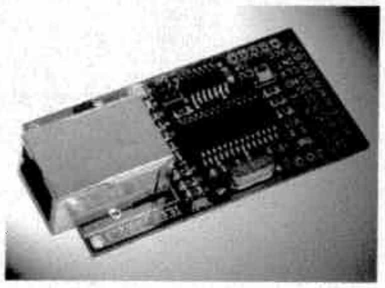
PIC MCU WEB SERVER

J. Gorasia

version 0.1.1

Contents



Chapter 1

Introduction

1.1 Audience

This document is a guide to develop a simple web server, the WEB1 using 8 bit PIC Microcontrollers. Preferably, the reader would have previous experience with PIC MCUs, and using the C18 compiler. It is also expected that the reader have some breadboarding skills, and the ability to understand basic circuit concepts. A knowledge of TCP/IP and/or Ethernet is not required.

1.2 Capabilities

Microchip developed a TCP/IP stack for use with their range of Ethernet enabled products, like the ENC29J60 and PIC18F97J60. This software stack is enables these products to become a functioning web server. Among the services the web server can carry out is: HTTP, FTP and DHCP. This makes it possible to use these products for applications like remote sensing, home automation and others.

1.3 About me

I am a Mechanical Engineering student at Franklin W. Olin College of Engineering in Needham, MA. This book is the culmination of a semester long independent study on developing a web server using PIC MCUs.

1.4 Acknowledgments

This endeavor would have been much more difficult if not for the advice of Professor Bradley Minch and the previous work by Jorge Amodio. I would also like to thank Professor Gill Pratt for providing me with the space to work, and the Boston Engineering SCOPE team for their endless entertainment.

Chapter 2

Ethernet Basics

As Alaska Senator Ted Stevens said, the Internet is a series of tubes. These tubes are very important as the Internet has boomed to become a crucial medium for communication. The Internet is a global system of interconnected computer networks that interchange data by packet switching using the standardized Internet Protocol Suite (TCP/IP). It is a "network of networks" that consists of millions of private and public, academic, business, and government networks of local to global scope that are linked by copper wires, fiber-optic cables, wireless connections, and other technologies. The Internet carries various information resources and services, such as electronic mail, online chat, file transfer and file sharing, online gaming, and the inter-linked hypertext documents and other resources of the World Wide Web (WWW).

Ethernet is an asynchronous Carrier Sense Multiple Access with Collision Detect (CSMA/CD) protocol/interface with a payload size of 45-1500 octets.It defines a number of wiring and signaling standards for the Physical Layer of the OSI networking model, through means of network access at the Media Access Control (MAC)/Data Link Layer, and a common addressing format.

Ethernet is a data link and physical layer protocol defined by the IEEE 802.3 specification[1].It comes in many flavors, defined by maximum bit rate, mode of transmission and physical transmission medium.

- Maximum Bit Rate (Mbits/s): 10,100,100,etc
- Mode of transmission: Broadband, Baseband
- Physical Transmission Medium: Coax, Fiber, UTP,etc

The rest of this chapter will cover more details about the TCP/IP protocol. It is useful to learn more about this before diving into the PIC MCU implementation as it will make many concepts and terminology more understandable.

2.1 TCP/IP Model

2.1.1 Layers

To understand how Ethernet works, it is first necessary to understand the concept of packet encapsulation, and how the protocol stack fits into this concept. The transfer protocol is divided up into many layers. Each layer is responsible for a particular level of functionality. Each higher layer in the model utilizes the underlying layers in a somewhat independent fashion. The interesting result of this is that the underlying technologies of the different layers have progressed independently of each other. For instance,

[1]IEEE 802.3 ETHERNET - http://www.ieee802.org/3/

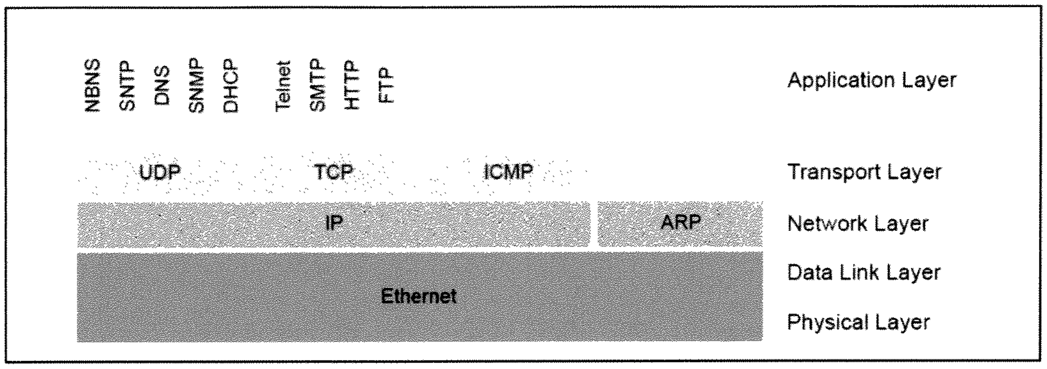


Figure 2.1: TCP/IP Stack. Notice how the different layers have many different kinds of services. A packet will only have to use a single service from each layer as it travels down the stack.

As Figure 2.1 shows, the TCP/IP model is broken up into five layers. To describe their function, we will track the path of a packet traveling down the stack.

1. A web browser would generate a HTTP request using an application specific command.
2. This request would be passed down to the TCP layer, which would construct a TCP packet consisting of a TCP header and TCP data. TCP header contains information particular to the TCP protocol, such as packet sequencing information, checksum information[2] and the source and destination port number (HTTP generally has a port number of 80[3]).
3. At the IP protocol layer, an IP datagram is constructed to hold the TCP packet. The IP header contains information about type of service, checksum information, protocol type (06h for TCP) and the source and destination IP addresses. The data field of the IP datagram contains the complete TCP packet to be transmitted.
4. At the data link/physical layer, the IP datagram is transported across the network using the IEEE 802.3 protocol. A MAC (Media Access Control) frame consists of a MAC header and a MAC payload (data). The MAC header contains information such as the source MAC address, the destination MAC address, and the length of the frame. The payload field contains he complete IP datagram to be transported.

[2]for basic error checking
[3]For a complete listing, visit: http://www.iana.org/assignments/port-numbers

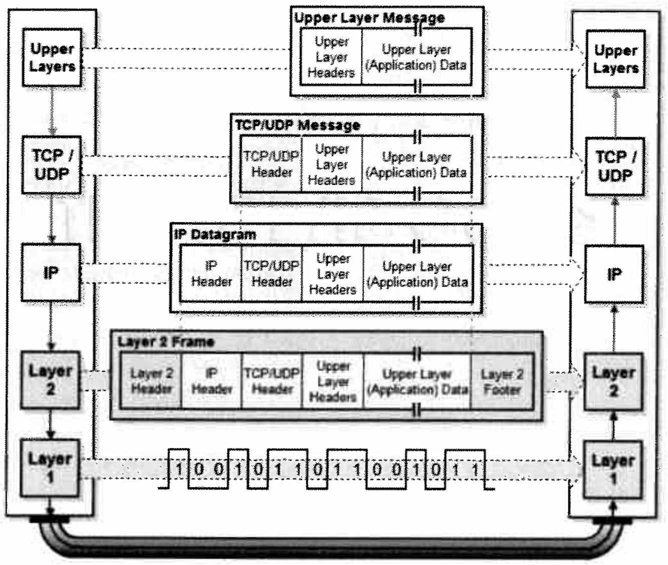


Figure 2.2: The evolution of the packet as it travels down the stack. Notice how the message stays essentially the same, and just gets added information as it travels down the stack.

2.1.2 Acronyms

There are many acronyms that emerge when looking through the stack. They are not very intuitively named, and can be very daunting for beginners. Here is a list of the ones implemented in this stack. Note that this is not a complete list of all the different acronyms in the TCP/IP model.

Acronym	Full Name	Function
ARP	Address Resolution Protocol	Method for finding a host's link layer (hardware) address when only its Internet Layer (IP) or some other Network Layer address is known
DNS	Domain Name Server	Hierarchical naming system for computers, services, or any resource participating in the Internet, which translates human readable names into their numerical identifiers
DHCP	Dynamic Host Configuration Protocol	Used by networked devices (clients) to obtain the parameters necessary for operation in an Internet Protocol network (IP address, ...)
FTP	File Transfer Protocol	Used to transfer data from one computer to another through the network.
HTTP	HyperText Transfer Protocol	Used for retrieving inter-linked text documents (hypertext)
SNTP	Simple Network Time Protocol	Internet protocol used to synchronize the clocks of computers to some time reference
SMTP	Simple Mail Transfer Protocol	Internet standard for electronic mail (e-mail) transmission across Internet Protocol (IP) networks
TCP	Transmission Control Protocol	TCP provides reliable, ordered delivery of a stream of bytes from one program on one computer to another program on another computer.
UDP	User Datagram Protocol	UDP is TCP without as much overhead and thus less error checking.
ICMP	Internet Control Message Protocol	Used by networked devices to communicate with each other, primarily to transmit error messages.
IP	Internet Protocol	The stack implements IPv4.
Telnet	Telecommunication network	Provides access to a command-line interface on a remote machine
RARP	Reverse Address Resolution Protocol	Used to translate hardware interface addresses to protocol addresses, such as a MAC address to an IP address.

2.2 Physical Medium Overview

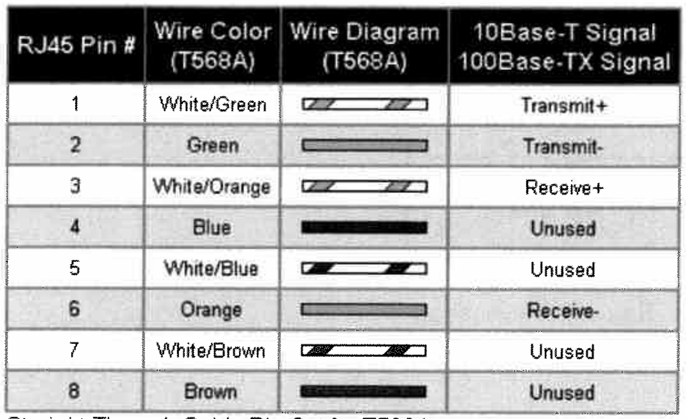
If you slice open an Ethernet cable, you will see 8 different wires. They exist as 4 twisted pairs, where each pair consists of one solid colored wire and one with white stripes. The green wires are the transmit signal while the orange cables are the receive signal. The blue and brown wires are only used by Gigabit Ethernet.

Twisting wires decreases interference because the loop area between the wires (which determines the magnetic coupling into the signal) is reduced. The two wires carry equal and opposite signals (differential mode) which are combined by addition at the destination. The common-mode noise from the two wires (mostly) cancel each other in this addition because the two wires have similar amounts of EMI that are 180 degrees out of phase. This results in the same effect as subtraction. Differential mode also reduces electromagnetic radiation from the cable, along with the attenuation that it causes[4].

The jack is a RJ45 jack which is a standard communications jack. It is possible to purchase jacks with integrated magnetics (used to filter the signal) and LEDs (indicators for the operation of the device). This is recommended as they are not much more expensive than the usual jacks.

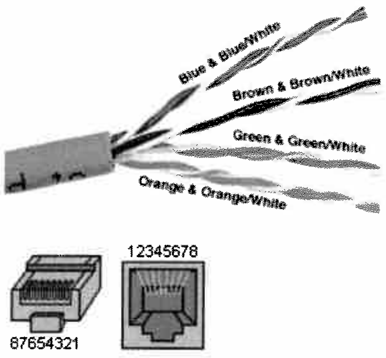
There are two indicator LEDs(usually) on the jack. The left one is amber or green and shows the connection speed while the right one is green and shows the signal activity. If the left LED is amber, the device is acting as a Gigabit connection, if green, the device is acting as a 100-Mbps connection while if off, it is operating as a

[4] `http://en.wikipedia.org/wiki/Twisted_pair`



RJ45 Pin #	Wire Color (T568A)	Wire Diagram (T568A)	10Base-T Signal 100Base-TX Signal
1	White/Green	▨	Transmit+
2	Green	■	Transmit-
3	White/Orange	▨	Receive+
4	Blue	■	Unused
5	White/Blue	▨	Unused
6	Orange	■	Receive-
7	White/Brown	▨	Unused
8	Brown	■	Unused

Straight-Through Cable Pin Out for T568A



RJ45 Plug and Jack Pin Out

Figure 2.3: Ethernet cable and RJ45 pin overview. Taken from `http://www.ertyu.org/steven_nikkel/ethernetcables.html`

10-Mbps connection. If the right LED is blinking, there is activity on the port. If it is on, that means a link has been established. However, if it is off, no link has been established.

2.3 Ethernet Specifications

As a final detail before we get into the implementation, we need to discuss the contents of the final IP packet that is sent between networked devices. This is informative as it really helps you appreciate the design that went into it. The following information is from AN1120 from Microchip,"*Ethernet Theory of Operation*"

1. Preamble - Seven octets of 55h. The preamble is present to allow the receiver to lock onto the stream of data before the actual frame arrives.

2. Start-of-Frame Delimiter - 10101011b (as seen on the physical medium). The SFD is sometimes considered to be part of the preamble. This is why the preamble is sometimes described as eight octets.

3. Destination Address - The 6-octet MAC address of the destination hardware

4. Source Address - The 6-octet MAC address of the source hardware.

5. Length/Type - If the value in this 2-octet field is $\leq$ 1500 (decimal), this represents the number of octets in the payload. If the value is $\geq$ 1536, this represents the EtherType (payload type). The following are the most common EtherType values: IPv4 = 0800h, IPv6 = 86DDh, ARP = 0806h, RARP = 8035h.

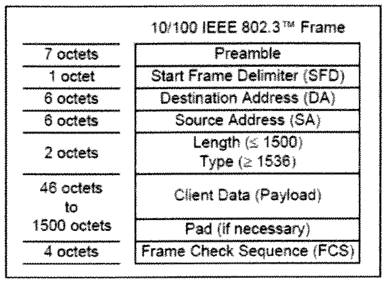
6. Payload - The client data, such as an IP datagram, etc. The minimum payload size is 46 octets; the maximum payload size is 1500 octets. While payloads below or above these limits do not meet the IEEE 802.3 specification, there is varied support for these payloads depending on the particular vendor.

7. Pad - Since the minimum payload size is 46 octets, pad octets must be inserted to reach this minimum if the payload size is less than 46 octets.

8. Frame Check Sequence (FCS) - The value of the 4-octet FCS field is calculated over the source address, destination address, length/type, data and pad fields using a 32-bit Cyclic Redundancy Check (CRC).

9. End-of-Stream Delimiter (ESD) - In 100 Mb/s operation, the PHY transmits a /T/R/ symbol pair after the FCS (during the inter-frame gap) to denote the end of the frame.

10. Special TP_IDL signal and network silence indicates the end of the frame.



10/100 IEEE 802.3™ Frame

7 octets	Preamble
1 octet	Start Frame Delimiter (SFD)
6 octets	Destination Address (DA)
6 octets	Source Address (SA)
2 octets	Length (≤ 1500) Type (≥ 1536)
46 octets to 1500 octets	Client Data (Payload)
	Pad (if necessary)
4 octets	Frame Check Sequence (FCS)

Figure 2.4: Basic IP frame format

Chapter 3

Build the web server

In this chapter, the steps needed to build the web server circuit is laid out. We will be using a PIC MCU with a large memory as the TCP/IP protocol stack takes up a lot of data and program memory. I broke up the circuit into subsystems in order to better explain how they are to be wired.

3.1 Subsystems

3.1.1 PIC Microcontroller (PIC18F4620)

The PIC Microcontroller I chose was a PIC18F4620. It is a relatively normal PIC MCU with a large program and data memory, 64 kbytes and 3968 bytes respectively. The pin numbering below is for the DIP version which I used. Note that the smaller version, 18F2620 (or similar device) could be used as many(7) pins were used for the LCD(labeled in green), which is optional, and 8 pins remain unused(labeled in blue) in this device.

Pin	Function	Remarks	Pin	Function	Remarks
1	$\overline{MCLR}$	hardware Master CLeaR	21	RD2	DB6 for LCD
2	AN0	Analog Input 0	22	RD3	DB7 for LCD
3	AN1	Analog Input 1 (not set up)	23	RC4	SDI for SPI interface
4	RA2	LED0, Status Indicator	24	RC5	SDO for SPI interface
5	RA3	LED1	25	TX	Transmit for the USART
6	RA4	LED2	26	RX	Receive for the USART
7	RA5	Enable for LCD	27	RD4	RS for LCD
8	RE0	Unused	28	RD5	R/W for LCD
9	RE1	Unused	29	RD6	Unused
10	RE2	Unused	30	RD7	Unused
11	Vcc	5V	31	VDD	GND
12	Vdd	Ground	32	VCC	5V
13	OSC1	Oscillator in	33	IN_0	Push button 0
14	OSC2	Oscillator out	34	IN_1	Push button 1
15	RC0	Unused	35	ENC_INT	Optional Input for ENC28J60 INT
16	RC1	Unused	36	ENC_CS	ENC28J60 CS
17	RC2	Unused	37	XPROM_CS	25LC256 Serial EEPROM CS
18	RC3	Used as SCK for SPI interface	38	RESET	Optional ENC28J60 RESET
19	RD0	DB4 for LCD	39	PGC	Clock signal for programming
20	RD1	DB5 for LCD	40	PGD	Data signal for programming

3.1.2 Ethernet Network Controller (ENC28J60)

The ENC28J60 does the hard work of providing the PASCAL layer of Ethernet connectivity to the PIC MCU. It connects to the PIC MCU via SPI, thus requires four pins of the MCU. Instead of building this circuit myself, I went out and purchased the nic28 board from LJCV Electronics. This board has the RJ45 jack, ENC28J60 chip, a 74ACT125M chip and some discrete components. The 74ACT125M is a tri-state buffer, which allows the SPI bus to be shared easily by reducing interference.

This circuit could just have as easily been put together, but buying this assembled board offers the advantage of a compact package that is tested. It is easy to make mistakes wiring up this circuit, and buying a premade solution prevents this from being a problem.

3.1.3 Serial EEPROM (25LC256)

Webpages are usually much too big to be stored in the program memory of the PIC MCU. By default the WEB1 server will use an external serial EEPROM to store the webpages. It is important to make sure that the webpages are small enough to fit on the serial EEPROM. The stack reserves the first 64 bytes to save the application configuration information, but the rest is available for use.

The commands to the EEPROM are pretty simple. To read more on it, consult the datasheet, and the C files in the WEB1 project.

3.1.4 Parallel LCD (Hitachi 44780 based)

LCDs are a great way to communicate with people, as they are able to display a lot of information, with relatively simple commands. Most LCDs come with an LCD controller attached, the most popular one being the Hitachi 44780 chip. This makes the interface to the LCD very simple, requiring 11 pins at most: 8 data pins, one read/write, one set/reset and one enable pin. You can also operate it in 4 bit mode, where only 4 data pins are used. The LCD is capable of producing any of the ACSII characters using a very close to ACSII syntax. Many Japanese characters are also possible. To find out more about the controller, as well as the timing sequence needed for its operation, go to `http://meteosat.pessac.free.fr/Cd_elect/Doc-CI/LCD/lcd-htm/LCD%20Hitachi.htm`.

3.2 Circuit diagram

3.3 Power on test

To test if the circuit is working, plug an Ethernet cable into the RJ45 jack and into a network port. If you see the LEDs light up, then you have the nic28 board operational. Everything should be all set, and you can proceed to programming the PIC MCU.

If you have problems getting this working, an incremental approach would be best. The LEDs on the RJ45 will light up if connected to a network and nothing else (other than power). You should be able to connect to the PIC MCU using a programmer. If all else fails, just look through the circuit diagram to see if any mistakes occured.

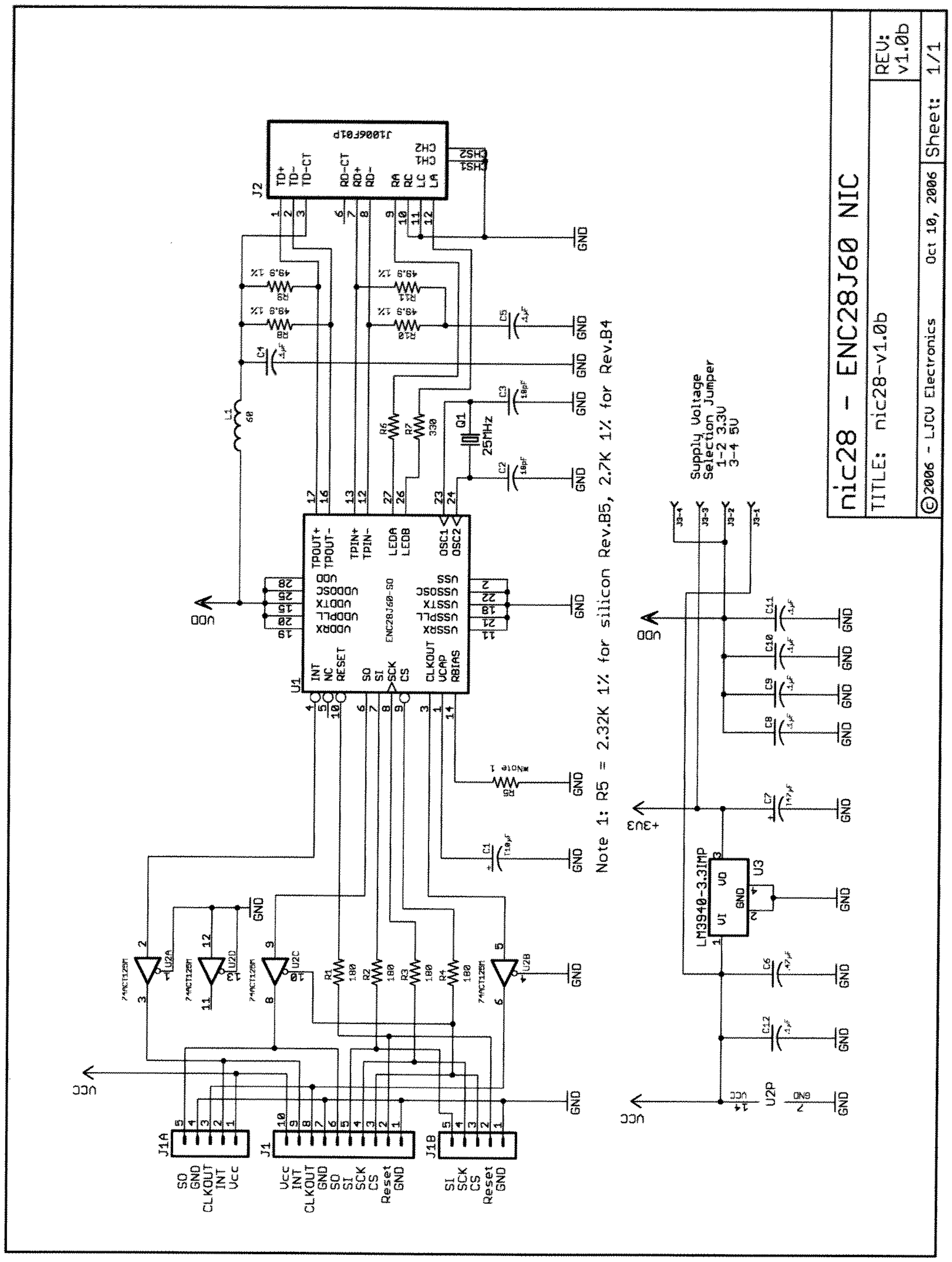


Figure 3.1: NIC28

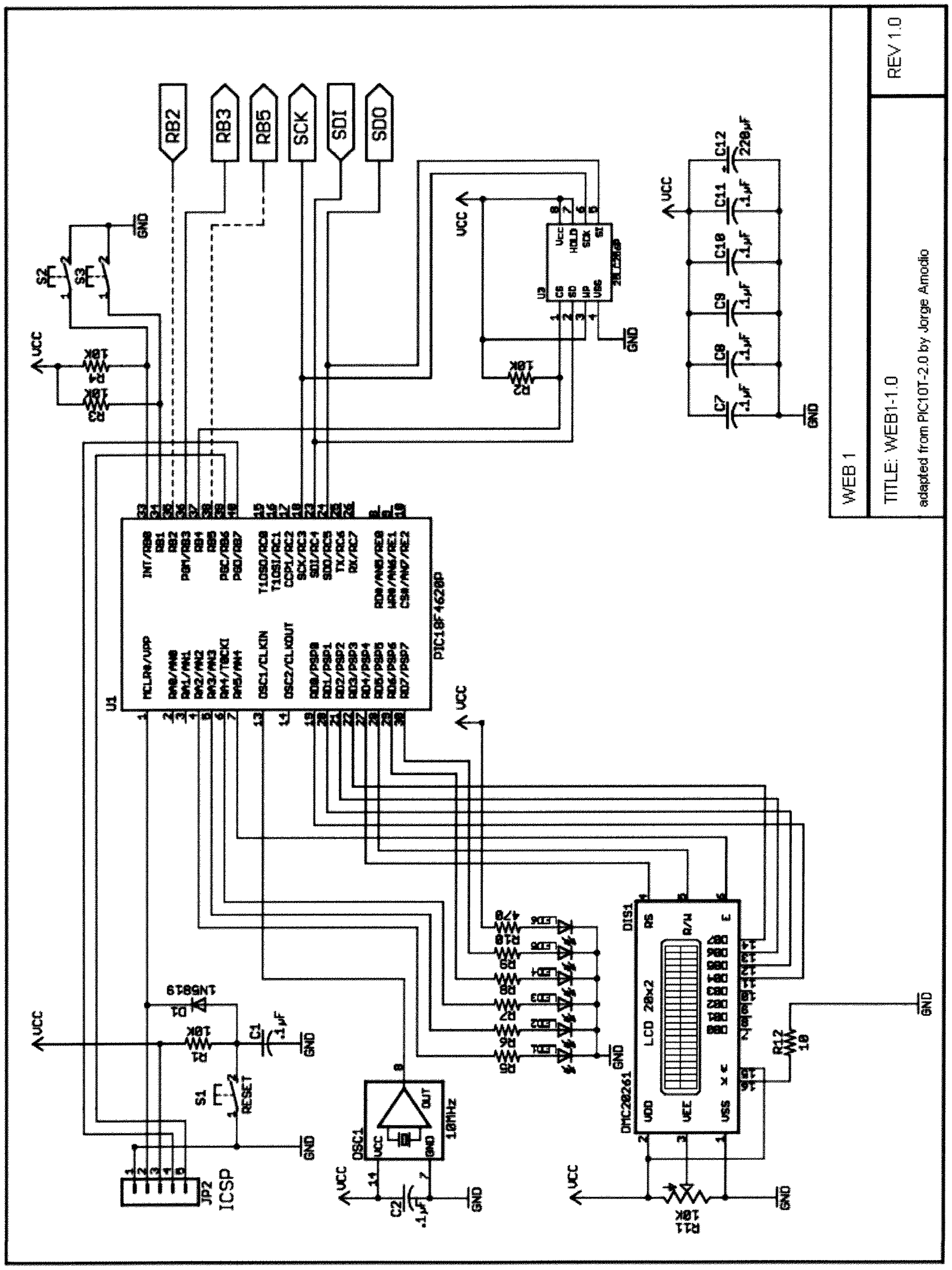


Figure 3.2: WEB 1 circuit. The connections to the right are all going towards the nic28. *IMPORTANT*, do not connect RB2 and RB5to the INT and CLK of the nic28. Badness will ensue.

Chapter 4

Getting the web server operational

4.1 Basic environment considerations

Before you can begin doing anything, you need to get a working environment to develop in. All the code presented here uses the MPLAB IDE (integrated development environment) from Microchip (version 8.10 at time of writing) and the C18 compiler. This is useful as this allows us to develop in C, a much more enjoyable experience compared to working in assembly.

To get MPLAB, visit this[1] site. MPLAB is distributed at no cost.

To get the C18 compiler, visit this site. If you want a free student version, you have to register with Microchip. This will give you a 60 day trial of the full compiler, after which the code optimizations are disabled. The optimizations are not required for the web server to function (the code still manages to fit on the PIC MCU).

You will now be all set to develop the web server.

4.2 Obtain TCP/IP Stack

The software for the stack is kept at xxxix. Extract the zip file and open up the Microchip MPLAB workspace, Web1. Open up the file using MPLAB. You can also find a very similar stack at `http://www.ljcv.net/files/MCHPTCP_v3.75.6.zip`. Use the PIC10T project, which is essentially the same as the WEB1.

4.3 Build and import

If your environment is set up correctly, all you need is to build the file. Then, using your programmer of choice (I use the PICKIT2) import the hex file into the pic18f4620.

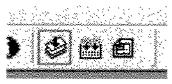


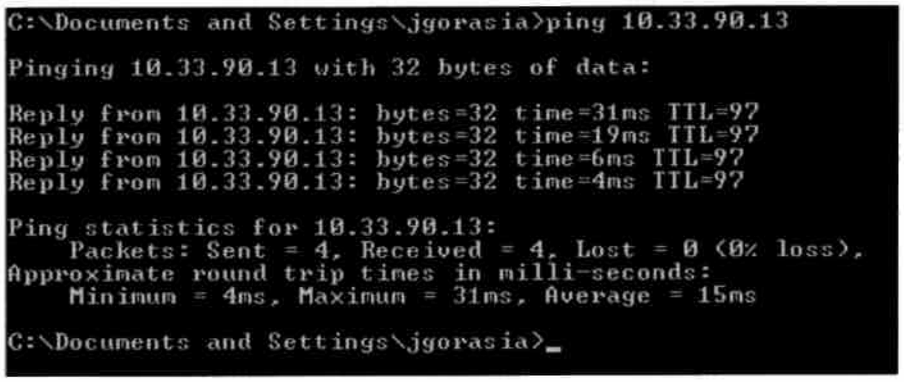
Figure 4.1: The red box is to build the project

4.4 Test connection

Now, plug an Ethernet cable into the RJ11 jack of the circuit, and the other end into a wall socket. Alternatively, use a crossover cable[2] and plug the cable into your computer's LAN port instead.

[1]http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002

[2]You could find this product from Amazon: Belkin Components CAT5E Crossover Cable, Red - 10ft



```
C:\Documents and Settings\jgorasia>ping 10.33.90.13

Pinging 10.33.90.13 with 32 bytes of data:

Reply from 10.33.90.13: bytes=32 time=31ms TTL=97
Reply from 10.33.90.13: bytes=32 time=19ms TTL=97
Reply from 10.33.90.13: bytes=32 time=6ms TTL=97
Reply from 10.33.90.13: bytes=32 time=4ms TTL=97

Ping statistics for 10.33.90.13:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 4ms, Maximum = 31ms, Average = 15ms

C:\Documents and Settings\jgorasia>_
```

Figure 4.2: Results from using the ping utility successfully.

In Windows, go to the command prompt, and use the **ping** utility to determine if the device can respond to requests. By default, the DHCP module is disabled, thus the IP address of the device is 192.168.1.201 . It is not recemmended to use static IP addresses when operating in a local area network[3].

The results from the ping utility should look like below:

4.5 Build webpage binary file

To build the webpage, you need to use the MPFS utility (found under the Tools folder) in order to compress webpages into a suitable size.

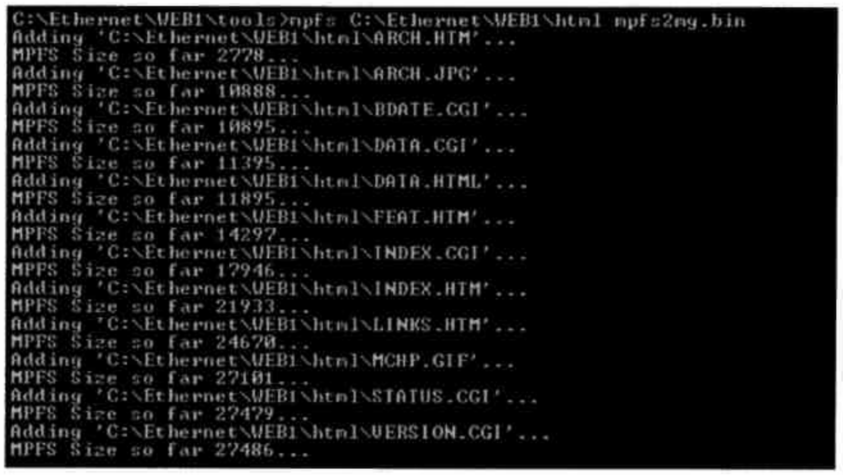
If you selected the MPFS_USE_EEPROM option to store the HTTP documents in an external serial EEPROM, you must upload the file system image by using FTP or via the serial command interface with XMODEM.

First you must create a MPFS binary image of the HTTP server documents, to do so you must use the MPFS.EXE utility passing as arguments the directory where the source documents are located (the example website documents are in the html directory) and the name of the output file that will be later uploaded.

If your design includes the new Microchip 25LC1024 serial EEPROM, you must create the image using the MPFS.EXE /l option (24 bit addressing). Remember also to include the USE_25LC1024 macro in your hardware configuration file to include the appropriate code and settings for this memory device.

Double check that your image file does not exceed the available capacity of the EEPROM memory, the stack reserves the first 64 bytes to save the application configuration information and the rest of the memory is available for the HTTP documents image (which includes a simple File Allocation Table).

[3]Your network might have already allocated that IP address to another device, and you do not really have a way to find out, until things start going wrong



```
C:\Ethernet\WEB1\tools>mpfs C:\Ethernet\WEB1\html mpfs2mg.bin
Adding 'C:\Ethernet\WEB1\html\ARCH.HTM'...
MPFS Size so far 2778...
Adding 'C:\Ethernet\WEB1\html\ARCH.JPG'...
MPFS Size so far 10888...
Adding 'C:\Ethernet\WEB1\html\BDATE.CGI'...
MPFS Size so far 10895...
Adding 'C:\Ethernet\WEB1\html\DATA.CGI'...
MPFS Size so far 11395...
Adding 'C:\Ethernet\WEB1\html\DATA.HTML'...
MPFS Size so far 11895...
Adding 'C:\Ethernet\WEB1\html\FEAT.HTM'...
MPFS Size so far 14297...
Adding 'C:\Ethernet\WEB1\html\INDEX.CGI'...
MPFS Size so far 17946...
Adding 'C:\Ethernet\WEB1\html\INDEX.HTM'...
MPFS Size so far 21933...
Adding 'C:\Ethernet\WEB1\html\LINKS.HTM'...
MPFS Size so far 24670...
Adding 'C:\Ethernet\WEB1\html\MCHP.GIF'...
MPFS Size so far 27101...
Adding 'C:\Ethernet\WEB1\html\STATUS.CGI'...
MPFS Size so far 27479...
Adding 'C:\Ethernet\WEB1\html\VERSION.CGI'...
MPFS Size so far 27486...
```

Figure 4.3: Using the MPFS utility. The utility converts the webpages into a binary file format suitable for storing on an EEPROM

4.6 Upload webpages

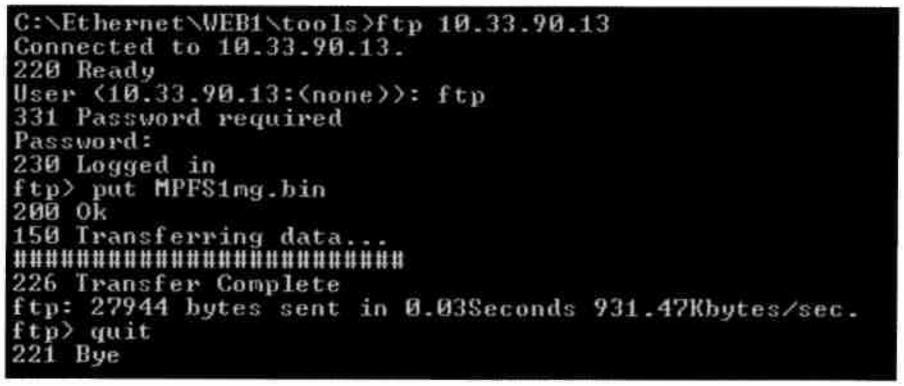
To upload webpages to the device we can use an FTP transfer. FTP does not check for available memory space, thus if you exceed the actual available memory, the code will wraparound and start writing over the previously written FAT and documents.

The WEB1 zip file includes MPFS images generated from the sample HTTP documents located in the html directory. The file mpfsimg.bin is the MPFS binary image with the standard 16 bit addressing (for a 25LC256 or 24LC256/512) and the mpfsimg_l.bin the binary image with 24 bit addressing (for a 25LC1024), both files are located in the current version top directory.

To use transfer the webpages using ftp:

1. First open the command prompt
2. Type: *ftp [device ip address]*
3. You will be prompted for the username and password. By default they are *ftp* and *microchip* respectively.
4. Then type: *put [your binary file's name].*
5. It will then show its progress, then announce "Transfer complete".
6. Type *quit* to exit the program

Below is a screenshot of me performing these steps.

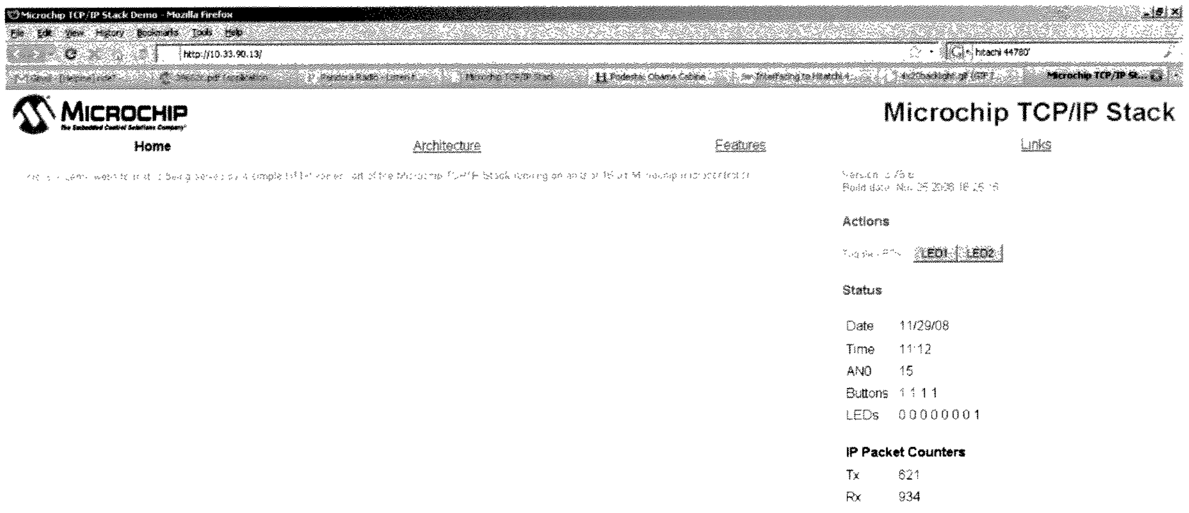


```
C:\Ethernet\WEB1\tools>ftp 10.33.90.13
Connected to 10.33.90.13.
220 Ready
User (10.33.90.13:(none)): ftp
331 Password required
Password:
230 Logged in
ftp> put MPFS1mg.bin
200 Ok
150 Transferring data...
###########################
226 Transfer Complete
ftp: 27944 bytes sent in 0.03Seconds 931.47Kbytes/sec.
ftp> quit
221 Bye
```

Figure 4.4: Using the ftp utility in Windows

4.7 Test webpages

To test whether everything worked, open up a web browser, and type in the IP address of the device. You should see the following. The webpage will not work well on browsers that have Javascript disabled as all the dynamic elements of the site require Javascript. You would still be able to continually refresh the webpage to see the variables update. Alternatively, go to http://[ip address]/index.cgi to see a version that does not require JavaScript. The next chapter will tell how to build such webpages with dynamic content, and well as modify the TCP/IP stack in order to add your own applications.



Status

Date	11/29/08
Time	11:12
AN0	15
Buttons	1 1 1 1
LEDs	0 0 0 0 0 0 0 1

IP Packet Counters

Tx	621
Rx	934

Figure 4.5: The homepage of the WEB1 webserver. This shows the dynamic variables and allows the some interaction with the two input buttons for the LEDs.

Chapter 5

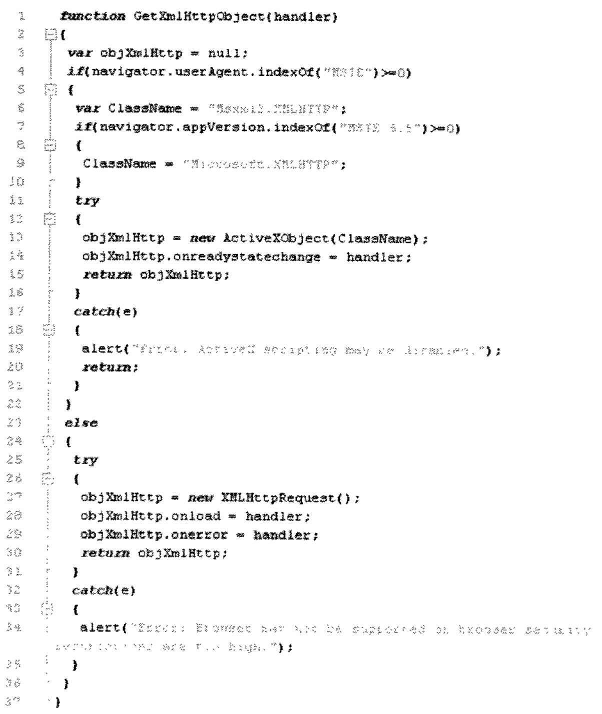
Webpages and modifying the stack

In this chapter, I will cover how to create the webpages appropriate for PIC MCUs. You will need to create a few CGI files in order to perform any interactive task with the webserver. Use the included files as a template.

5.1 Basic AJAX

Since we are creating dynamic webpages, we will need to use AJAX in order to update portions of the webpage, without reloading the whole thing. AJAX stands for Asynchronous Javascipt And XML, a concatenation of two powerful technologies.

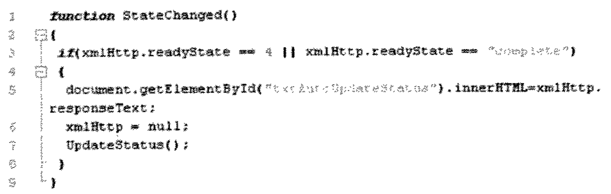
The first step is to add a script to the header of the file to determine the appropriate XMLHttpRequest to use. The different browsers use different techniques, and using this script allows one to create webpages that can function in multiple browsers. XMLHttpRequests are what is used by webpages to perform requests without refreshing the entire page.



```
function GetXmlHttpObject(handler)
{
 var objXmlHttp = null;
 if(navigator.userAgent.indexOf("MSIE")>=0)
 {
  var ClassName = "Msxml2.XMLHTTP";
  if(navigator.appVersion.indexOf("MSIE 5.5")>=0)
  {
   ClassName = "Microsoft.XMLHTTP";
  }
  try
  {
   objXmlHttp = new ActiveXObject(ClassName);
   objXmlHttp.onreadystatechange = handler;
   return objXmlHttp;
  }
  catch(e)
  {
   alert("Error. ActiveX scripting may be disabled.");
   return;
  }
 }
 else
 {
  try
  {
   objXmlHttp = new XMLHttpRequest();
   objXmlHttp.onload = handler;
   objXmlHttp.onerror = handler;
   return objXmlHttp;
  }
  catch(e)
  {
   alert("Error: Browser may not be supported or browser security
restrictions are too high.");
  }
 }
}
```

Figure 5.1: XMLHTTPObject script to determine the appropriate XMLHttpRequest to use.

Then, another script needs to get the html file ready to execute the XMLHttpRequests.
Finally you have these other scripts that do stuff.



```
function StateChanged()
{
 if(xmlHttp.readyState == 4 || xmlHttp.readyState == "complete")
 {
  document.getElementById("txtAutoUpdateStatus").innerHTML=xmlHttp.
responseText;
  xmlHttp = null;
  UpdateStatus();
 }
}
```

Figure 5.2: Set the html file to the right state for AJAX

5.2 Transmit information

The way you transfer information to the web server is using http GET statements. This executes a callback to the HttpExecCmd function (located in main.c). The function will parse the input in the following way:

If the Http argument was : `index.htm?name=Joe&age=25`

- argv[0] – index.htm
- argv[1] – name
- argv[2] – Joe
- argv[3] – age
- argv[4] – 25

The function will first find the appropriate CGI file to execute the commands through. By default, everything reverts to the `index.cgi` page. Next, it will check the number for the command to execute. To simplify coding, the current implementation only allows for single digit commands, thus only 10 commands. This limit is merely as a means to simplify coding, and could be extended. The WEB1 server will respond to two commands:

- `[anypage]?0=0`: which toggles LED1
- `[anypage]?0=1`: which toggles LED2

Modifying this function is pretty easy. In main.c, the names of the callback commands are named (search for CMD_LED1). Just add another command there with an appropriate single digit number. Then, go back to the HttpExecCmd function, look for the switch case statement that chooses commands and another command. You can pretty much put any command you want into this.

5.3 Receive information

To receive information about from the web server, you need to use specific callback requests. The web server processes all CGI files, looking for predefined commands. The format of all commands is %XX, where XX is some hexadecimal number. Below is a list of the commands that are specified in the WEB1 server.

Variable Name	Value (0x00-0xFF)	Function
VAR_LED0	00	
VAR_LED1	01	
VAR_LED2	10	
VAR_LED3	11	
VAR_LED4	12	
VAR_LED5	13	
VAR_LED6	14	
VAR_LED7	15	
VAR_ANAIN_AN0	02	
VAR_DIGIN0	04	
VAR_DIGIN1	0D	
VAR_DIGIN2	0E	
VAR_DIGIN3	0F	
VAR_STACK_VERSION	16	
VAR_STACK_DATE	17	
VAR_IPCNTR_TX	20	
VAR_IPCNTR_RX	21	
VAR_DATE	22	
VAR_TIME	23	

To add more commands, just add another case to the case switch statement that scrolls through the different variable cases. If the variable to be returned is just a single digit, the code is very easy. Just set *val to the variable(look at the code. It will make a whole lot more sense). However, for longer variables you need to use to following construct:

```
*val = variable[(BYTE)ref];
if (variable[(BYTE)ref] == '\0')
  return HTTP_END_OF_VAR;
else if (variable[(BYTE)++ref] == '\0')
  return HTTP_END_OF_VAR;
return ref
```

This construct is flexible enough to handle variable of any length.

5.4 Adding periodic functions

There will be a need to preform periodic function like process an analog signal, or control some lights. These processes should not take too much time, in order not to detract from the operation of the web server. An example of a periodic function is provided in the ProcessIO function in main.c.

Currently, ProcessIO is used to process the input to the analog to digital converter. You could add other processes to the ProcessIO function, or make a new function similar to ProcessIO to handle them.

Chapter 6

TCP/IP: In depth

6.1 Frame Sizes

6.2 MAC Addresses

6.3 Stream Construcion/Deconstruction

6.4 Stream timing

6.5 10 Mb/s Stream contents

Machester encoding

6.6 Auto Negotiation

6.7 Auto Crossover

Chapter 7

Appendix

7.1 Build of materials

Name	Quantity	Digikey Part Number	Unit Price ($)
PIC18F4620 I/P	1	PIC18F4620-I/P-ND	9.93
ENC28J60	1	ENC28J60/SP-ND	3.70
Solderless breadboard	1		
$10k\Omega$ resister	4		