0;
    WRITEX = 1;
    CS = 0;

    TRISD = 0xff;
}

/***************************************************************
**  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(0x22)&0x20))
    {
        WriteSeiko(TCP_Data_Send,0);// If full send data
        while(ReadSeiko(Socket_Status_H));// Wait until done
    }

    while(!BUSY);// Check if S-7600A is busy
    CS = 1;        // 1st cycle sets register
    RS = 0;        // address
    WRITEX = 0;
    PORTD = Socket_Data;// Write to socket
    TRISD = 0;
    READX = 1;
    READX = 0;
    WRITEX = 1;
    RS = 1;
    CS = 0;

    CS = 1;        // 2nd writes the data to
    WRITEX = 0;// the register
    PORTD = data;// Data to write
    READX = 1;
    READX = 0;
    WRITEX = 1;
    CS = 0;

    TRISD = 0xff;
}

/***************************************************************************
*  Filename: LCD_CUT.C
****************************************************************************
*   Author:    Stephen Humberd/Rodger Richey
*   Company:   Microchip Technology
*   Revision:  RevA0
*   Date:      5-31-00
*   Compiled using CCS PICC
****************************************************************************
*
*  This file contains the LCD interface routines to send a nibble, byte,
*  initialize the LCD, goto an x,y coordinate and write a byte.
*
*  External Clock provided by Si2400 modem = 9.8304MHz
*
***************************************************************************/


/**************************************************************
**  void lcd_send_nibble( byte n )                           **
**  Writes 4-bits of information to the LCD display.         **
**************************************************************/
void lcd_send_nibble( byte n )
{
    lcd.data = n;// Write nibble to port
    delay_cycles(1);// delay
    lcd.enable = 1;// clock data in
    delay_us(2);
    lcd.enable = 0;
}


/**************************************************************
**  void lcd_send_byte( byte address, byte n )               **
**  Writes the byte n to the address in address.             **
**************************************************************/
```

```
void lcd_send_byte( byte address, byte n )
{
    set_tris_b(LCD_WRITE);// set TRIS bits for output
    lcd.reg_sel = 0;
    delay_us(50);
    lcd.reg_sel = address;// select register to write
    delay_cycles(1);
    lcd.rd_w = 0;// set for writes
    delay_cycles(1);
    lcd.enable = 0;
    lcd_send_nibble(n >> 4);// write data byte in nibbles
    lcd_send_nibble(n & 0xf);
    TRISB=0xfd;
}


/**************************************************************
**  void lcd_init(void)                                      **
**  Initializes the LCD display.                             **
**************************************************************/
void lcd_init()
{
    byte i;

    set_tris_b(LCD_WRITE);// set tris bits
    lcd.reg_sel = 0;// select configuration
    lcd.enable = 0;
    delay_ms(15);

    lcd_send_byte(0,0x28);// Write config info
    lcd_send_byte(0,0x0C);
    lcd_send_byte(0,0x01);
    TRISB=0xfd;
}


/**************************************************************
**  void lcd_gotoxy(byte x,byte y)                           **
**  Changes the cursor position to x,y.                      **
**************************************************************/
void lcd_gotoxy( byte x, byte y)
{
    byte address;

    if(y!=1)   // Check for line 1 or 2
        address=lcd_line_two;
    else
        address=0;
    address+=x-1;
    lcd_send_byte(0,0x80|address);// Write cursor position
}

/**************************************************************
**  void lcd_putc(char c)                                    **
**  Writes the byte c to the current cursor position. Routine**
**  detects form feeds, returns, and backspaces.             **
**************************************************************/
void lcd_putc( char c)
{
    switch(c)
    {
        case ‘\f’: // form feed
            lcd_send_byte(0,1);
            delay_ms(2);
            break;
        case ‘\n’: // new line
```

```
            lcd_gotoxy(1,2);
            break;
        case ‘\b’: // backspace
            lcd_send_byte(0,0x10);
            break;
        default:   // character
            lcd_send_byte(1,c);
        break;
    }
}

/***************************************************************************
*  Filename: INT_EE.C
****************************************************************************
*   Author:    Stephen Humberd/Rodger Richey
*   Company:   Microchip Technology
*   Revision:  RevA0
*   Date:      5-31-00
*   Compiled using CCS PICC
****************************************************************************
*
*  This file contains the routines to read and write data to the internal
*  EEPROM on the PIC16F877 device.
*
***************************************************************************/

/**************************************************************
**  void WriteIntEE(char Addr, char *DataPtr, char NumBytes) **
**  This routine writes data to the internal EEPROM on the   **
**  PIC16F8xx devices.  It is designed to write 1 byte up to **
**  255 bytes.                                               **
**    Addr: address in EEPROM to begin writing               **
**    DataPtr: address in RAM to read data for writing       **
**    NumBytes: number of bytes to write                     **
**************************************************************/
void WriteIntEE(char Addr, char *DataPtr, char NumBytes)
{
    char i;

    EEADR = Addr;  // Set starting address
    EEPGD = 0;         // Select data EEPROM
    WREN = 1;          // Enable Writes

    for(i=0;i<NumBytes;i++)
    {
        EEIF = 0;      // Clear interrupt flag
        EEDATA = *DataPtr;// Set data to write
        WREN = 1;      // Ensure that writes are enabled
        EECON2 = 0x55;// Write sequence
        EECON2 = 0xaa;
        EEWR = 1;      // Start write operation

        while(!EEIF);// Wait for write to complete
        DataPtr++;     // Increment pointer to RAM
        EEADR++;       // Increment EEPROM address
    }
    WREN = 0;          // Disable writes
}
```

```
/**************************************************************
**  void ReadIntEE(char Addr, char *DataPtr, char NumBytes)  **
**  This routine reads data from the internal EEPROM on the  **
**  PIC16F8xx devices.  It is designed to read 1 byte up to  **
**  255 bytes.                                               **
**    Addr: address in EEPROM to begin reading               **
**    DataPtr: address in RAM to write data that was read    **
**    NumBytes: number of bytes to read                      **
**************************************************************/
void ReadIntEE(char Addr, char *DataPtr, char NumBytes)
{
    char i;

    EEADR = Addr;  // Set address in EEPROM
    EEPGD = 0;         // Select data EEPROM

    for(i=0;i<NumBytes;i++)
    {
        EERD = 1;      // Start read operation
        *DataPtr = EEDATA;// Write read data into RAM
        EEADR++;       // Increment EEPROM address
        DataPtr++;     // Increment RAM address
    }
}

/**************************************************************
**  void Write_data(void)                                    **
**  This routine writes the item names and prices to the     **
**  internal EEPROM.                                         **
**************************************************************/
void Write_data(void)
{
    WriteIntEE(0x50, Item_1, 9);
    WriteIntEE(0x60, Item_2, 9);
    WriteIntEE(0x70, Price_1, 5);
    WriteIntEE(0x78, Price_2, 5);
}

/**************************************************************
**  void Read_data(void)                                     **
**  This routine reads the item names and prices from the    **
**  internal EEPROM.                                         **
**************************************************************/
void Read_data(void)
{
    ReadIntEE(0x50, Item_1, 9);
    ReadIntEE(0x60, Item_2, 9);
    ReadIntEE(0x70, Price_1, 5);
    ReadIntEE(0x78, Price_2, 5);
}

/***************************************************************************
*  Filename: GET_INFO.C
****************************************************************************
*   Author:    Stephen Humberd/Rodger Richey
*   Company:   Microchip Technology
*   Revision:  RevA0
*   Date:      5-31-00
*   Compiled using CCS PICC
****************************************************************************
*  This file contains the routine to prompt the user to enter new
*  information such as username, password and telephone number.  It also
*  has a routine to read out the contents of the internal EEPROM and dump
*  it to the user’s terminal.
***************************************************************************/
```

```
/**************************************************************
**  void Get_username(void)                                  **
**  Requests and reads the user name from the input terminal.**
**************************************************************/
void Get_username(void)
{
    i=0;
    // Print request to terminal
    printf(“%c[2J”,esc);
    printf(“%c[12;20H 32 chars max”,esc);
    printf(“%c[10;20H Enter user name: “,esc);

    while(1)       // Read characters until a
    {                     // CR is read or 32 chars
        user[i]=0; // have been read
        ch=GETC();
        if(ch==0x0D)
            break;
        putc(ch);
        if(ch != 0x08)
        {
            user[i]=ch;
            i++;
        }
        if(i==32) break;
    }
    // write user name to the Internal EEPROM
    WriteIntEE(0, user, 0x1f);
}

/**************************************************************
**  void Get_password(void)                                  **
**  Requests and reads the password from the input terminal. **
**************************************************************/
void Get_password(void)
{
    i=0;
    // Print request to terminal
    printf(“%c[2J”,esc);
    printf(“%c[12;20H 32 chars max”,esc);
    printf(“%c[10;20H Enter password: “,esc);

    while(1)       // Read characters until a
    {                     // CR is read or 16 chars
        pass[i]=0; // have been read
        ch=getc();
        if(ch==0x0D)
            break;
        if(ch != 0x0A)// line feed
        {
            putc(ch);
            if(ch != 0x08)
            {
                pass[i]=ch;
                i++;
            }
        }
        if(i==16) break;
    }
    // write password to the Internal EEPROM
    WriteIntEE(0x20, pass, 0x1f);
}
```

```
/**************************************************************
**  void Get_phone(void)                                     **
**  Requests and reads the telephone number from the input   **
**  terminal.                                                **
**************************************************************/
void Get_phone()
{
    // Print request to terminal
    printf(“%c[2J”,esc);
    printf(“%c[12;20H 16 chars max”,esc);
    printf(“%c[10;20H Enter phone number: “,esc);

    i=0;
    while(1)       // Read characters until a
    {                      // CR is read or 16 chars
        phone[i]=0;// have been read
        ch=getc();
        if(ch==0x0D)
            break;
        if(ch != 0x0A)// line feed
        {
            putc(ch);
            if(ch != 0x08)
            {
                phone[i]=ch;
                i++;
            }
        }
        if(i==16) break;
    }

    // write phone number to the Internal EEPROM
    WriteIntEE(0x40, phone, 16);
}

/**************************************************************
**  void Write_data_to_rs232(void)                           **
**  Debugging routine that dumps the contents of the EEPROM  **
**  to the terminal.  Must uncomment line in main().         **
**************************************************************/
void Write_data_to_rs232(void)
{
    // Read and print data EEPROM contents
    for(index=0;index<0x50;index++)
    {
        ReadIntEE(index, &ch, 1);
        temp=isalnum(ch);
        if(temp)
            putc(ch);
        else
            putc(‘.’);
    }
}

/**************************************************************
**  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;
    CTS=0; // enable send
    while(ch != 0x1b)
    {
        lcd_putc(“\fPC Terminal menu”);
        printf(“%c[2J”,esc);
        printf(“%c[8;25H 1   Enter user name”,esc);
        printf(“%c[10;25H 2   Enter password”,esc);
        printf(“%c[12;25H 3   Enter phone number”,esc);
        printf(“%c[17;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;
        }
    }
    CTS = 1;   // disable send
}

/**************************************************************************
*  Filename: F877.H
***************************************************************************
*   Author:    Stephen Humberd/Rodger Richey
*   Company:   Microchip Technology
*   Revision:  RevA0
*   Date:      5-31-00
*   Compiled using CCS PICC
***************************************************************************
*
*  This is a header file containing register and bit definitions for the
*  PIC16F877.
*
*  External Clock provided by Si2400 modem = 9.8304MHz
*
**************************************************************************/


//----- Registers -------------------------------------------------------
#byte INDF=0x000
#byte TMR0=0x001
#byte PCL=0x002
#byte STATUS=0x003
#byte FSR=0x004
#byte PORTA=0x005
#byte PORTB=0x006
#byte PORTC=0x007
#byte PORTD=0x008
#byte PORTE=0x009
#byte PCLATH=0x00A
#byte INTCON=0x00B
#byte PIR1=0x00C
#byte PIR2=0x00D
#byte TMR1L=0x00E
#byte TMR1H=0x00F
#byte T1CON=0x010
#byte TMR2=0x011
#byte T2CON=0x012
#byte SSPBUF=0x013
#byte SSPCON=0x014
```

```
#byte CCPR1L=0x015
#byte CCPR1H=0x016
#byte CCP1CON=0x017
#byte RCSTA=0x018
#byte TXREG=0x019
#byte RCREG=0x01A
#byte CCPR2L=0x01B
#byte CCPR2H=0x01C
#byte CCP2CON=0x01D
#byte ADRESH=0x01E
#byte ADCON0=0x01F

#byte OPTION_REG=0x081
#byte TRISA=0x085
#byte TRISB=0x086
#byte TRISC=0x087
#byte TRISD=0x088
#byte TRISE=0x089
#byte PIE1=0x08C
#byte PIE2=0x08D
#byte PCON=0x08E
#byte SSPCON2=0x091
#byte PR2=0x092
#byte SSPADD=0x093
#byte SSPSTAT=0x094
#byte TXSTA=0x098
#byte SPBRG=0x099
#byte ADRESL=0x09E
#byte ADCON1=0x09F

#byte EEDATA=0x10C
#byte EEADR=0x10D
#byte EEDATH=0x10E
#byte EEADRH=0x10F

#byte EECON1=0x18C
#byte EECON2=0x18D


//----- STATUS Bits --------------------------------------------------------

#bit IRP =STATUS.7
#bit RP1 =STATUS.6
#bit RP0 =STATUS.5
#bit NOT_TO = STATUS.4
#bit NOT_PD = STATUS.3
#bit Z =STATUS.2
#bit DC =STATUS.1
#bit C = STATUS.0

//----- INTCON Bits --------------------------------------------------------

#bit GIE =  INTCON.7
#bit PEIE = INTCON.6
#bit T0IE = INTCON.5
#bit INTE = INTCON.4
#bit RBIE = INTCON.3
#bit T0IF = INTCON.2
#bit INTF = INTCON.1
#bit RBIF = INTCON.0

//----- PIR1 Bits ----------------------------------------------------------

#bit PSPIF =PIR1.7
#bit ADIF =PIR1.6
#bit RCIF =PIR1.5
```

```
#bit TXIF =PIR1.4
#bit SSPIF =PIR1.3
#bit CCP1IF =PIR1.2
#bit TMR2IF =PIR1.1
#bit TMR1IF =PIR1.0

//----- PIR2 Bits ----------------------------------------------------------

#bit EEIF =PIR2.4
#bit BCLIF =PIR2.3
#bit CCP2IF =PIR2.0

//----- T1CON Bits ---------------------------------------------------------

#bit T1CKPS1 =T1CON.5
#bit T1CKPS0 =T1CON.4
#bit T1OSCEN =T1CON.3
#bit NOT_T1SYNC =T1CON.2
#bit T1INSYNC =T1CON.2
#bit TMR1CS =T1CON.1
#bit TMR1ON =T1CON.0

//----- T2CON Bits ---------------------------------------------------------

#bit TOUTPS3 =T2CON.6
#bit TOUTPS2 =T2CON.5
#bit TOUTPS1 =T2CON.4
#bit TOUTPS0 =T2CON.3
#bit TMR2ON =T2CON.2
#bit T2CKPS1 =T2CON.1
#bit T2CKPS0 =T2CON.0

//----- SSPCON Bits --------------------------------------------------------

#bit WCOL =SSPCON.7
#bit SSPOV =SSPCON.6
#bit SSPEN =SSPCON.5
#bit CKP =SSPCON.4
#bit SSPM3 =SSPCON.3
#bit SSPM2 =SSPCON.2
#bit SSPM1 =SSPCON.1
#bit SSPM0 =SSPCON.0

//----- CCP1CON Bits -------------------------------------------------------

#bit CCP1X =CCP1CON.5
#bit CCP1Y =CCP1CON.4
#bit CCP1M3 =CCP1CON.3
#bit CCP1M2 =CCP1CON.2
#bit CCP1M1 =CCP1CON.1
#bit CCP1M0 =CCP1CON.0

//----- RCSTA Bits ---------------------------------------------------------

#bit SPEN =RCSTA.7
#bit RX9 =RCSTA.6
#bit SREN =RCSTA.5
#bit CREN =RCSTA.4
#bit ADDEN = RCSTA.3
#bit FERR =RCSTA.2
#bit OERR =RCSTA.1
#bit RX9D =RCSTA.0

//----- CCP2CON Bits -------------------------------------------------------

#bit CCP2X =CCP2CON.5
```

```
#bit CCP2Y =CCP2CON.4
#bit CCP2M3 =CCP2CON.3
#bit CCP2M2 =CCP2CON.2
#bit CCP2M1 =CCP2CON.1
#bit CCP2M0 =CCP2CON.0

//----- ADCON0 Bits --------------------------------------------------------

#bit ADCS1 =ADCON0.7
#bit ADCS0 =ADCON0.6
#bit CHS2 =ADCON0.5
#bit CHS1 =ADCON0.4
#bit CHS0 =ADCON0.3
#bit GO =ADCON0.2
#bit NOT_DONE =ADCON0.2
#bit CHS3 =ADCON0.1
#bit ADON =ADCON0.0

//----- OPTION Bits --------------------------------------------------------

#bit NOT_RBPU =OPTION_REG.7
#bit INTEDG =OPTION_REG.6
#bit T0CS =OPTION_REG.5
#bit T0SE =OPTION_REG.4
#bit PSA =OPTION_REG.3
#bit PS2 =OPTION_REG.2
#bit PS1 =OPTION_REG.1
#bit PS0 =OPTION_REG.0

//----- TRISE Bits ---------------------------------------------------------

#bit IBF = TRISE.7
#bit OBF = TRISE.6
#bit IBOV = TRISE.5
#bit PSPMODE = TRISE.4

//----- PIE1 Bits ----------------------------------------------------------

#bit PSPIE =PIE1.7
#bit ADIE =PIE1.6
#bit RCIE =PIE1.5
#bit TXIE =PIE1.4
#bit SSPIE =PIE1.3
#bit CCP1IE =PIE1.2
#bit TMR2IE =PIE1.1
#bit TMR1IE =PIE1.0

//----- PIE2 Bits ----------------------------------------------------------

#bit EEIE =PIE2.4
#bit BCLIE =PIE2.3
#bit CCP2IE =PIE2.0

//----- PCON Bits ----------------------------------------------------------

#bit NOT_POR = PCON.1
#bit NOT_BOR = PCON.0

//----- SSPCON2 Bits -------------------------------------------------------

#bit GCEN =SSPCON2.7
#bit ACKSTAT =SSPCON2.6
#bit ACKDT =SSPCON2.5
#bit ACKEN =SSPCON2.4
#bit RCEN =SSPCON2.3
#bit PEN =SSPCON2.2
```

```
#bit RSEN =SSPCON2.1
#bit SEN =SSPCON2.0

//----- SSPSTAT Bits -------------------------------------------------------

#bit SMP =SSPSTAT.7
#bit CKE =SSPSTAT.6
#bit D_A =SSPSTAT.5
#bit P =SSPSTAT.4
#bit S =SSPSTAT.3
#bit R_W =SSPSTAT.2
#bit UA =SSPSTAT.1
#bit BF =SSPSTAT.0

//----- TXSTA Bits ---------------------------------------------------------

#bit CSRC =TXSTA.7
#bit TX9 =TXSTA.6
#bit TXEN =TXSTA.5
#bit SYNC =TXSTA.4
#bit BRGH =TXSTA.2
#bit TRMT =TXSTA.1
#bit TX9D =TXSTA.0

//----- ADCON1 Bits --------------------------------------------------------

#bit ADFM =ADCON1.5
#bit PCFG3 =ADCON1.3
#bit PCFG2 =ADCON1.2
#bit PCFG1 =ADCON1.1
#bit PCFG0 =ADCON1.0

//----- EECON1 Bits --------------------------------------------------------

#bit EEPGD = EECON1.7
#bit WRERR = EECON1.3
#bit WREN = EECON1.2
#bit EEWR = EECON1.1
#bit EERD = EECON1.0

/***************************************************************************
*  Filename: S7600.H
****************************************************************************
*   Author:     Stephen Humberd/Rodger Richey
*   Company:    Microchip Technology
*   Revision:   RevA0
*   Date:       5-31-00
*   Compiled using CCS PICC

****************************************************************************
*
*  This file contains the register address definitions of the S-7600A.
*
*  External Clock provided by Si2400 modem = 9.8304MHz
*
***************************************************************************/

// Seiko S-7600A TCP/IP Stack IC
// Header File

// #case
#define   Revision                 0x00
#define   General_Control          0x01
#define   General_Socket_Location  0x02
#define   Master_Interrupt         0x04
#define   Serial_Port_Config       0x08
```

```
#define   Serial_Port_Int          0x09
#define   Serial_Port_Int_Mask     0x0a
#define   Serial_Port_Data         0x0b
#define   BAUD_Rate_Div_L          0x0c
#define   BAUD_Rate_Div_H          0x0d
#define   Our_IP_Address_L         0x10
#define   Our_IP_Address_M         0x11
#define   Our_IP_Address_H         0x12
#define   Our_IP_Address_U         0x13
#define   Clock_Div_L              0x1c
#define   Clock_Div_H              0x1d
#define   Socket_Index             0x20
#define   Socket_TOS               0x21
#define   Socket_Config_Status_L   0x22
#define   Socket_Status_M          0x23
#define   Socket_Activate          0x24
#define   Socket_Interrupt         0x26
#define   Socket_Data_Avail        0x28
#define   Socket_Interrupt_Mask_L  0x2a
#define   Socket_Interrupt_Mask_H  0x2b
#define   Socket_Interrupt_L       0x2c
#define   Socket_Interrupt_H       0x2d
#define   Socket_Data              0x2e
#define   TCP_Data_Send            0x30
#define   Buffer_Out_L             0x30
#define   Buffer_Out_H             0x31
#define   Buffer_In_L              0x32
#define   Buffer_In_H              0x33
#define   Urgent_Data_Pointer_L    0x34
#define   Urgent_Data_Pointer_H    0x35
#define   Their_Port_L             0x36
#define   Their_Port_H             0x37
#define   Our_Port_L               0x38
#define   Our_Port_H               0x39
#define   Socket_Status_H          0x3a
#define   Their_IP_Address_L       0x3c
#define   Their_IP_Address_M       0x3d
#define   Their_IP_Address_H       0x3e
#define   Their_IP_Address_U       0x3f
#define   PPP_Control_Status       0x60
#define   PPP_Interrupt_Code       0x61
#define   PPP_Max_Retry            0x62
#define   PAP_String               0x64
```

NOTES:

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

DNV Certification, Inc.
USA

ANSI • RAB
QMS
ACCREDITED

DNV MSC
The Netherlands
Accredited by the RvA

DNV

ISO 9001 / QS-9000
REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai)
Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai)
Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai)
Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai)
Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai)
Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/18/02