



Building the modified Microchip TCP/IP Stack Version 3.75.6

Contents

1 - Introduction.....	3
2 - Directory structure	5
3 - Configuration.....	6
3.1 - Anatomy of the config.h file	7
3.2 - Anatomy of the hardware configuration files	12
3.3 - Main application source code	17
4 - Building the code image	18
5 - Uploading the HTTP documents	20
6 - Testing the TCP/IP Stack	22
6.1 - Sending ICMP echo requests (ping)	23
6.2 - Using Microchip's Ethernet Discoverer.....	24
6.3 - Using the serial interface	25
6.4 - Uploading the HTTP documents binary image with FTP.....	26
6.5 - Testing the HTTP server	27
7 - The udptest module.....	29
8 – Connecting to the Internet	31

1 - Introduction

This modified version of the TCP/IP stack is based on the original distribution of the Microchip TCP/IP stack v3.75. The main goal for creating this version was to be able to have a consistent and common source code base across different projects and reference designs, more easy to configure, better organized and easier to read and modify.

While most of the code in this version follows the same logic as the original version, some of the differences with previous versions are the directory structure, file names and the way the stack options and hardware configurations are defined.

There are also some fixes and improvements, such as the LCD routines that have been replaced with new code that includes support for a character LCD module using the standard parallel interface or the SPI interface using a Microchip MCP23S08 port extender.

Another addition is for example support for the new Microchip 25LC1024 1Mb serial EEPROM and better code selection to handle 24 and 32 bit addressing.

All the source code is organized in the `src` directory with subdirectories for each of the major modules, such as the code for the TCP/IP stack in the `net` directory, the LCD driver functions in the `lcd` directory, etc.

Each of these subdirectories also contain an `include` subdirectory where the associated header files for each module are located.

All the files went through a clean up process to improve readability and better code organization, some definitions have been moved to other include files, all configuration options have been moved to the main `config.h` and hardware configuration files, all the header and source code files have been reorganized in sections and in an order such as the prototype definitions for every and each function could be removed eliminating a substantial amount of clutter from the source files.

All the *code heritage* or version log history has been preserved but reorganized, all tabs replaced by white spaces to permit using any text viewer or editor without loosing the indentation for easy readability.

Most of the files now follow a common editing style.

There are several modifications based on suggestions, findings and comments from many members of the Microchip User's forum and some ideas taken out from the latest release (v4.02) of the original Microchip TCP/IP stack code.

Note: The current version DOES NOT support the HI-TECH or CCS compilers yet.

Thanks to all members of the Microchip Users Forum that actively participate in the forum making great contributions and providing countless suggestions and ideas.

A zip file containing the latest release of this software and an online version of this document is available at <http://www.ljcv.net/projects/mchptcp3.75/>.

Happy Networking !!!
Jorge Amodio
LJCV Electronics
San Antonio, Texas
USA

2 - Directory structure

For this version all the source and include files have been reorganized in a directory tree where each module has its own subdirectory, all file names have been converted to lower case and stripped of white spaces and many of them renamed.

Below is the directory structure for the current version

```
MCHTCP          Main Directory
|
+- v3.75.X      Current version, MPLAB project files
|
|   +- bin        Binary Images (Intel Hex)
|   +- doc        Documentation, Microchip License Agreement
|   +- html       Sample HTTP server documents
|   +- linker     Linker Scripts for MPLINK C18 & C30
|   +- tools      Utilities (mpfs.exe, etc)
|   +- src        Main source code directory
|
|   +- include    General include headers and configuration files
|   +- net        TCP/IP Modules source code and includes
|   +- lcd        LCD Driver source code and includes
|   +- mpfs      Microchip File System source code and includes
|   +- eeprom    Serial EEPROM drivers source code and includes
|   +- time      RTCC and time support source code and includes
|   +- uart      Serial UART interface source code and includes
```

3 - Configuration

There are at least three files that you will need to edit and modify to reflect your particular hardware configuration and code options, the `config.h` file, the particular *hardware configuration file* for your board and if necessary *the main application source code*.

Once you have the files "tailored" for your project or application you must select the correct processor on MPLAB IDE and if necessary the language toolset that will be used to compile the code.

Remember that some microcontrollers such as the PIC18F452 or PIC18F4520 have limited amount of program memory space to fit all stack modules and leave available space for your application. A recommended pin compatible device with more memory both program and RAM, and more advanced peripherals is the PIC18F4620 or its 28-pin version the PIC18F2620.

The main configuration file and other parts of the code use a special macro that define which is the hardware profile to be included for building the stack, you need to add this macro to the project build options on MPLAB IDE.

For example if you are planning to build the code for the Microchip PICDEM.net board, on the **Projects->Build Options->Project** menu and MPLAB C18 tab you must add the macro `PICDEMNET`.

3.1 - Anatomy of the config.h file

The config.h file is included throughout the code to select particular pieces of code and initialization values based on the particular hardware configuration (more about it later) and TCP/IP stack configuration and options.

The config.h is located in the src/include subdirectory.

The file is organized in different sections. Below is an explanation of each section and some of the configuration options/macros.

The first section defines a string with the current version of the code, this string will be displayed on the LCD if present and available as a variable for the HTTP server in case like the HTTP documents examples you want to show the version on a HTML page.

```
#define VERSION      "3.75.6"           // Firmware version
```

The following group of conditional macros selects which hardware configuration file to include based on a macro that must be defined in the projects options. The software distribution includes several MPLAB IDE project files that have the correct macro defined for each project. Notice that this macro value will be used in other parts of the code such as the main application code to set the appropriate processor configuration bits.

Also each project may have some variants based on the particular hardware configuration, for example if you want to compile the code for the PICNet 1 Board without the SPI LCD drivers you must comment the line that includes the picnet1_spilcd.h file and uncomment the one that includes the picnet1.h.

```
*****  
// Define or include your hardware configuration  
//  
#if defined(PIC10T)  
    #include "include/pic10t.h"  
#elif defined(MINIPIC10T)  
    #include "include/minipic10t.h"  
#elif defined(PICNET1)  
    //    #include "include/picnet1.h"  
    #include "include/picnet1_spilcd.h"  
#elif defined(EIP10)  
    //    #include "include/eip10.h"  
    #include "include/eip10_lcd.h"  
    //    #include "include/eip10_spilcd.h"  
#elif defined(PICDEM2)  
    #include "include/picdem2.h"  
#elif defined(PIC18_NIC28)  
    #include "include/pic18_nic28.h"  
#elif defined(PICDEMNET)  
    #include "include/picdemnet.h"
```

```
#elif defined(PICDEMNET2)
    #include "include/picdemnet2.h"
#elif defined(HPC_EXPLORER)
    #include "include/hpc_explorer.h"
#elif defined(PICWEB1)
    #include "include/picweb1.h"
#elif defined(EXP16_DSPIC33)
    #include "include/exp16_dspic33.h"
#elif defined(EXP16_PIC24H)
    #include "include/exp16_pic24h.h"
#elif defined(EXP16_PIC24F)
    #include "include/exp16_pic24F.h"
#elif defined(PIC24FJ64_NIC28)
//    #include "include/pic24fj64.h"
    #include "include/pic24fj64_lcd.h"
#else
    #error: CFG001: No hardware configuration defined
#endif
```

The following group of macros set the default MAC Address and IP configuration. Observe that for the MAC and IP addresses the values are separated by commas.

```
*****  
// TCP/IP Config  
//  
#define DEFAULT_MAC_ADDRESS { 0x00, 0x04, 0xa3, 0x00, 0x02, 0x00 }  
#define DEFAULT_IP_ADDRESS { 192, 168, 1, 201 }  
#define DEFAULT_NETMASK { 255, 255, 255, 255 }  
#define DEFAULT_GATEWAY { 192, 168, 1, 1 }  
#define DEFAULT_NS1 { 192, 168, 1, 1 }  
#define SNTP_SERVER_IP { 192, 43, 244, 18 } // time.nist.gov  
  
#if defined(PICDEM2)  
    #define DEFAULT_NETBIOS_NAME "PICDEM2"  
#elif defined(PIC10T) || defined(MINIPIC10T)  
    #define DEFAULT_NETBIOS_NAME "PIC10T"  
#elif defined(EIP10)  
    #define DEFAULT_NETBIOS_NAME "EIP10"  
#elif defined(PICNET1)  
    #define DEFAULT_NETBIOS_NAME "PICNET1"  
#else  
    #define DEFAULT_NETBIOS_NAME "MCHPBOARD"  
#endif
```

The following group of macros select which modules of the TCP/IP stack will be included, there is no need to include the UDP and TCP modules since they will be automatically include based on the requirements of each of the modules listed below.

To include a particular module just uncomment the corresponding line. Be aware that some modules like the TCP Client Example will automatically include the DNS module regardless if the line in this section is commented or not.

```
*****  
// Stack Modules  
//  
#define STACK_USE_ICMP          // ICMP reply (ping) module  
#define STACK_USE_HTTP_SERVER    // HTTP server
```

```
//#define STACK_USE_IP_GLEANING          // Obtain IP address via IP Gleaning
//#define STACK_USE_DHCP                 // DHCP client
#define STACK_USE_FTP_SERVER            // FTP server
//#define STACK_USE_TCP_EXAMPLE1         // HTTP client example in tcp_client_ex1.c
#define STACK_USE_ANNOUNCE              // Ethernet Device Discoverer server/client
#define STACK_USE_SNTP                  // SNTP client
//#define STACK_USE_DNS                 // DNS client
#define STACK_USE_NBNS                 // NetBIOS Name Service Server
//#define STACK_USE_UDPTEST             // Enable UDP Test code
```

The following macros define various parameters used by the Timer tick counter and the Time and SNTP modules.

```
*****  
// Some time related definitions  
//  
// Define Ticks per second for the tick manager  
//  
#define TICKS_PER_SECOND      (100)    // 10ms  
  
#define USE_TIME                // Include time routines  
#define TIME_SOURCE_TICK        // Time source is Timer0 Tick counter  
//#define TIME_SOURCE_32KTIMER  
  
#define TZ_OFFSET               (-5)     // Time Zone offset (negative west of GMT)  
#define SNTP_UPDATE_SECS        (43200)  // SNTP update interval in seconds (12hrs)
```

The next group of macros defines various options and parameters for the TCP/IP stack and main application, each of the macros include a descriptive comment of its use and accepted values.

```
*****  
// Miscellaneous Stack Options  
//  
// Define the Baud Rate for the RS-232 Serial Interface  
  
#define BAUD_RATE (19200)           // bps  
  
// Include the code for the configuration menu via the RS-232 serial interface  
//  
#define ENABLE_BUTTON0_CONFIG  
  
// Include User process code in main.c  
//  
//#define ENABLE_USER_PROCESS  
  
// Define Username and Password for the FTP Server  
//  
#define FTP_USERNAME "ftp"  
#define FTP_PASSWORD "microchip"  
#define FTP_USER_NAME_LEN (10)  
  
// Define default TCP port for HTTP server  
//  
#define HTTP_PORT (80)  
  
// Enable Hardware assisted IP checksum calculation  
// Some Ethernet controllers such as the ENC28J60 include a feature that
```

```
// permits to use the DMA module in checksum mode to assist in the calculation
// of checksum values. The current silicon revisions of the ENC28J60 (B1-B5)
// have a bug that may produce incoming packet loss if this option is enabled
// and a packet is received when the DMA module is in checksum mode.
// If this option is commented the code will set the STACK_USE_SW_CKSUM and
// include the appropriate code to calculate the checksums by software.
//
//#define STACK_USE_HW_CKSUM

// Uncomment following line if this stack will be used in client mode.
// In client mode, some functions specific to client operation are enabled.
// This option will be enabled by default if you include any stack modules
// in the "Stack Modules" section above that require the stack to operate in
// this particular mode.
//
//#define STACK_CLIENT_MODE

// Comment following line if TCP state machine should wait for acknowledgement
// from the remote host before transmitting another packet.
// Commenting following line may reduce throughput.
//
//#define TCP_NO_WAIT_FOR_ACK

// This macro is specific to the Microchip Ethernet controllers.
// If a different Ethernet controller is used, this define is not used.
// If a Microchip controller is used and a self memory test should be done
// when the MACInit() function is called, uncomment this line.
// The test requires ~512 bytes of program memory.
//
//#define MAC_POWER_ON_TEST

// This macro is specific to the Microchip Ethernet controllers.
// If a different Ethernet controller is used, this define is not used.
// Ideally, when MAC_FILTER_BROADCASTS is defined, all broadcast packets that
// are received would be discarded by the hardware, except for ARP requests for
// our IP address. This could be accomplished by filtering all broadcasts, but
// allowing the ARPs using the pattern match filter.
// The code for this feature has been partially implemented, but it is not
// complete nor tested, so this option should remain unused in this stack
// version.
//
//#define MAC_FILTER_BROADCASTS

// Maximum number of TCP sockets allowed
// Note that each TCP socket consumes 38 bytes of RAM.
//
#define MAX_SOCKETS          (6u)

// Maximum number of TCP sockets allowed
//
#define MAX_UDP_SOCKETS      (4u)

// Maximum numbers of simultaneous HTTP connections allowed.
// Each connection consumes 10 bytes of RAM.
//
#define MAX_HTTP_CONNECTIONS (3u)
```

The TCP/IP stack includes a simple HTTP server for which the documents must be stored either in an external memory or included as part of the code in program memory. Uncomment the appropriate line based on your hardware configuration.

```
*****  
// Storage options  
//  
#define MPFS_USE_EEPROM           // Use external serial EEPROM  
//#define MPFS_USE_PGRM            // Use Program Memory
```

The last section of the config.h file includes a large list of conditional macros that will validate the configuration files and set specific parameters based on the hardware configuration and previously selected options, such as for example the speed and page size for the serial EEPROM, etc.

Normally you should not need to change anything below the following comment lines.

```
/////////  
/////////  
///      UNLESS NECESSARY AND YOU REALLY KNOW WHAT YOU ARE DOING YOU      ///  
///      SHOULD NOT NEED TO CHANGE ANY OF THE LINES BELOW                 ///  
/////////  
/////////
```

3.2 - Anatomy of the hardware configuration files

This new version of the modified Microchip TCP/IP stack includes now a separate header file for each development board or reference design file that defines the particular hardware configuration, options and initialization values for various registers.

The configuration file must include all the required macro definitions for the particular drivers for each device such as the Ethernet controller, LCD display or serial EEPROM memory. If something is missing you most probably will get an error during compilation time.

The current distribution includes several hardware configuration files for different development boards or designs and different hardware options.

You can use these files as a reference to build the configuration file for your specific design.

The main config.h file will select which hardware configuration file to include in the project based on the macro previously defined in MPLAB IDE.

All hardware configuration files are located in the general src/include directory.

The hardware configuration files included in this distribution are:

- eip10.h	LJCV Electronics eIP-10 Board
- eip10_lcd.h	LJCV Electronics eIP-10 Board + character LCD module
- eip10_spilcd.h	LJCV Electronics eIP-10 Board + character LCD module with SPI interface and MCP23S08 port extender
- exp16_dspic33.h	Microchip Explorer 16 Board with dsPIC33FJ256GP710 PIM and Microchip Ethernet PICTail+
- exp16_pic24f.h	Microchip Explorer 16 Board with PIC24FJ128GA010 PIM and Microchip Ethernet PICTail+
- exp16_pic24h.h	Microchip Explorer 16 Board with PIC24HJ256GP610 PIM and Microchip Ethernet PICTail+
- hpc_explorer.h	Microchip HPC Explorer with PIC18F8722 + Ethernet PICTail
- minipic10t.h	Jorge Amodio mini PIC10T reference project
- pic10t.h	Jorge Amodio PIC10T reference project
- pic24fj64.h	Jorge Amodio PIC24FJ64GA002 reference project
- pic24fj64_lcd.h	Jorge Amodio PIC24FJ64GA002 reference project + character LCD module with SPI interface and MCP23S08 port extender
- picdem2.h	Microchip PICDEM 2+ (mod) + LJCV Electronics nic28 NIC
- picdemnet.h	Microchip PICDEM.net Board with RTL8019AS controller
- picdemnet2.h	Microchip PICDEM.net 2 Board with PIC18F87J60
- pic18_nic28.h	Connecting a 28 pin PIC18F to LJCV Electronics nic28 NIC
- picnet1.h	LJCV Electronics PICNet 1 Development Board
- picnet1_spilcd.h	LJCV Electronics PICNet 1 Board + character LCD module with SPI interface and MCP23S08 port extender
- picweb1.h	Celeritous PICWEB1 Module with PIC18F67J60

All the hardware configuration files include references about where to obtain additional information about each board and detailed schematics.

Example description for the PICNet 1 Board with SPI LCD driver hardware configuration file (`picnet1_spilcd.h`):

First you must include the appropriate header file for the microcontroller family.

```
// Include the appropriate header file for the microcontroller family
#include <p18cxx.h>
```

Next you must define the CPU clock frequency in Hertz. Notice that this is the actual CPU clock and not the frequency of the crystal or oscillator source used, the value depends on the oscillator configuration bits and registers. For example if your hardware design uses a PIC18F4620 with a 10MHz crystal or ceramic resonator and you set the HS-PLL the actual clock frequency will be 40 MHz.

`TCY_CLOCK` defines the actual instruction cycle frequency as a function of the CPU clock, for PIC18 devices one instruction takes four clock cycles to execute then `TCY_CLOCK=CPU_CLOCK/4`, on PIC24 and dsPIC33 devices each instruction takes two clock cycles, then `TCY_CLOCK=CPU_CLOCK/2`.

These two parameters are use throughout the code to provide delays and clock configuration for peripherals such as the MSSP for SPI or I2C communications, the USART module for the RS232 interface, etc.

```
*****  
// Define Microcontroller Clock Frequency in Hertz  
//  
#define CPU_CLOCK      (40000000)  
#define TCY_CLOCK       (CPU_CLOCK/4)
```

The following section includes for each available Input/Output port, pin by pin a description of what each pin is used for and its initial direction and state.

The macros defining initial direction and state will be used by the initialization routine in the main application to properly set the direction and initial state of each I/O port.

Below is an example of the definitions for PORTA.

```
//*****
// GPIO Ports assignments, configuration and initial default value
// Define the direction for each Input/Output pin (0-Output, 1-Input) and
// the initial state at the application startup
//
// PORTA Direction and initial status
// +----- n/a OSC1
// |+----- n/a OSC2
// ||+----- RA5 = LED Z
// |||+----- RA4 = LED Y
// ||||+----- RA3 = LED X
// |||||+----- RA2 = MCP23S08 CS
// |||||+----- RA1 = n/c
// |||||+----- RA0 = n/c
#define INIT_TRISA  (0b00000000)
#define INIT_PORTA  (0b00000100)
```

There are other registers that may require an initial or configuration value, such as the configuration of the microcontroller analog features and input ports. The next group of macros defines the initial values for such registers.

```
//*****
// Initialization values for various registers
//
#define INIT_ADCON0 (0b00000000)      // ADON=0, Channel 0
#define INIT_ADCON1 (0b00001111)      // No analog inputs
#define INIT_ADCON2 (0b10111110)      // Right justify, Fosc/64 (~21.0kHz)
```

The TCP/IP stack code uses a series of name macros to refer to specific items such as LEDs, pushbuttons or control signals associated with I/O ports. The following group of macros “map” the logic name of the items to the corresponding I/O port and pin.

```
//*****
// Available LEDs and switches macro name definitions for application use
//
#define LED0_IO          (LATAbits.LATA3)
#define LED1_IO          (LATAbits.LATA4)
#define LED2_IO          (LATAbits.LATA5)
#define LED3_IO          (LATAbits.LATA5) // No LED3 map to LED2
#define LED4_IO          (LATAbits.LATA5) // No LED4 map to LED2
#define LED5_IO          (LATAbits.LATA5) // No LED5 map to LED2
#define LED6_IO          (LATAbits.LATA5) // No LED6 map to LED2
#define LED7_IO          (LATAbits.LATA5) // No LED7 map to LED2

#define BUTTON0_IO        (PORTBbits.RB0)
#define BUTTON1_IO        (PORTBbits.RB1)
#define BUTTON2_IO        (PORTBbits.RB1) // No BUTTON2 map to BUTTON1
#define BUTTON3_IO        (PORTBbits.RB1) // No BUTTON3 map to BUTTON1
```

Each driver module requires specific macros to be defined, which command the compiler to include the appropriate code and set specific options. The next three sections of the file show various macros used to include the LCD, Ethernet and serial EEPROM drivers and some of their options and control signal mappings.

```
*****  

// LCD Module features and configuration  

// For this particular project the PICNet 1 has a character mode LCD  

// driven via the SPI interface using a MCP23S08 I/O port extender  

//  

#define USE_LCD  

#define USE_CM_LCD           // Include Character Mode LCD Driver  

#define LCD_USE_BUFFER        // Enable local RAM LCD Buffer  

#define LCD_USE_CGCHARS       // Enable Custom Characters support  

#define LCD_USE_SPI  

#define LCD_4BIT_IFACE  

#define LCD_ROWS 2  

#define LCD_COLS 16  

#define USE_MCP23S08  

#define PORTX_CS_IO           (LATAbits.LATA2)  

#define PORTX_SPI_IF           (PIR1bits.SSPIF)  

#define PORTX_SSPBUF           (SSPBUF)  

#define PORTX_SPISTAT          (SSPSTAT)  

#define PORTX_SPICON1          (SSPCON1)  

#define PORTX_ADDRESS           (0x00)  

#define PORTX_SPICON1_CFG      (0x20)  

#define PORTX_SPISTAT_CFG      (0x40)  

//#define PORTX_SAVE_SPI_CFG  

*****  

// Definitions for ENC28J60 Ethernet Controller interface  

//  

#define USE_ENC28J60  

#define ENC_CS_IO                (LATBbits.LATB3)  

#define ENC_SPI_IF                (PIR1bits.SSPIF)  

#define ENC_SSPBUF                (SSPBUF)  

#define ENC_SPISTAT               (SSPSTAT)  

#define ENC_SPISTATbits           (SSPSTATbits)  

#define ENC_SPICON1                (SSPCON1)  

#define ENC_SPICON1bits            (SSPCON1bits)  

#define ENC_SPICON1_CFG           (0x20)    // SPI master, SCK=Fosc/4, idle low  

#define ENC_SPISTAT_CFG           (0x40)    // Tx from active to idle clock  

                                         // Rx sample at middle  

*****  

// Definitions for 25LC256 or 25LC1024 Serial EEPROM interface  

//  

//#define USE_25LC256  

#define USE_25LC1024  

#define EEPROM_CS_IO              (LATBbits.LATB4)  

#define EEPROM_SPI_IF             (PIR1bits.SSPIF)  

#define EEPROM_SSPBUF             (SSPBUF)  

#define EEPROM_SPICON1            (SSPCON1)  

#define EEPROM_SPISTAT             (SSPSTAT)  

#define EEPROM_SPISTATbits         (SSPSTATbits)  

#define EEPROM_SPICON1_CFG        (0x20)  

#define EEPROM_SPISTAT_CFG        (0x40)
```

For devices with more than one UART peripheral module you can specify which one the TCP/IP stack will use as the default interface for several routines using the UART by defining the `UART_NO` value macro with the values 1 or 2. For example the Microchip Explorer 16 board uses the second UART, then on the hardware configuration file you will see a section such as:

```
//*****
// Select UART Module Number for RS232 Interface
// Valid values are 1 for UART1 and 2 for UART2
//
#define UART_NO 2
```

3.3 - Main application source code

The main application code is in the `main.c` file. This file also includes the MPLAB IDE directives for setting the configuration bits for the processor. Depending on your particular hardware configuration and options you may have to modify or add the correct directives to set the configuration bits consistent with your project.

The `main.c` file also includes the callback functions for the HTTP server to perform specific actions and provide variable replacement values for dynamic pages.

If you need to add specific code for your application remember that the stack code was written as a cooperative multitasking “multi-level” state machine with a continuous execution loop. Your code has to take in account this context and “cooperate” with the rest of the code and preserving the continuous execution loop.

4 - Building the code image

To build the code image you must have installed Microchip's MPLAB IDE (version 7.61 or later), MPLAB C18 Compiler (version 3.12 or later) if you are using a PIC18 part or MPLAB C30 Compiler (version 3.01 or later) if you are using a PIC24 or dsPIC part.

After you are done with the configuration of the stack you must include in your MPLAB IDE project the correct linker script file (if a different processor than the existing one in the project will be used).

For MPLAB C18 you have to use the **Overlay** storage class and **Large data model** (all RAM banks). These two options are selected in the **Projects->Build Options->Project** menu and MPLAB C18 tab, Categories **General** and **Memory Model**.

For 16-bit devices, certain parts of the code may use the UART interface as *stdout* for functions like *printf()*, this feature requires setting some heap memory, for example 100 bytes, this option is set in the **Projects->Build Options->Project** menu and MPLAB LINK30 tab.

In this version the header files for the main code and different modules are distributed in separate include directories, then for both compilers it is necessary to include those directories in the "search path". The directories you must include in the search path are "src", "../src" and "../../src". This is done in the **Projects->Build Options->Project** menu, **Directories** tab and "**Show directories for: Include Serach Path**".

The dialog window should look like the one shown in Figure 1 below.

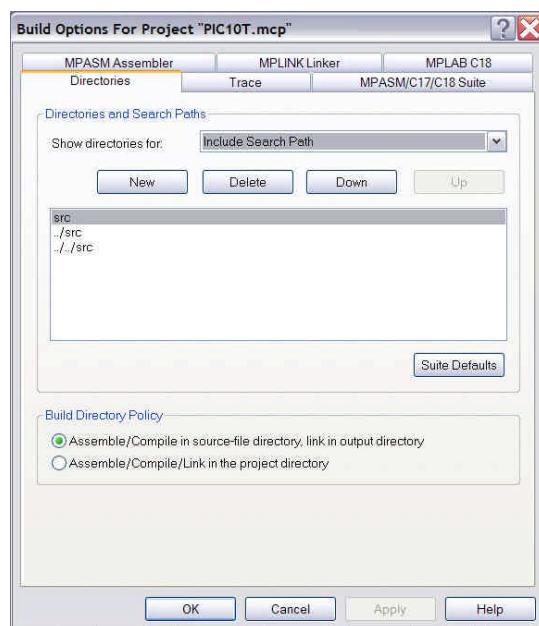


Figure 1

The linker directory includes several script files for various processors compatible with the boards/designs included in this version.

If in the `config.h` file you enabled the HTTP server module and storage on program memory (`MPFS_USE_PGRM`), previously to build the image, you must first generate a "C language image" source file using the `MPFS.EXE` utility using the `/c` option, move the output C file to the main source directory, include it in the project Source file list and edit it to replace the line that has the `#include "...\\Include\\Compiler.h"` by `#include "include/config.h"`.

The current distribution includes a C source code image of the sample website in the file `mpfsimg.c` in the main source directory.

To build your project just use the make or build functions on MPLAB IDE.

If there are some missing configuration options or errors the compiler will stop and show the error on the output screen.

If you use any of the existing MPLAB IDE project files to build the code image, all the project options are pre-set and all objects will be stored under the `obj` subdirectory for each module and the binary files, cof and map files will be stored in the `bin` directory.

If you try to open any of the project files with an MPLAB IDE version prior to 7.61 you will most probably get an error due the format change in the MPLAB workspace files, to avoid this error and be able to open the project just remove the corresponding `".mcw"` file from the main directory.

The project files also assume that the C18 compiler is installed in the "`C:\\MCC18`" directory and the C30 compiler in the "`C:\\MPLAB C30`", if your install path is different you will need to change the language tool suite directories accordingly.

Use your preferred PIC programmer to download the image to the microcontroller program memory.

While it is recommended to use the MPLAB IDE, C18 and C30 versions mentioned before, the code should compile with previous versions.

MPLAB IDE must not be prior to 7.41 to support the include search path, C18 not prior to 3.02 and C30 3.00. Be aware that some device specific features may not be present or supported in earlier versions of the tools and the latest releases include fixes associated with several devices that can be used to run the TCP/IP Stack.

5 - Uploading the HTTP documents

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.

Figure 2 below shows a command prompt window after generating the MPFS image.

```
C:\LJCU_SRC\MCHPTCP>cd v3.75.6
C:\LJCU_SRC\MCHPTCP\>tools\mpfs /l html mpfsimg_1.bin
Adding 'html\ARCH.HTM'...
MPFS Size so far 2738...
Adding 'html\ARCH.JPG'...
MPFS Size so far 10848...
Adding 'html\BDATE.CGI'...
MPFS Size so far 10855...
Adding 'html\FEAT.HTM'...
MPFS Size so far 13257...
Adding 'html\INDEX.CGI'...
MPFS Size so far 16906...
Adding 'html\INDEX.HTM'...
MPFS Size so far 22298...
Adding 'html\LINKS.HTM'...
MPFS Size so far 25035...
Adding 'html\MCHP.GIF'...
MPFS Size so far 27466...
Adding 'html\STATUS.CGI'...
MPFS Size so far 27844...
Adding 'html\VERSION.CGI'...
MPFS Size so far 27851...
C:\LJCU_SRC\MCHPTCP\>
```

Figure 2

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).

Nor FTP or the XMODEM uploading options check for available memory space, if you exceed the actual available memory, the code will wraparound and start writing over the previously written FAT and documents.

The current distribution 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_1.bin` the binary image with 24 bit addressing (for a 25LC1024), both files are located in the current version top directory.

6 - Testing the TCP/IP Stack

After building the code and programming the microcontroller, you may have to perform a Power-On-Reset (power cycle) if you changed the microcontroller's configuration bits since the microcontroller only loads them after a Power-On-Reset and not after a normal Reset.

If everything is correct the LED connected to the pin mapped to `LED0_IO` will start blinking about once a second, and if you have a LCD module, it will show a series of message screens that will include an initial screen with the software version number, and then after a small delay the current time (if you compiled the time and SNTP modules using the `USE_TIME` and `STACK_USE_SNTP` options in the config.h file) and the values for the IP Transmit and Receive packet counters.

For designs or boards with a two line LCD module, the LCD will cycle between two screens showing for about 5 seconds the current time (or version number if time support has not been included) and then for about 10 seconds the IP packet counters.

If you included the time and SNTP module, until the unit is able to communicate with the NTP server to obtain the reference clock you will see the legend “---/---/--- ---:---” on the display indicating that the current clock is not valid, and as soon the unit successfully receives a time update from the NTP server the display will start showing the current date and time.

Figures 3 and 4 below shows an example of the different messages you will see on the LCD for both two and four lines LCD modules.

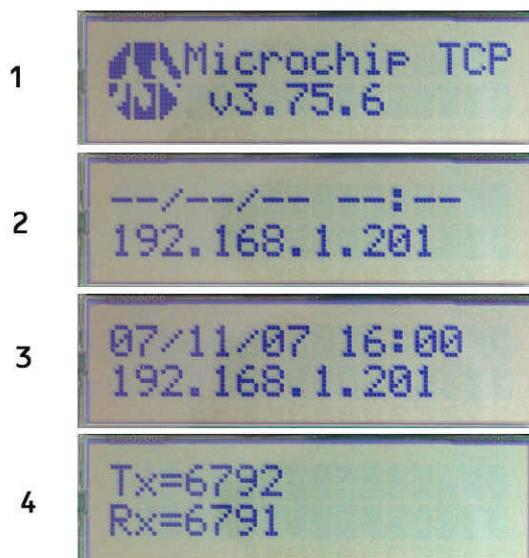


Figure 3

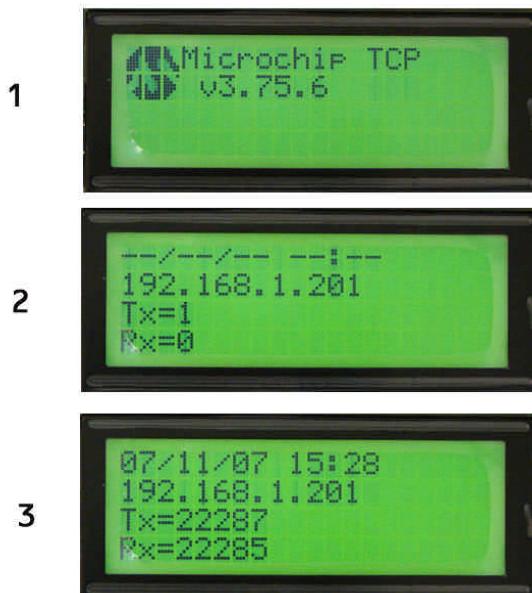


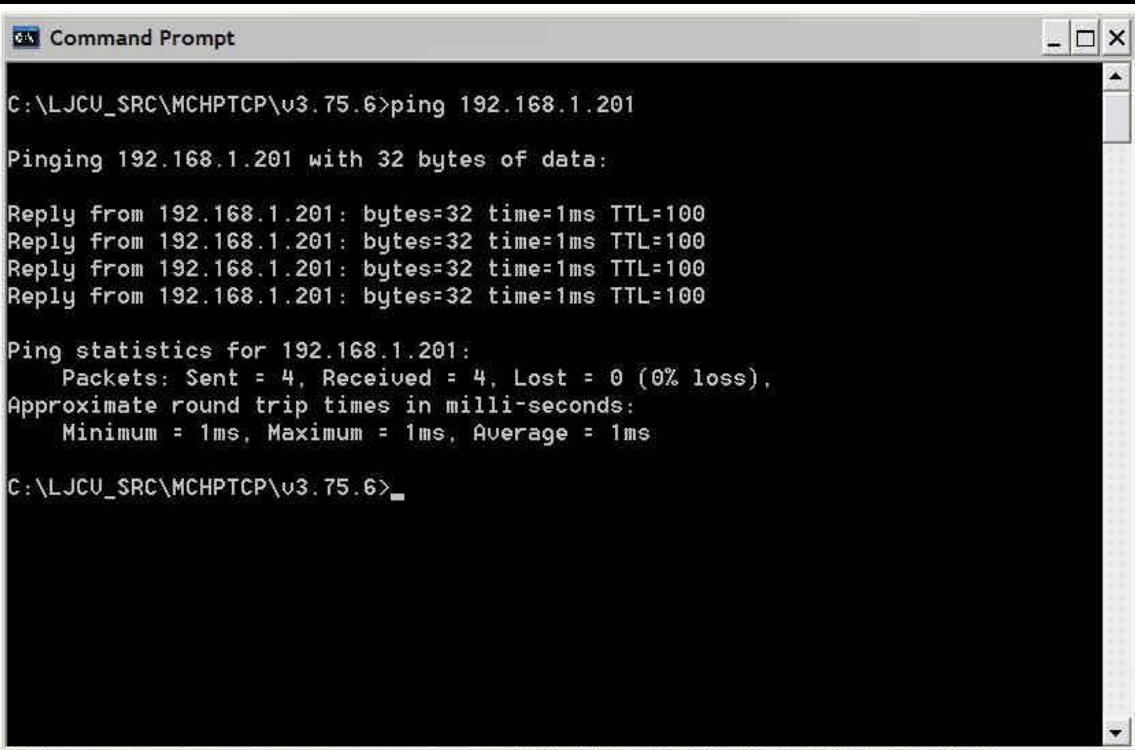
Figure 4

Also if you have connected serial interface and a terminal program (the stack default configuration for the serial interface is 19,200 bps, 8 bits, 1 stop bit, no parity) you will see the current IP address on the terminal screen.

6.1 - Sending ICMP echo requests (ping)

The most popular program today available in almost any operating system to test if you can reach a remote IP host is “ping”, which sends an ICMP message packet to a remote host requesting an “echo reply”. The stack (unless it was disabled in the configuration file) includes a basic ICMP module that will reply to an “echo request”. Be aware that the current version only supports ICMP requests with a payload no larger than 32 bytes, then in some operating systems (Windows default is 32 bytes) you will need to specify the packet size.

Figure 5 shows the command screen after running a ping to a PICNet 1 board running the TCP/IP Stack.



```
C:\LJCU_SRC\MCHPTCP\v3.75.6>ping 192.168.1.201
Pinging 192.168.1.201 with 32 bytes of data:
Reply from 192.168.1.201: bytes=32 time=1ms TTL=100

Ping statistics for 192.168.1.201:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 1ms, Average = 1ms

C:\LJCU_SRC\MCHPTCP\v3.75.6>
```

Figure 5

6.2 - Using Microchip's Ethernet Discoverer

One of the utilities from Microchip included in the distribution is the “Ethernet Device Discoverer”.

If you compiled the code with the `STACK_USE_ANNOUNCE` macro defined, the stack includes a simple UDP daemon that listens to the `ANNOUNCE_PORT` (defined as 30303 in the code) for a discovery request.

If the request is received, the daemon will send back to the requesting host the current NetBIOS hostname, MAC and IP addresses.

Just run the executable `MCHPDiscover.exe` available in the tools directory and you will see a windows screen as in Figure 6 below.

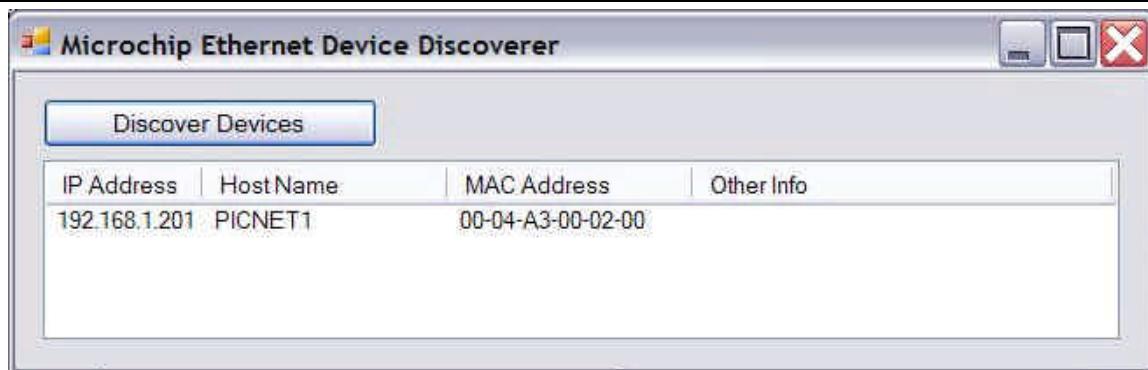


Figure 6

6.3 - Using the serial interface

When the TCP/IP Stack code starts to run after Reset, the code will send the current IP address via the serial interface.

If you compiled the stack code with the `ENABLE_BUTTON0_CONFIG` option, when you reset the microcontroller and hold low for an instant the I/O pin mapped to `BUTTON0_IO` the code will run the configuration routine via the serial interface.

Note about configuration information stored in EEPROM:

If you use an external serial EEPROM, the current stack code will reserve the first 64 bytes to write the configuration information, such as IP address, netmask, etc.

When you run the stack code for the very first time the default configuration hard coded in program memory will be written to the serial EEPROM. The next time the stack code starts will load the previously stored configuration from the serial EEPROM.

If you change the default hard coded configuration in the `config.h` file, you must invalidate the previously stored configuration in the serial EEPROM.

To do so hold low the I/O port mapped to `BUTTON0` for at least 4 seconds after reset or while in the serial configuration menu, the old configuration in the serial EEPROM will be marked as invalid, and the serial interface will show a message, the LDEs will turn on for few seconds and the microcontroller will reset automatically.

After Reset the code will write the new default configuration into the serial EEPROM.

Note that the `BUTTON0` option to rewrite the configuration in the serial EEPROM and the configuration menu via the serial interface will be included in the code only if you define the `ENABLE_BUTTON0_CONFIG` macro in the `config.h` file.

(The stack default configuration for the serial interface is 19,200 bps, 8 bits, 1 stop bit, no parity).

Figure 7 shows the terminal screen after forcing a configuration change.

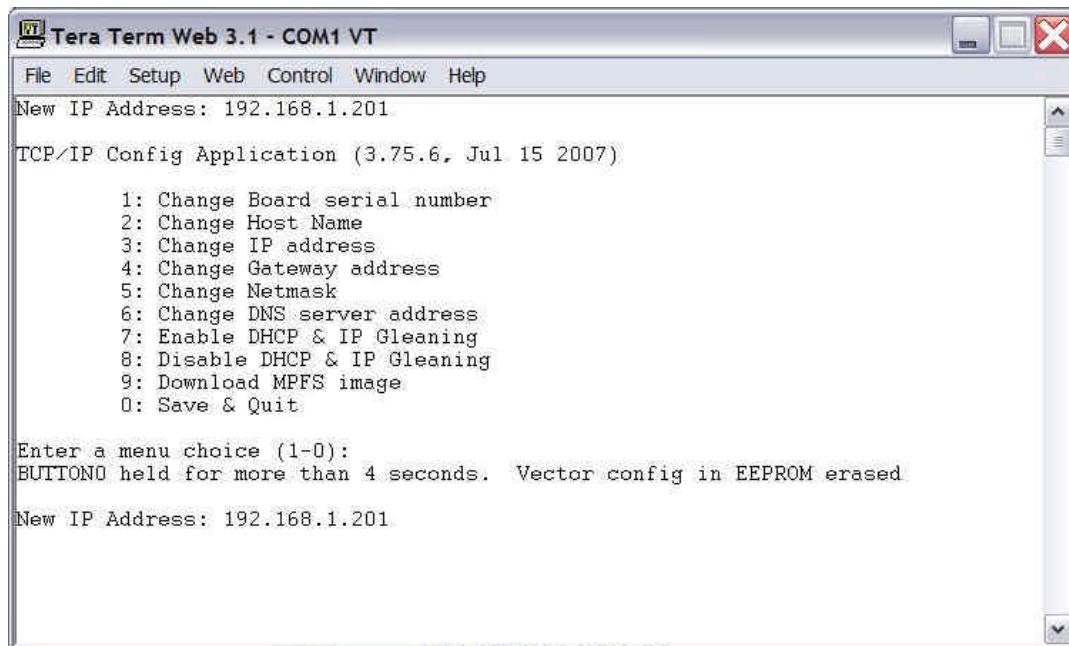


Figure 7

6.4 - Uploading the HTTP documents binary image with FTP

To upload your image to memory via FTP just use the FTP command available in any Windows or Unix machine, enter the username (the default is "ftp") and the password (the default is "microchip") and then execute the "put" command providing the file name of your image file.

Figure 8 shows the command prompt window after transferring the MPFS binary image using FTP.

```
C:\LJCU_SRC\MCHPTCP\v3.75.6>ftp 192.168.1.201
Connected to 192.168.1.201.
220 Ready
User (192.168.1.201:(none)): ftp
331 Password required
Password:
230 Logged in
ftp> put mpfsimg.bin
200 Ok
150 Transferring data...
#####
226 Transfer Complete
ftp: 27944 bytes sent in 0.00Seconds 27944000.00Kbytes/sec.
ftp> quit
221 Bye

C:\LJCU_SRC\MCHPTCP\v3.75.6>
```

Figure 8

6.5 - Testing the HTTP server

If you included the HTTP server in TCP/IP Stack code image using any web browser you can connect to the server using the board IP address or optionally if you included the NetBIOS Name Service module using the assigned hostname with an URL such as <http://PICNET1/>.

The browser window will show a screen similar to the image shown in Figure 9.

Notice that the home page includes AJAX code to continuously update the status of several variables shown on the page such as the IP packet counters or the LEDs status. The sample HTTP documents includes a variant of the home page without the AJAX code but including the variables, the whole page will reload automatically every few seconds and the variables will show the updated value.

The screenshot shows a Mozilla Firefox browser window with the title "Microchip TCP/IP Stack Demo - Mozilla Firefox". The address bar displays "http://192.168.1.201/". The page content is titled "Microchip TCP/IP Stack" and includes the Microchip logo. It features a navigation menu with "Home", "Architecture", "Features", and "Links". The main content area contains several paragraphs of text about the stack's capabilities, including support for different configurations and Ethernet controllers. It also mentions a "Links" page for additional information and reference designs. On the right side, there are sections for "Actions" (with buttons for "Toggle LEDs", "LED1", and "LED2"), "Status" (showing Date: 07/15/07, Time: 17:40, AN0: 2, Buttons: 1111, LEDs: 00000000), and "IP Packet Counters" (Tx: 1894, Rx: 1894). A "Done" button is located at the bottom left of the content area.

Figure 9

7 - The udptest module

This version of the modified TCP/IP Stack distribution includes the udptest module that implements a throughput test using UDP.

To include this module you have to define the `STACK_USE_UDPTEST` macro in the `config.h` file.

Every time that the `UDPTest()` function is called it will send over the network a UDP datagram to a broadcast address or to a specific target address defined in the `udptest.h` header file (`UDP_TST_TARGET_MAC` and `UDP_TST_TARGET_IP`). Datagrams are sent to the UDP port defined by `UDP_TST_TARGET_PORT`.

Datagram size is determined by the `MAX_UDP_DATA` value defined in the code, and all data bytes will be filled with the value `0x41` (A).

If you will be using a specific host address as a target is recommended that on the target host you run a process able to listen on the UDP target port, if there is no process listening to the port depending on the TCP/IP implementation you may start getting back ICMP messages reporting that the service is not available.

In Unix you can use `nc` or `netcat` for this purpose.

WARNING: While the number of packets per second and traffic volume this function generates is not very high, if the target address is configured as a broadcast address, depending on the configuration of your network or other network devices this function can produce a broadcast storm or other undesirable network problems. Use for testing purposes only.

The table in Figure 10 shows the results obtained by running the udptest module in different hardware implementations, with different microcontrollers and clock speeds.

The image code was compiled only with the ICMP and udptest modules and underlying protocols (IP and UDP) and the LCD drivers if the hardware platform included a display.

The bits per second and packets per second metrics were obtained from the interface statistics of a Cisco Catalyst switch after running the test for more than 30 minutes for each tested device and configuration, the target was a dual processor Pentium server running RedHat Linux.

Microchip TCP/IP Stack

Version 3.75.5b

UDP Test Metrics 5/19/07

Board or Reference Design	MCU	CPU CLOCK	Memory		UDP Test	
			Program	RAM	bps	pps
eIP10	PIC18F2620	40	6446	779	399000	86
eIP10 + LCD	PIC18F2620	40	7159	869	391000	84
eIP10 + SPI LCD	PIC18F2620	40	7225	875	390000	84
Explorer 16	dsPIC33FJ256GP710	80	6023	278	1642000	354
Explorer 16	PIC24HJ256GP610	80	6023	278	1642000	354
PIC10T	PIC18F4620	40	7230	832	394000	85
PICDEM.net	PIC18F4620	19.66	6612	750	156000	19
PICDEM2+nic28	PIC18F4620	16	7160	821	160000	34
PICDEM2+nic28	PIC18F4620	25	7160	821	249000	54
PICDEM2+nic28	PIC18F4620	40	7162	821	393000	85
PICNet1	PIC18F4620	25	6517	788	251000	54
PICNet1	PIC18F4620	40	6519	788	397000	85
PICNet1 + SPI LCD	PIC18F4620	25	7293	836	248000	53
PICNet1 + SPI LCD	PIC18F4620	40	7295	836	392000	84

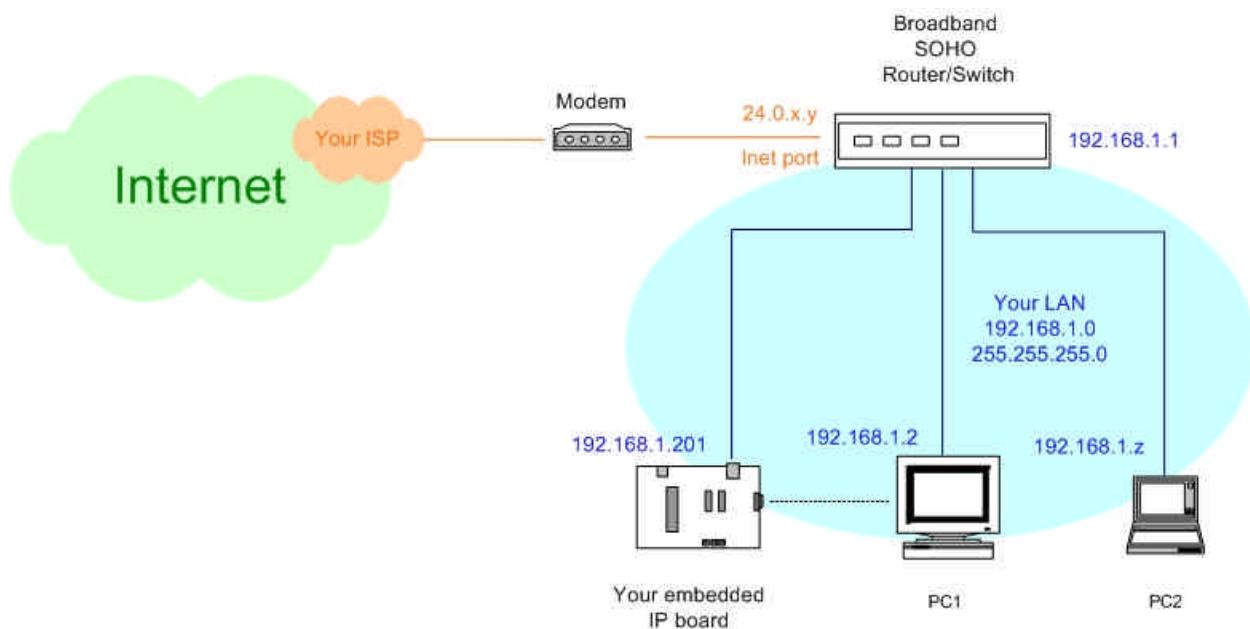
Figure 10

8 – Connecting to the Internet

If you are planning to get your embedded IP board access some services over the Internet, for example if you included the SNTP module to retrieve the time reference from a remote time server, there are some configuration details for your board and network setup that you must take in account.

This section will provide a brief description about the overall network setup and configuration settings for a standard home office network. You must have a basic understanding of the TCP/IP protocol suite, IP addressing, routing, DHCP and NAT (there are plenty of on-line resources and publications to learn more about TCP/IP basics and related services and protocols).

The following diagram shows the typical situation in a Small Office/Home Office (SOHO) with a broadband connection to the Internet.



In this example Internet access is provided via a broadband connection such as cable or DSL, the broadband modem is connected to the **Internet Port** of a broadband router/switch. Switch ports of the broadband router are used to connect different devices to the Local Area Network, including your embedded IP board.

The Internet Service Provider dynamically assigns a **public** IP address (in the example 24.0.x.y). The Local Area Network uses the **private** network address 192.168.1.0 with netmask 255.255.255.0, making the 192.168.1.1–192.168.1.254 range available for devices in the LAN.

The broadband router is also configured to provide dynamic IP address assignments for the LAN using DHCP with the 192.168.1.100–192.168.1.149 address range reserved for the DHCP server allocations.

The router LAN Ethernet interface has the statically assigned 192.168.1.1 address; PC1 is configured with static IP 192.168.1.2 and the embedded IP board 192.168.1.201 all with netmask 255.255.255.0 and default gateway 192.168.1.1; and PC2 will obtain its IP address, netmask and gateway via DHCP.

You also have the option to obtain the IP address for your embedded IP board via DHCP, but if you are planning to provide access to your board from remote sites in the Internet you will have better chances of success using the static IP address.

As an example the figure below shows the configuration screen of a Linksys BEFSR81 broadband switch/router based on the parameters mentioned above.



Now to provide access to your embedded IP board for connections originated from a remote site in the Internet you have to enable *port forwarding* in your broadband router.

As you should know your router performs Network Address Translation in both directions, since your embedded IP board has a private IP address there is no way for the remote host to know how to reach it, the only public address you have is the one assigned to your router by your ISP, then your router must support this feature and has to be configured to *forward* packets to a specific port (UDP, TCP or both) to a specific IP address in your LAN, in our case 192.168.1.201.

If you are planning to provide access to the HTTP server included in the Microchip TCP/IP stack from a remote site, it is recommended that you change the HTTP server port to avoid conflicts with the internal HTTP server in the broadband router.

We'll use then for example 8080. (The HTTP server port in the current version is defined by the macro `HTTP_PORT` in the `config.h` file)

The following figure shows the Linksys BEFSR81 configuration screen where any TCP connection with destination port 8080 will be *forwarded* to the 192.168.1.201 address in your LAN and port 8080, ie your embedded IP board HTTP server.



Once you have your router configured you will be able to access the HTTP server on the board using the URL=<http://24.0.x.y:8080/>.

Now it will be much easier and fancy to access your board using a host name and domain name. Also the IP address dynamically assigned by your ISP is subject to change and there are no guarantees that you will be assigned always the same address.

Then for this you need what is called **Dynamic DNS** or **DDNS**, mapping a fully qualified domain name to the current IP address in use by your router.

There are several DDNS service providers that offer this service on a temporary basis for free (if you want a permanent assigned name you must pay for the service), and many standard broadband routers include this feature and support several DDNS providers.

The following figure shows the DDNS configuration screen for the Linksys BEFSR81 using the services of **DynDNS.com** and the hostname `myboard` and `dnsalias.net` domain.



With this feature enabled and configured, when your router starts up or when the IP address changes it will register the IP address with DynDNS.com associated with myboard.dnsalias.net, and you will be able to access your HTTP server on the embedded IP board by using the URL=http://myboard.dnsalias.net:8080/.

Revision History:

June 2007, Original en002.

July 2007, Updated en002b for software release v3.75.6.

Notes:

Information provided in this document is believed to be accurate and reliable. However LJCV Electronics assumes no responsibility for errors or omissions. LJCV Electronics assumes no responsibility for the use of this information or the devices and/or systems referenced in this document. This document may be subject of future updates and LJCV reserves the right to discontinue production and change specifications and prices without notice. LJCV Electronics makes no warranty of merchantability or fitness for any purposes.

©2007, LJCV Electronics, All Rights Reserved.
www.ljcv.net

SPI is a registered trademark of Motorola Corporation. The Microchip name and logo, PIC, PICmicro, MPLAB, ICSP are registered trademarks of Microchip Technology Inc. The Ethernet Boot Loader is copyrighted by Brush Electronics and Andrew Smallridge and is provided for non-commercial use. Reverse Engineering of the code is strictly prohibited.

All other trademarks mentioned in this document are property of their respective companies.