OpenTCP

A Free Open Source TCP/IP Stack

1.0 Introduction

OpenTCP® is a license and royalty free TCP/IP stack available to the public. It was created by Viola Systems in Finland (www.violasystems.com). The code is supported and distributed via the web site www.opentcp.org. Other services that available for the stack are HTTP, DHCP, TFTP, SMNP, POP3, BOOTP,DNS Lookup as well as Ethernet support. Full documentation on the stack and sample code are also available.

2.0 OpenTCP Architecture

2.1 Stack Size

This TCP/IP was create to be sensitive to embedded applications. The ROM and RAM usage are about 20k/1k bytes. Because of this, you may ask yourself "If this stack is so small, what did they leave out?" The answer is that by making certain limitations of the stack, they were able to greatly reduce the ROM and RAM requirements normally associated with a TCP/IP stack. This does not mean the stack will only run on certain networks, it just means that you may have to take some considerations into account when passing large amounts of data.

2.2 Stack Limitations

As mentioned above, the TCP/IP stack has removed some features that enable it to shrink considerably in size. The restriction placed on the stack is that it doesn't support IP Fragmentation or TCP Segmentation. I will explain….

2.3 IP Fragmentation

In the IP protocol, if an IP packets traveling across various networks encounters a network that cannot handle its packet size, that packet may be broken up into smaller pieces in order to be transmitted between the sending and receiving nodes for that particular network. This is called IP Fragmentation, a feature that OpenTCP does not support. If a packet is sent to the OpenTCP device across the internet, it may be broken up into pieces along the way in order to adhere to the required maximum packet size on other networks. This is fine. But, before it is delivered to the OpenTCP device, it must be reassembled.

Now knowing that, is this going to be a problem? Probably not. If you are using Ethernet for example, a max Ethernet packet size can be 1526 bytes. Since you can instruct the TCP layer to only send you less than say 1500 bytes at a time, the remote side has no need to fragmented any packets anyway because they will never be larger than the maximum size designated by Ethernet. Knowing that, loosing IP fragmentation is not a big deal.

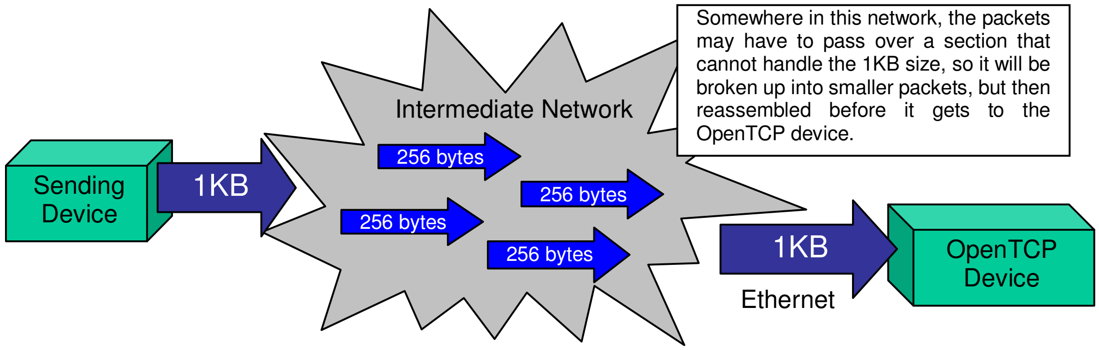

Figure 1 - IP Fragmentation over Intermediate Networks

2.4 TCP Segmentation

TCP Segmentation can be described as what you do when you have 1MB of data to send, but Ethernet can only handle 1.5KB packets at a time. The TCP protocol allows you break up that data into smaller packets in order to transfer them. That is what's know as TCP Segmentation.

TCP also has what is called a TCP Window. The window is how much un-ACKed data you can send to a device at once. When a TCP connection is being established, the two communicating devices negotiate on a Window size that they both can handle (pretty much the smaller size of the two wins). It should also be known that when a TCP packet is sent, the receiving side has to acknowledge that the packet has been received without errors. So, a TCP Window tells the sending side how much outstanding data (packets not ACKed yet) it may have at one time.

Now with all that information, I can explain how the OpenTCP stack can run with 1K of buffer space instead of other stacks that may need 128KB of buffer. If for example a Window size is determined to be 4Kbytes, then a sending side can send four 1Kbyte packets then if it wants to. The problem is that now the receiving side has to account for those smaller packets and if they come out of order. Also, let's say 1 of those packets gets lost, but the other 3 get though. After the receiving side sends the ACKs for those three 1Kbyte packets, the sender might start sending another 3Kbytes of data because only 1Kbytes in the TCP window are outstanding. But, the receiving side still can't pass the data it got back up to its application because it's still waiting for the sending side to re-send that missing piece. So now, the receiving side will be getting new data yet still hasn't been able to free up the buffer space being taken up from the previous sent data.

What OpenTCP does is instead of allowing data to keep coming, it doesn't ACK any data that comes out of order. The only packets that get ACKed are the next packet in the sequence. If the sending side doesn't see an ACK for the packets it sent, it will re-send them again anyway.

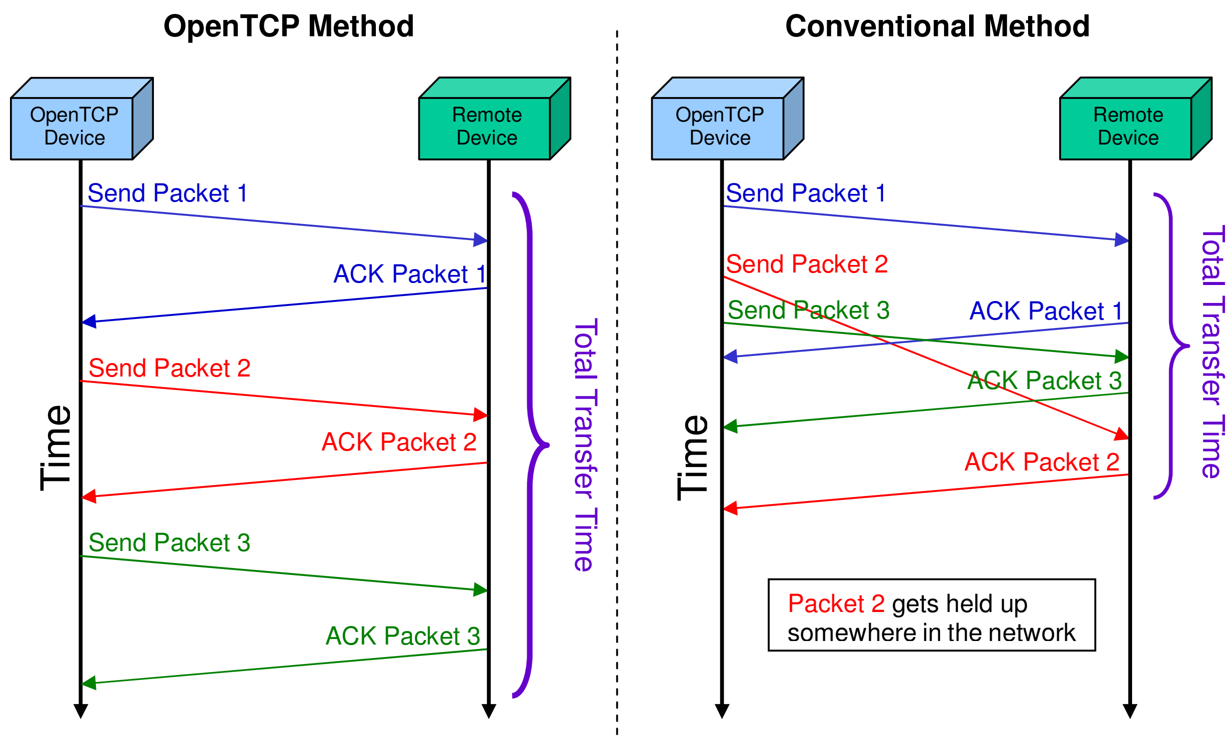

Figure 2 – TCP Segmentation

Now for sending packets, OpenTCP will only send the next packet after it has an ACK for the last data that was sent. If that packet doesn't get ACKed after a certain amount of time, Open TCP will try and send it again.

It should also be noted that if the OpenTCP device is communicating to a higher horsepower network device such as a PC or server, OpenTCP sets the TCP Window to 1Kbytes. Now, if the PC wants to send a lot of data to the OpenTCP device, it will probably send 1Kbyte packets one at a time because that is the most that it can send according to the TCP Window.

As you can imaging, by reducing the complexity of the state machine for sending and receiving packets, ROM and RAM space are reduced greatly. The cost for this is speed. Instead of being able to send out a bunch of TCP packets at once, and then later go back and resend the ones that didn't make it, you are sending data one packet at a time.

2.5 UDP Packets

A UDP or User Datagram Packet is like a TCP packet except that the receiving side does not ACK back to the send whether or not it got the data correctly. Since there is no state machine involved with this method, OpenTCP can stream out data (audio, video, etc) as fast as it wants.

2.6 No Buffer Copy

Another aspect that OpenTCP uses is that when you want to send data, the buffer that you pass to the TCP layer will be used to form the actual packet in that gets send to the Ethernet module. In other stacks, that data is first copied into a separate buffer, and sometimes copied more than once before it actually gets transmitted. By using the same buffer that you passed to the TCP module, the system can save on RAM as well as processing time used for copying. All you have to keep in mind is that you must pass a buffer that enough space at the beginning of your data for the TCP/IP headers. These amounts are already defined for you in the OpenTCP header files so that's not a problem.

The TCP receive routines use the same theory. When a new TCP packet arrives, the pointer that is passed to the application is for the same buffer that was used to read out the packet from the Ethernet chip. If your application would like to save that data, you can then copy that data someplace else before allowing OpenTCP to use that buffer to get another incoming network packet.

3.0 Demo Extras

3.1 File2C Utility

A command line utility was created called File2C.exe. This utility is used to convert the contents of any file into a 'C' array so that it may be added to the file system in the demo. A file may be created to reside in either RAM or ROM (using const). A corresponding .h header file is also created in order to extern the 'C' arrays.

This utility also has the ability mark place holders or TAGS when converting an .html or .htm file. By placing accent characters ' ` ' in your file, the utility will record that particular file index as a #define as well as remove the accent character from being entered into the 'C' array. The #defines are then added to the resulting .h header file. An example of a "TAG define" in a header file is as follows…

```
/* File Tag defines */
#define index_html_TAG_1 682
#define index_html_TAG_2 690
```

3.2 File System

A file system was created for the demo that can hold files in a mix of RAM and Flash so that they could be dynamically changed before being served. By being able to store the file in a mix of RAM and Flash, we would not have to keep the entire file image in RAM (wasting RAM space on parts of the file that would never change).

For example, every time text is passed to the LCD driver, it is saved to a text string in memory. When the index.html file is served, it displays the current text on the LCD. To do this, the html file must be able to be modified before it is sent out. Tags (mentioned in the previous section) were added to the html file in order to separate out the strings that would represent the LCD text in the HTML file. So, when the index.html was added to the file system for the demo, the portions of the html to be served that would not changed were added as pointers to file image that resided in Flash. As for the portions of the HTML that would be dynamically changed, the LCD text strings, those were added as pointers to the text strings maintained by LCD driver. So at any time the HTML file is requested, it will contain the current LCD text being displayed.

4.0 Running the OpenTCP demo

While the Open TCP code comes with some simple examples, we though it would be more impressive to have an interactive demonstration. The following explains how to setup and run the interactive demo for each the M16C, H8S and the SH MCUs. The demo consists of firmware that can be compiled to run on the mentioned MCUs, a custom Windows GUI program and the ability to interact with a web browser.

4.1 Setting the IP

In order to use the demo, you must first assign an IP address the board will use. To do this, you need to edit the gui_demo.c file and rebuild your project. You just also change the default gateway for your network. If you are using a cross cable in order to plug directly into your laptop for example, you should use an IP number on the same sub-net as you laptop. For example, if the IP address of your laptop is 10.10.10.34, you could assign to your board the IP address 10.10.10.20.

From file gui_demo.c

Change the numbers in RED.

```
/* Program in static IP address */
localmachine.localip = (UINT32)(10 * 0x1000000) +
                       (UINT32)(10 * 0x10000) +
                       (UINT32)(10 * 0x100) +
                       (UINT32) 20 ;              /* 10.10.10.20 */

/* Default gateway */
localmachine.defgw = (UINT32)(10 * 0x1000000) +
                     (UINT32)(10 * 0x10000) +
                     (UINT32)(10 * 0x100) +
                     (UINT32) 1 ;                 /* 10.10.10.1 */

/* Subnet mask */
localmachine.netmask = 0xFFFFFF00;                /* 255.255.255.0 */
```

Also, if you plan on running more than one board at once, you will need to change the Ethernet MAC address so the two boards don't have the same hardware address.

From file gui_demo.c

Change the numbers in RED.

```
/* Ethernet (MAC) address [5]:[4]:[3]:[2]:[1]:[0]*/
localmachine.localHW[5] = 0x08;
localmachine.localHW[4] = 0x00;
localmachine.localHW[3] = 0x70;
localmachine.localHW[2] = 0x33;
localmachine.localHW[1] = 0x44;
localmachine.localHW[0] = 0x55;
```

After the code has been built, download and run it using appropriate debug tool. The LCD should change. After initialization, the LCD should display "No Conn" mean no connection to the board has been made yet.

4.2 Windows GUI Program

Now run the Interactive Windows GUI program **tcp_gui.exe**. Change the IP address in the upper left-hand corner of the window by clicking on the IP address button. You should change the IP address to the one that you assigned to your board.

Now click the "Establish connection" button in order to open a TCP and UDP connection to your target board. If a successful connection is made, the LCD on the target board will read "Conn Est" meaning connection established. The demo has been set up so that the OpenTCP stack will close a socket if it remains inactive for more than 2 minutes.

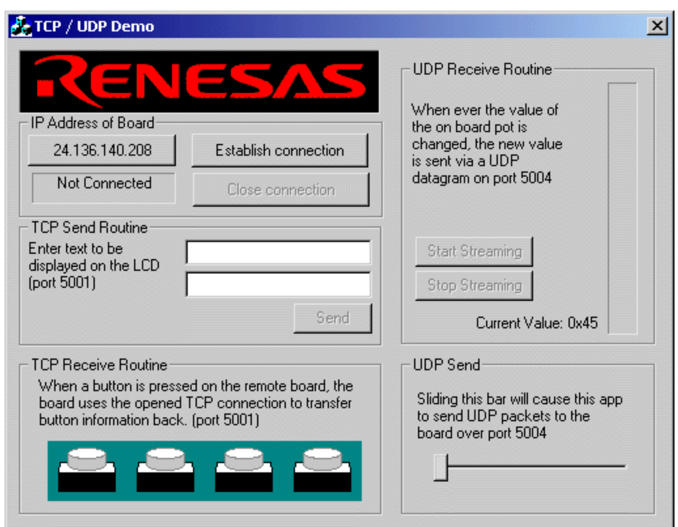

Figure 3 - Interactive TCP Windows GUI Program

To write to the LCD, simply type in the edit boxes (one for each line on the LCD) and click the "Send" button. This sends the LCD text using the TCP socket it opened when the connection was established.

When a button is pressed on the target board, that info is also sent via the TCP connection to the windows program and the corresponding picture of the button on the GUI will change.

To send/receive UDP packets, click the "Start Streaming" button. This will send a TCP packet to the target board telling it to start sending the current A2D value of the pot via a UDP packet every time it changes. Also, when the slider bar on the GUI is moved, the new value is send to the target board via UDP packets and the value is displayed on the LCD.

4.3 HTTP Web Browser Interface

The demo is also running as a HTTP server in the background. If you open a web browser and type in the IP address of your demo board as the URL, a web page will be served back showing the current text on the LCD. The OpenTCP source code comes with a HTTP server component, but the file system was created by Renesas. The web page that was passed back actually resides in a mix of RAM and Flash space, that is how the page can be modified before send back.

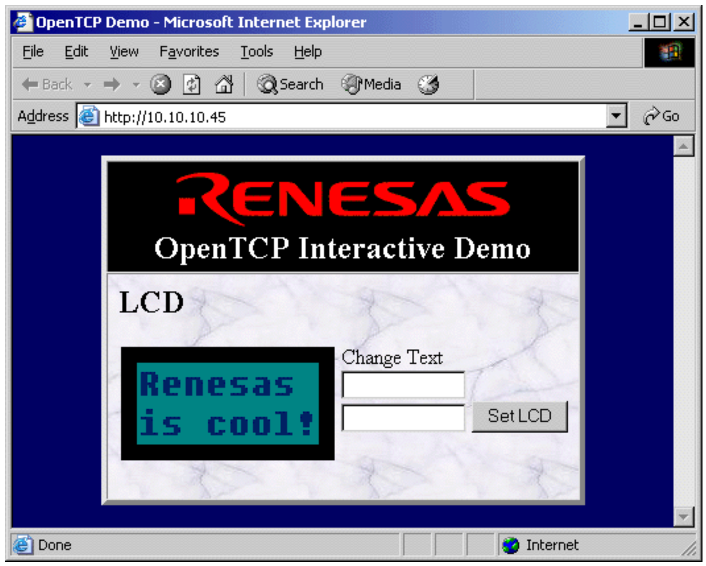

Figure 4 - HTTP Interface of Demo

5.0 Demo Setups

The following describe the development environments used to crate this demonstration.

5.1 M16C M16C Environment

- MCU: M16C/62P (M30626FHPxx)
- Board: SKP16C62P (12MHz XTAL)
- Compiler: NC30WA V5.20 Release 1
- IDE: Hew3
- Debugger GUI: KD30 V.3.10
- Debugger Interface: FoUSB-ICD
- Demo Size:
  - Code: 17,584 bytes
  - ROM Data: 3,644 bytes
  - RAM Data: 6,632 bytes
- NOTES:
  - The TM project was set up to #define M16C_62P for all the compiled files. This was to determine which

MCU specific code need to be used for files that were common between the M16C, H8S and SH platforms.

5.2 H8S Environment

- MCU: H8S/2674R (HDxxxx)
- Board: EDOSK2674R
- Compiler: H8S, H8/300 Toolchain Ver 5.0.2.0
- IDE: Hew 2.2
- Debugger GUI: HDI Ver 4.00 (w/ monitor support)
- Debugger Interface: Serial HMON
- Demo Size:
  - Code: 19,689 bytes
  - ROM Data: 7,068 bytes
  - RAM Data: 6,732 bytes
- NOTES:
  - The project used the HDI debugger and HMON Serial Monitor. In order to use HMON has the debugger interface, the ROM sections had to be mapped to RAM. The HEW 2.2 project created has 2 build configurations: Debug and Release. The Debug configuration is set up to locate all the code in RAM space so that it may be downloaded and run with HMON. The Release configuration locates sections in Flash so that the demonstration may be used without the debugger.
  - The HEW project was set up to #define H8S_2674 for all the compiled files. This was to determine which MCU specific code need to be used for files that were common between the M16C, H8S and SH platforms.
  - An add-on board was created for the EDOSK board for user interaction. The add-on board connected the board J1 connector and consisted of an LCD, 4 push buttons and a pot. The schematic for this add-on board is below.

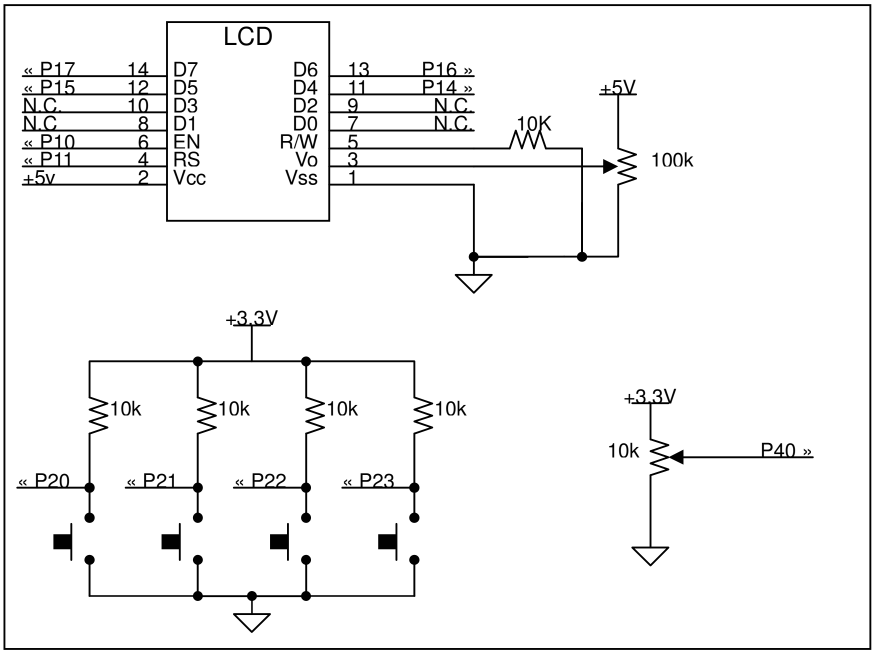

Figure – Add-on Board Schematic for EDOSK 2674R

6.0 Reference

OpenTCP® is a registered trade mark of **Viola Systems Ltd**, Finland

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com/

Contact for Renesas Products

E-mail: csc@renesas.com

Data Sheet

(Use the latest version on the home page: http://www.renesas.com/)

User's Manual

(Use the latest version on the home page: http://www.renesas.com/)