Appendix A: TERN Implementations for I2Chip TCP/IP Module

A.1 Background

The I2CHIP TCP/IP module from Wiznet provides a hardware implementation of the TCP/UDP IP protocol stack. The module allows for vastly improved network performance by offloading time-consuming network management code to hardware. Detailed documentation for the module is available from Wiznet or on your TERN development CD.

FIGURE A1: I2Chip Module

The I2CHIP module is mapped directly into processor memory space for better performance. The direct mapping location, as well as the size of the mapped space can differ from controller to controller. When working with different TERN controllers enabled with the I2CHIP module, you must select the proper #define statements to describe your hardware configuration. See the section on 'i2chip_hw' below for more details.

A.2 I2Chip Module Summary

The I2CHIP module allows a total of 4 simultaneous sockets. Each socket can be configured for TCP, UDP, as well as lower-level raw IP packet modes. The sockets can also be configured for a 'listen' server socket, or as a client socket. The module allows for a total of 8 KB of packet RECEIVE memory buffer, and 8 KB of packet TRANSMIT buffer across all 4 sockets.

The buffers are mapped directly into the processor's memory, meaning they can be accessed directly from a pointer. The existing driver code sits on top of the memory layer and provides easier interfaces to this underlying data.

The user application has full ability to set most (all?) meaningful TCP/UDP/IP flags, ranging from the obvious (port/address), to the less obvious (timeout options). The provided drivers should work correctly under both small AND large memory model compilations.

The module is interrupt-driven, and updates each socket as needed when the state of the connection changes. Available socket states include all TCP states (CLOSED, SYNSENT, SYNACK, SYNRECV, CLOSE_WAIT1, CLOSE_WAIT2, etc...).

A.3 Network Initialization

The first step is to configure your board with the proper network settings.

1) You will need to connect a standard Ethernet cable with RJ45 connector to your local area network.

2) Determine your network settings. Ask your network administrator, or, on a Windows PC connected to the same local area network (through a hub or switch), and run **'ipconfig'** inside a DOS window.

The results will show three sets of values (sample values shown in Figure A2):

```
C:\Documents and Settings\HP_Administrator>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.2.100
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.2.1
```

FIGURE A2

3) Your C-Eye controller should be set to use the subnet mask and default gateway shown here. In the sample code, these values are in these two statically-defined arrays (Figure A3):

```
u_char GatewayAddress[] = {192, 168, 2, 1};
u_char SubMask[] = {255, 255, 255, 0};
```

FIGURE A3

4) You need to select another IP address on the same subnet as your PC. This means the first 3 values will be the same as your PC's address in the above example; the last digit needs to be different from any other device on your subnet. For example, we choose: 192.168.2.206. (192.168.2.xxx remains the same.) This value should also be set statically, as below (Figure A4):

```
u_char ControllerTcpAddress[] = {192, 168, 2, 206};
```

FIGURE A4

5) The MAC address needs to be unique from any other device on the subnet. If you're using multiple C-Eye's on the same network, connected to the same hub, make sure they have different MAC addresses. Contact TERN if you are working with multiple C-Eyes and would like our help selecting unique MAC address ranges.

6) Save the file after making your network changes, and build this node again (right-click on the .axe node, and select 'Build node', as seen in Figure A5).

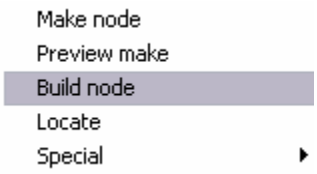

FIGURE A5

A.4 User Application Details

The typical user application follows this basic model:

- Configure the i2chip module with network parameters (4-byte IP address, 6-byte MAC address, 4-byte gateway address, 4-byte netmask, and buffer allocation between the 4 available sockets if needed); After this is done, the module can already be remotely ping'ed at the above address.

- Establish a socket with appropriate type/port parameters (TCP stream server socket on port 80, for example).

- Regularly monitor the status of the socket using the 'select' function as part of the main application loop. ie. respond to incoming connections, create out-going connections, and read/write data.

A.5 'i2chip_hw'

Because of the large number of controllers with which the i2chip module is used, you must be careful to define properly the specific hardware you're working with. Expect the 'i2chip_hw' code to be updated over time to support newer systems. The documentation in 'i2chip_hw.h' has precedence over the architectures listed in this email.

Each hardware architecture is selected via '#define' statements. These are listed below. To set a #define value (should be uniform across your entire target, not just in a single .c or .h file):

- Right click on the .axe node,
- Select 'Edit local options' (Figure A6)

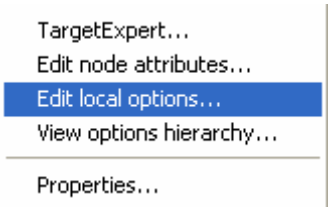

FIGURE A6

- Select 'Compiler->Defines' (Figure A7)

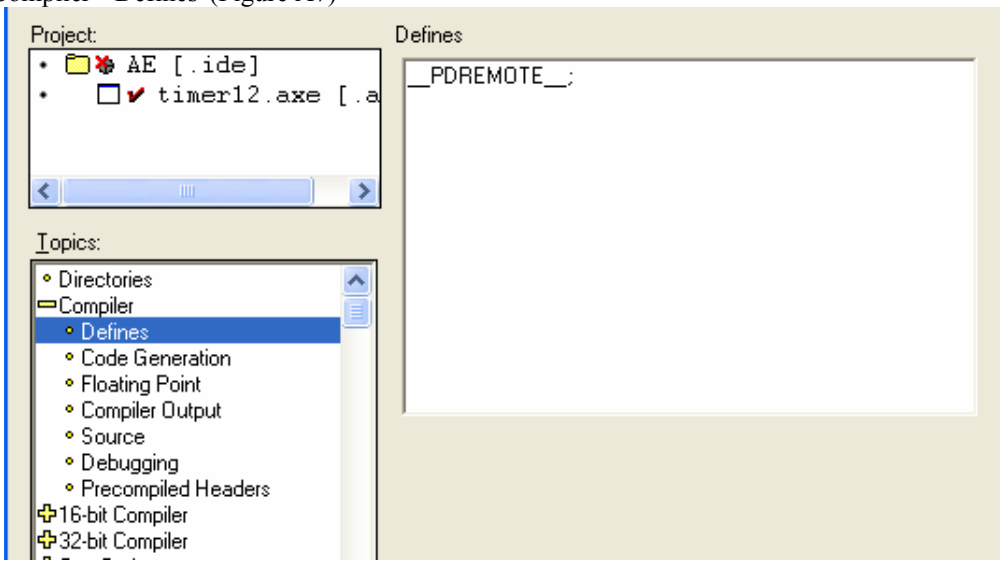

FIGURE A7

- In this field, all of the relevant #define values should be stored.

By default, this should include __PDREMOTE__ (a value informing the complier that you're currently debugging the application). You can add to this to form, for example: __PDREMOTE__;TERN_186;TERN_P51 (186-based controller on a P51 expansion board).

Supported **#define** values included:

- **TERN_186:** 186-based A- boards (EE, C-Eye included).
  - **TERN_ST**: For the Smart-TFT, this must be explicitly defined.
- **TERN_RE:** All other 186-based boards (RE, RL, RD, RA, RB) when used with an expansion board.
  - **TERN_RD:** For the R-Drive, this macros MUST be explicitly defined.
  - **TERN_RL:** For the R-Engine-L, this macros MUST be explicitly defined.
- **TERN_586:** ... 586-based boards, 5E or 5P. With 586, all expansion modules are accessed through I/O mode only (I2CHIP_WINDOW_IO).
  - **TERN_5E:** This must be explicitly defined for the 586-Engine boards. It would indicate custom-mapping for the 5E hardware (typically requiring added wires for J4).
- **TERN_P51:** Indicates P51 expansion board and 'windowed' access to the memory mapped module; another macros also needed to indicate the Engine architecture you're working with.
- **I2CHIP_MMC:** Indicates MMC expansion board and also 'windowed' access to module; for the MM-C, make sure 'i2chip\mmc.c' is also included in your project.

  Note: Defining TERN_MMC or I2CHIP_MMC will have different meanings in httpd_fs sample code (using CompactFlash filesystem).
  - **I2CHIP_MMC:** Indicates we should be using MM-C for I2CHIP/Ethernet capability.
  - **TERN_MMC:** Indicates we should also be using MM-C for file-system access.
- **TERN_SC** ; **TERN_RL:** For the SensorCore/R-Engine-L controllers, implying a direct mapping at 0x80000 with 'even byte' only shifted addressing.
- **TERN_EE ; TERN_CEYE:** For the EE/CEye controller, implying direct mapping at 0x80000 (if no expansion board used).

Also, be very aware that every board requires a different runtime library. When working with the 186-based boards (EE, C-Eye), you should be compiling with 'ae.lib'. When working with the R- series of boards (RE, RL, SC), you should be compiling with 're.lib'.

A.6 Sample Code

Wiznet provides a variety of generic TCP/UDP/IP sample applications, including FTP, DHCP, SMTP, etc.

TERN has ported a few of these over specific to the TERN platform, and expanded them by adding support to TERN peripherals (like the CF-based FAT16 filesystem).

These samples are described below.

A.6.1 Testing

As you download and test the code below, you can follow two basic steps for testing:

1) Make sure you can "ping" the board. From a PC located on the same LAN subnet, open a DOS command prompt window. In this window, run the command 'ping xxx.xxx.xxx.xxx' (where xxx.... corresponds to the address of your
board, such as 192.168.2.205).

2) For the HTTP server code, you can open up Internet Explorer and open up the URL: http://192.168.2.205

3) For generic TCP server code, you can open up a client connection. On Windows PCs, you can type the command 'telnet xxx.xxx.xxx.xxx yy' from any DOS command prompt (xxx = board address, yy = port number).

A.6.2 Code Arrangement

NOTE: Within the i2chip sample project, for convenience, we have created an "i2chip_src" [SourcePool]. This node contains the files common to all i2chip-based applications. The applications (.axe) nodes in this directory all refer "indirectly" to this SourcePool (indicated by BOLD letters). Within your own application, you need to add the same files in the source pool (socket.c, i2chip_hw.c) to your own .axe node.

A.6.2-1 186-Based Boards

For 186-based boards, your project is in the project *\tern\186\samples\i2chip\i2chip.ide*.

*** httpd_fs**

(httpd_fs_u.axe, and httpd_fs_r.axe for different hardware platforms: the EE40, and the RE + P51 expansion board respectively)

This sample layers a simple http daemon server on top of the i2chip module, and the TERN FAT16 filesystem. For details on the TERN FAT16 filesystem, see \tern\186\include\fileio.h, and \tern\186\samples\flashcore\readme.txt.

Relevant files are:
- httpd.c
The main application + request handling code. The httpd prepares 3 server sockets (each with 2 KB tx/rx buffers), all on standard port 80. Incoming requests are parsed (only GET requests are handled at this point). Once the request is processed by locating the proper data (or error response is set), the appropriate HTTP headers are created, and the data is finally sent out the socket.

- httpd_fs.c
This file translates the request into filesystem behavior. Based on path, the right subdirectory is found, and the file located/loaded. When the request is finalized, the file is closed. (Depending on the desired behavior, it might be faster/efficient to create a filesystem "cache" to load frequently read files... rather than opening/closing the same file every time a request is processed.)

- socket.c
Implementation of the socket abstraction for the i2chip module. Required for any user application using i2chip module. Functions are defined in 'socket.h',

- i2chip_hw.c
The relevant hardware description file; Require for any user application using i2chip module.

- ae.lib: Used for A- family of boards (including the EE, C-Eye)
OR re.lib: Used for R- family of boards (including RL, RD)
- filesy16.lib: Necessary libraries for filesystem access.
- mm16.lib
- heapsize.c: Define heap needed for dynamic allocation; used by fs.

*** tcp_client**.

This sample provides for a simple RS232-based TCP client. A 115200 baud, N81 serial connection is opened on SER1 with a simple menu based interface.

Supported commands include:
- 'c' (for establishing a new connection to a remote TCP server),
- 'i' (for listing local network configuration),
- 'w' (to begin sending/receiving data over the newly created connection).

The relevant files are:
- tcp_client.c: The user application which connects serial<->TCP data.
- socket.c: Socket implementation, as above.
- i2chip_hw.c: Hardware interfacce, as above.
- ae.lib: Library for A/R- family of boards, as above.
OR re.lib

* tcp_echo

This sample provides for a simple TCP echo server on 4 ports: 4000, 4001, 4002, 4003. The 4 sockets are established to listen to each of the 4 ports.

When a conection is established, the program monitors incoming data. Any data that is received is immediately sent back out the same port.

The relevant files are:
- tcp_echo.c: The user application which echoes incoming TCP data.
- socket.c: Socket implementation, as above.
- i2chip_hw.c: Hardware interfacce, as above.
- ae.lib: Library for A/R- family of boards, as above.
OR re.lib

* io_ping

This very basic sample is only useful for the now discontinued I2CHIP, bus-based expansion board.

* httpd_img

For customers using the C-Eye controller, a HTTP-based daemon serving up bitmap images is available in the samples\ceye\ directory.

* http_adc

This sample responds to any incoming TCP connection (on port 80) by serving up a basic HTML page, with the contents of some hypothetical ADC calculations.

The main .c file (http_adc.c) defines the HTML return definitions near the top of the program. It relies on two generic "adc" functions which the user can re-program to provide actual ADC data:

void adc_init(void);
UINT16 adc_rd(UCHAR8 channel);

Related files:
- socket.c, socket implementation, as above.
- i2chip_hw.c, hardware interfacce, as above.
- ae.lib, library for A/R- family of boards, as above.
OR re.lib

A.6.2-2 586-Based Boards

For 586-based boards, your project is the file: *\tern\586\samples\i2chip\586_io.ide*

The same samples as above are duplicated. Please refer to above comments for more details about basic implementation. Related files include:

- socket.c, socket implementation, as above.
- i2chip_hw.c, hardware interface, as above.
- 586.lib, 586-based system library.
- filesy16.lib, for integrated CompactFlash on 586-Engine-P boards.

filesys.lib, for expansion CompactFlash on MM-C or FC-0 expansion boards.
- mm16.lib, for integrated CompactFlash on 5P boards.
mma.lib, for expansion CompactFlash on MM-C or FC-0 expansion boards.
- i2chip\mmc.c, for accessing MM-C page/window modes.

A.7 Version Info

- Version 1.00: Initial release of this sample set.
- Version 1.1: Adding httpd_img, and http_adc to this sample set.
- Version 1.11: Adding RD and RL notes.
- Version 1.2: Added 586-based versions for both MMC and P51.