The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN205383

FR, MB91460, ADA-FR-Ethernet,Opentcp

This application note describes how the OpenTCP stack from Viola Systems is integrated in the demo application for the ADA-FR-ETHERNET adapter board. Unlike the demo, which is available for 16FX, the FR demo utilizes the features of FreeRTOS™ to handle scheduling and synchronization of different Ethernet-related tasks.

Contents

1 Introduction....1
2 Hardware....2
2.1 Development Setup....2
3 Software....2
3.1 Configuration of the OpenTCP stack....4
3.2 Timer functions....4
3.3 High level Ethernet functions....4
3.4 Basic software test....5
4 Web server....6
4.1 How web pages are stored in FLASH....6
4.2 How to address the web pages....6
4.3 How to deliver the web pages....7
4.4 The hash algorithm....7
4.5 How to build dynamic webpages....8
5 Telnet....9
5.1 The Cypress Telnet Server....9
5.2 The Cypress FShell....9
6 How to debug Ethernet Applications....11
6.1 Connection through crossover cable....11
6.2 Connection through a switch or hub....13
7 Status Display....13
8 Additional Information....14
Document History....15
Worldwide Sales and Design Support....16

1 Introduction

This application note describes how the OpenTCP stack from Viola Systems is integrated in the demo application for the ADA-FR-ETHERNET adapter board. Unlike the demo, which is available for 16FX, the FR demo utilizes the features of FreeRTOS™ to handle scheduling and synchronization of different Ethernet related tasks.

ADA-FR-ETHERNET is an adapter board for the SK-FR-144PMC-91467B evaluation board. It features a 10/100 MBps/s Ethernet controller, which is connected via the external bus interface.

Although it is possible to use the OpenTCP stack without an operating system, the OpenTCP stack is written and prepared to be used within an operating system. A typical use case for Ethernet connectivity in an embedded device is to use the Ethernet connection as a communication path for another task. This involves sharing execution time between the OpenTCP stack and other applications that are running on the microcontroller. This is a typical territory of an operating system. Using FreeRTOS as the host for the OpenTCP stack allows a better distinction between OpenTCP and other applications that are running on the microcontroller. This increases the maintainability and expandability of the demo.

This product includes software developed by Viola Systems (http://www.violasystems.com). A big advantage of the OpenTCP stack is that it has a variety of client and server software integrated. Some protocols of the stack are already configured in the demo application. The OpenTCP stack needs at least around 27 KB of flash and 7 KB of RAM.

FreeRTOS is an open-source real-time operating system originally written by Richard Barry. In FreeRTOS Version 4.7.2, Cypress Microelectronics Europe provided the architecture-dependent parts to support the MB91460 series (FR) and the MB96300 series (16FX). The FreeRTOS project is located at http://www.freertos.org/.

2 Hardware

The base for the OpenTCP stack is the FR evaluation board SK-FR-144PMC-91467B. The board is assembled with the MB91F467B MCU, which features 1088 KByte FLASH memory, and 40 KByte RAM. The Ethernet connection is provided by the ADA-FR-ETHERNET adapter board.

The starter kit is normally powered by an external power supply. The external power supply can also be used for the ADA-FR-ETHERNET and SK-FR-144PMC-91467B in combination. Power over USB is also possible for the SK-FR-144PMC-91467B.

2.1 Development Setup

For an easy and uncomplicated debugging, a special debug setup is recommended. Before working with the ADA-FR-ETHERNET, the device must be connected to a network. It is better to connect the Ethernet hardware only to one computer, thus it is easier to monitor the Ethernet activities and to test the function of hardware and software.

For a direct connection to the computers network card, a cross over cable is needed. The pin layout of the cross over cable is mentioned below. For a convenient work environment, it is better to install a second network card in the computer. The first card is connected to the company network and the second one is connected to the ADA-FR-ETHERNET board. Using a dedicated network card allows to set up a private network for the development which reduces the errors from unknown trespassing network traffic and avoids the risk of disturbing the company network and your IT department.

Table 1

RJ45 Pin No.	Wire color first end		RJ45 Pin No	Wire color second end	
1	Orange/White		1	Green/White	
2	Orange		2	Green	
3	Green/White		3	Orange/White	
4	Blue		4	Brown/White	
5	Blue/White		5	Brown	
6	Green		6	Orange	
7	Brown/White		7	Blue	
8	Brown		8	Blue/White	

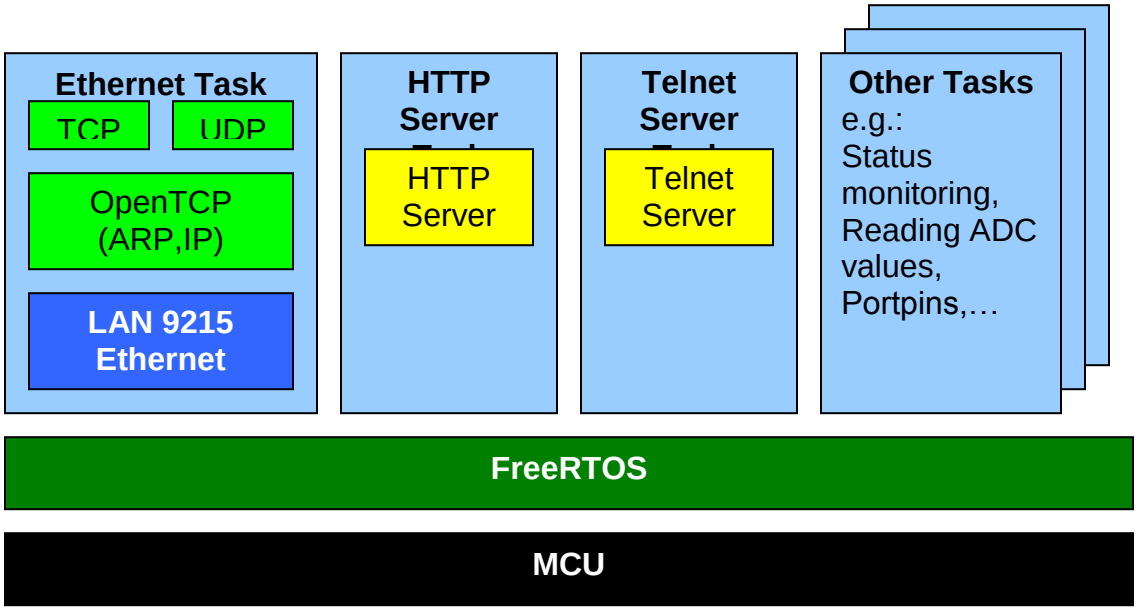
To monitor and analyze the traffic on the local microcontroller network tools like Wireshark can be used. To configure Wireshark please look at chapter 6. These tools capture all packets of a specified network card and show the Ethernet packets in real time. The software is able to interpret the packets and decodes the bytes corresponding to its protocol.

3 Software

For Ethernet communication, a freely available IP Stack called OpenTCP is used. The OpenTCP Stack was developed by Viola Systems. It is mainly used for 8/16 bit microcontroller platforms. The latest branch of the OpenTCP stack can be downloaded from this website: http://sourceforge.net/projects/opentcp/

The stack has built-in support for ARP, IP, UDP, TCP and ICMP. It is very easy to activate higher layer protocols like HTTP, TELNET, POP3, SMTP or TFTP. The following Diagram shows how the OpenTCP Stack is hosted by FreeRTOS in the ADA-FR-Ethernet demo application.

Figure 1



This Demo uses FreeRTOS™ to host the OpenTCP Stack. FreeRTOS™ is a Real time OS which was originally written by Richard Barry. A complete introduction to FreeRTOS™ is out of scope of this document. Documentation about FreeRTOS™ and its API can be found on the FreeRTOS™ Webpage http://www.freertos.org .

One challenge when using OpenTCP is to maintain a fair scheduling between the different Ethernet related tasks. In the 16FX demo, we used a main loop, which called the different run functions one after the other. This had the annoying side affect that one “task” could block all other “tasks” for an undefined time just by not returning to the main loop. In a sense, this is what would call cooperative scheduling. By using FreeRTOS as a task scheduler/Operating System, we can get preemptive scheduling with minimal overhead. This allows us to separate the different tasks into really different processes, which also makes the code more maintainable and easier to understand. Additionally, the demo application becomes more expandable by the user because new functionality can be added without the need to rewrite the code of the other parts of the application.

In the diagram above you can see a rough overview how the different tasks are organized. The former main loop is now running in the Ethernet task which initializes the Ethernet hardware and after that manages the ARP cache tables and the open TCP connections. The server functions now run in their own tasks and are scheduled by FreeRTOS.

The task initializes the http server and after that periodically calls the https_run() function to execute the http server. The task deletes itself if anything goes wrong during initialization of the http server. To provide some status information to the user, one segment of the 7-segment display of the board is set while the https_run() function is executing. It can be easily monitored when the http server has work to do by looking at the 7-segment display. The corresponding segment will flicker if the task has work to do. An overview of the used segments can be found in chapter 7.

Listing 1

```
static void vHTTPServerTask( void *pvParameters )
{
   /* Start httpd */
   printf("Starting HTTP Server...");
  if(https_init() >= 0)
  {
    printf("SUCCESSFUL\n");
    for(;;)
    {
      PDR06_D0=0;
      https_run();
      PDR06_D0=1;
      vTaskDelay( ( portTickType ) 100 );
    }
  }
```

This task is added to the task list by the Ethernet task after it has successfully initialized the Ethernet Hardware and the OpenTCP Stack. Listing 2 shows the function call which adds the http server to the FreeRTOS task list.

Listing 2

network.c:293

```
#ifdef HTTP_SERVER
  printf("\nCreating HTTPServer Task...");
  check_error(xTaskCreate( vHTTPServerTask,( signed portCHAR * )
"HTTPServer",0x100, NULL, tskIDLE_PRIORITY+2,NULL ));
#endif
```

3.1 Configuration of the OpenTCP stack

The only configuration need is setting the MAC and IP address, Subnet mask and gateway address. This is done within the Ethernet task. The settings are stored within the local machine structure which holds all the necessary information for the OpenTCP stack.

3.2 Timer functions

The OpenTCP stack uses clock functions to time different events like connection timeouts. These timers need to have millisecond precision.

Listing 3

```
 // OpenTCP needs
clock_time_t clock_time(void)  // return actual time in milliseconds
void clock_init(void)          // initializes clock
void delay_ms(unsigned long delay)    // delays for given milliseconds
```

Fortunately, FreeRTOS can provide this for us. We do not have to bother about the Open TCP Stack timer tick and we save a reload timer that now can be used for other purposes.

3.3 High level Ethernet functions

To pass and receive packets from the OpenTCP stack, some low-level functions need to be implemented. These functions are listed in the table below:

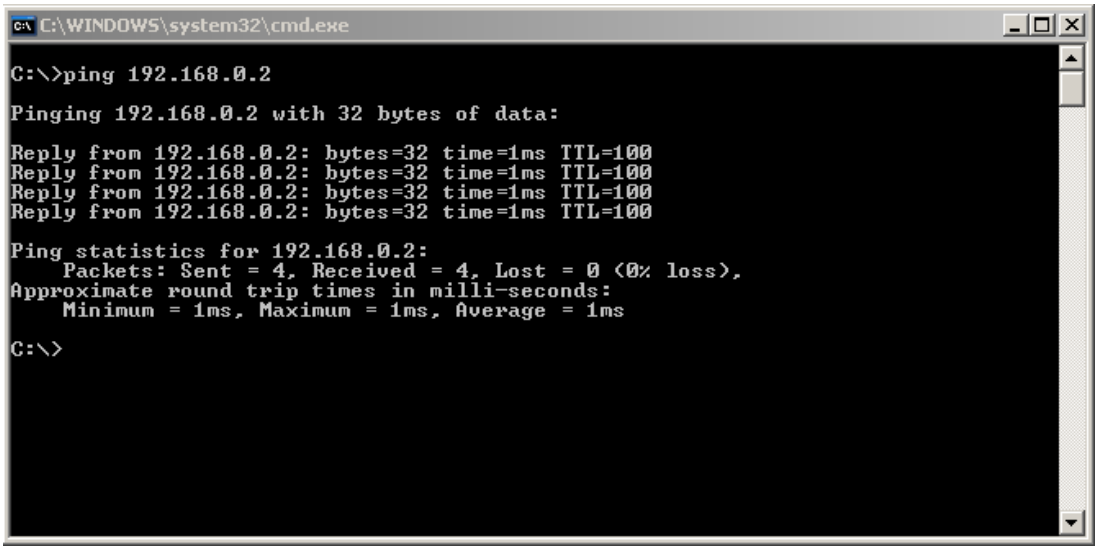
Table 2

Macro definition	Description
RECEIVE_NETWORK_B()	Returns one byte from the received Ethernet frame.
RECEIVE_NETWORK_BUF(c, d)	Function reads d from the received Ethernet frame and stores it at start address of c.
SEND_NETWORK_B(c)	Writes c into the Ethernet sendbuffer.
SEND_NETWORK_BUF(c, d)	Writes d bytes from pointer c into the sendbuffer.
NETWORK_CHECK_IF_RECEIVED()	Returns true if there is a packet in the buffer and stores source/destination MAC addresses.
NETWORK_RECEIVE_INITIALIZE(c)	Set the packet pointer to the specified position.
NETWORK_RECEIVE_END()	Set the packet pointer to zero (Packet dropped).
NETWORK_COMPLETE_SEND(c)	Write the frame with length c into the Ethernet controller.
NETWORK_SEND_INITIALIZE(c)	Set transmit pointer for TX packet to zero
NETWORK_ADD_DATALINK(c)	Writes the source, destination, and proto information into the packet memory.
RESET_SYSTEM()	Triggers a software reset
OS_EnterCritical()	Uses FreeRTOS™ Critical section implementation
OS_ExitCritical()	Uses FreeRTOS™ Critical section implementation

3.4 Basic software test

The OpenTCP stack comes with built-in ICMP support. This is very helpful for testing purposes. Even without any application (web server, telnet, pop3, etc.) running on top of the OpenTCP stack the ADA-FR-ETHERNET board should be available on the network. To test whether the board is properly connected to the network the utility “ping” can be used.

Figure 2



4 Web server

The board is now running with the OpenTCP stack properly. The board is connected to the network and is able to reply to ARP and ICMP messages. The next step is to bring the http server up. The server is shipped with the OpenTCP stack and the files are located in opentcp/http.

4.1 How web pages are stored in FLASH

To serve web pages with the OpenTCP web server the pages have to be stored in the FLASH memory first. The pages are available as C arrays. All pages are defined in http-fs.h located in the opentcp/http directory. Either the html pages can be stored like this:

Listing 4

```
const char http_index_html[] „<html>This is a
sample!</html>\n"
```

Or it can be stored in a data array. The problem is that it is not very easy to edit the hex dump directly. Pictures must be converted into a C-array. In the http-fs.c file some standard pages are stored like the error page or some table constructs for dynamic web pages. These files can be modified but should not be deleted. If new websites should be implemented the http-fs.h must also be changed.

4.2 How to address the web pages

The web server does not know the name of any file stored in the FLASH. The server uses a pseudo file system. The file system stores no filenames but hashes. To calculate these hashes the *hashgen* utility from the CD can be used. If the web server receives, for example, the hash number 43, it chooses the correct file array from the hash table. The next picture shows a part of the hash table:

Listing 5

```
switch (hash)
{
  case 11: /* index.html */
  {
    printf("  ==> Requesting index.html ...\n");
    https[ses].fstart = 0x00000001;
    https[ses].flen = strlen(&https_main_html[0]);
    https[ses].fpoint = 0;
    https[ses].funacked = 0;

    return (-1);
  }
```

In this case, the index.html file has the hash number 11. If you would like to receive this file from the web server, a callback function is called to deliver the content of the requested page. The content is defined by the fstart parameter. It is important that the index.html fstart number is the same than the fstart number in the delivery function. Information on how to deliver webpages is placed in the next chapter. The flen parameter specifies the length of the defined file. If binary files are served (e.g. images), a valid size must be defined because no “\0” detection works. The fpoint and funacked parameters are for the web server only. The value fpoint specifies how many characters are delivered and how many are waiting. The funacked parameter indicates whether a packet has been acknowledged or not. If it is acknowledged the web server delivers the next packet.

4.3 How to deliver the web pages

The web server detects the right web page from the fstart address. The following Listing shows a small code snippet:

Listing 6

```
if( https[ses].fstart == 0x00000001 )
{
  for(i=0; i < (https[ses].flen - https[ses].fpoint); i++)
  {
     if(i >= buflen)
            break;

     *buf++ = https_main_html[https[ses].fpoint + i];
  }

  return(i);
}
```

The "`if`" clause identifies the data area. The code in the loop cycle copies the data directly into the Ethernet buffer (the MCU RAM, not the LAN9115 RAM!). If the buffer is full, the "`if`" clause returns and the stack can send the packet to the receiver. After an ACK has been received, then the next packet can be send. The buffer is filled from fpoint till buflen. The number of bytes written into the buffer must be returned. The stack will build the TCP packet for the upper layer.

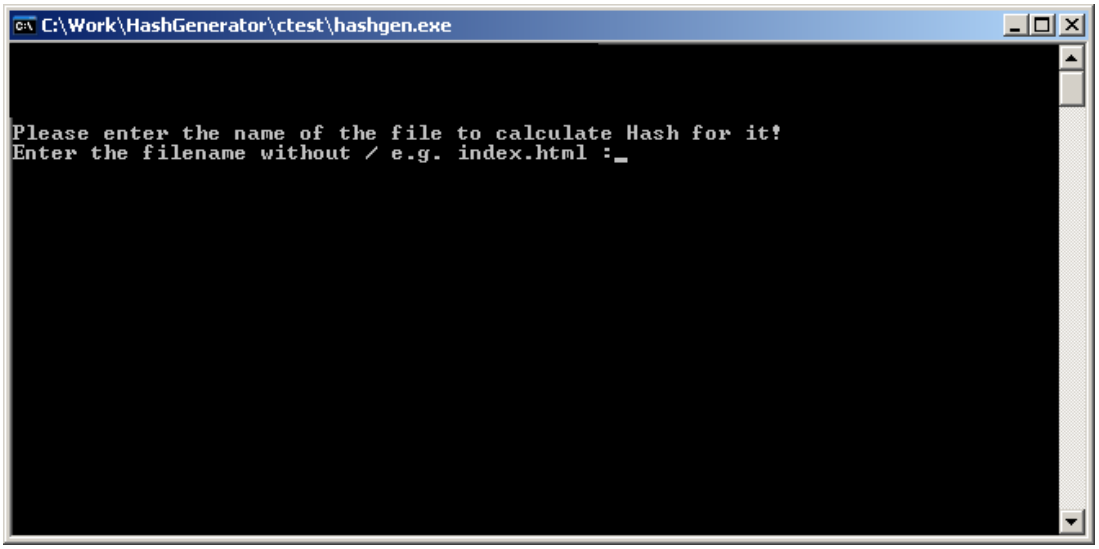
4.4 The hash algorithm

If the web server receives a file request the browser passes the file name via the GET command as a string. Because strings normally are very large and the microcontroller has not very much RAM the web server calculates the hash from "test.html". With this hash the web server can address the files in the FLASH memory. With the hash generator from the CD, calculation of new hashes is very easy. The hash numbers are unique.

4.4.1 Using the tool hashgen.exe to calculate the hash

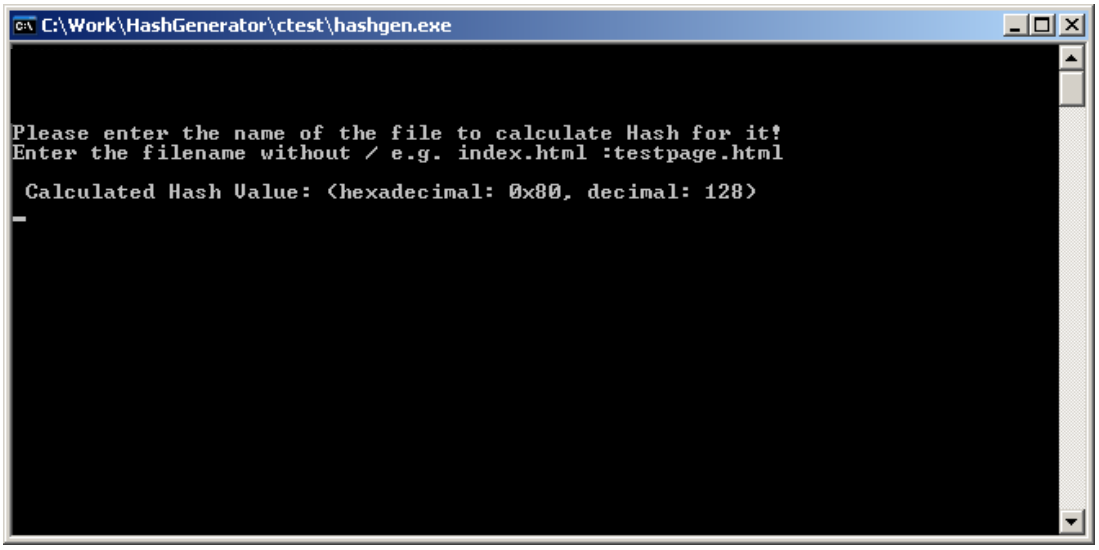
In order to get the correct hash code for a file name the checksum can be calculated by hand or the tool *hashgen.exe* from the CD can be used. When starting hashgen.exe the following screen will appear

Figure 3



In this screen hashgen.exe asks for a filename in order to calculate the hash. Enter the filename and press enter. For example "testpage.html". The calculated hash will appear on the screen.

Figure 4



4.5 How to build dynamic webpages

In order to deliver the dynamically generated web pages the http header has to be defined first. To do this the function https_generateheader() can be used or an own header can be used. A typical http header could look like this:

Listing 7

```
HTTP/1.0 200 OK\r\n
Last-modified: Fri, 18 Oct 2002 12:04:32 GMT\r\n
Server: ESERV-10/1.0\nContent-type: text/html\r\n
Content-length: 400\r\n\r\n
```

After the header the normal html webpage is delivered. To assemble your dynamic webpage (i.e. to provide near live information) several methods are possible.

- The first method is to build the complete webpage in memory and send it. This is the fastest method but needs much memory. To prevent stack overflow the stack size needs to be at least 3000 bytes. To change the stack size the option STACK_SYS_SIZE in the file start.asm needs to be changed.
- The second method is to generate only parts of the web page in RAM. This can be done by define a working array like myhtmlpart[2000]. Thus it is easy to build the webpage with some sprintf() functions pointed to this array. After the array is full it can be passed to the TCP layer. The TCP layer builds some option fields around the data and sends it out.
- The third possible method is to separate the static and the dynamic data of the webpage. Newer browsers support the xmlHttpReq message. This message is the base for most of the so called AJAX applications. By using JavaScript you can load the static page and fill some placeholders with data from an xml file. This xml file can be small enough to be generated completely in ram while the large html site comes from flash. Also the xml file can be periodically polled which allows to display new data without the need to reload the whole website.

5 Telnet

To communicate directly with the microcontroller a telnet server is available. To connect to the telnet server a telnet client is needed. After connecting to the server the FShell drops out a prompt.

5.1 The Cypress Telnet Server

The Cypress Telnet server runs on top of the OpenTCP stack and provides 4 callback functions for the shell. These functions are used to handle the connections from/to the Telnet server and to manage the command and data channels. To develop a shell these functions can be used to send and receive bytes. To do this the telnetd.h needs to be included the source. The telnetd.h provides following functions:

Listing 8

```
UINT8 telnetd_init();
void telnetd_run(void);
INT8 telnetd_close(void);
INT32 telnetd_eventlistener(INT8, UINT8, UINT32, UINT32 );
INT8 telnetd_sendstring(const char*);
INT8 telnetd_sendcommand(const unsigned char*, UINT8);

void telnetd_registershell(void (*shconnect)(UINT32, UINT32), void
(*shcommand)(char*, UINT32), void (*shdata)(char*, UINT32), void
(*shclose)(UINT32, UINT32));

INT8 telnetd_getsock(void);
```

At first the callback functions have to be registered to the telnet server. This can be done with the telnetd_registershell() function. If a client wants to connect to the FR board the telnet server calls the shconnect() callback function. The parameter value is the IP address and port of the client. If a client disconnects from the telnet server the function shclose() is called. The parameter value is the IP address and port of the client. If the telnet server receives data on the command channel the shcommand() callback is called. The parameter value is a pointer to received characters and length of the array.

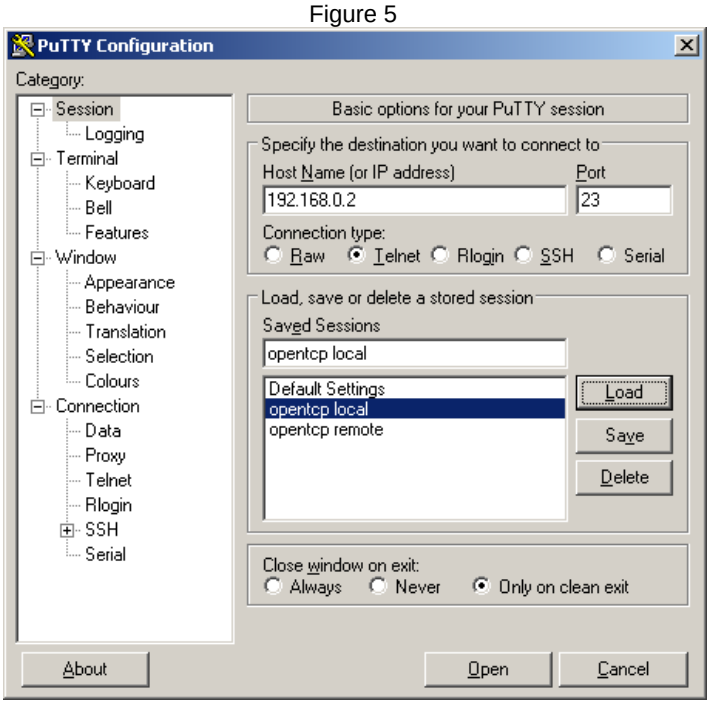
5.2 The Cypress FShell

With the Cypress FShell the FR Ethernet board can be controlled with a standard telnet program. For a connection test the telnet program “Putty” is used. Be sure to make the right settings in putty, otherwise the connection will fail.

5.2.1 Configure putty

To connect to the ADA-FR-Ethernet Putty has to be configured properly. Be sure to choose the connection type “Telnet”. The port number will change to 23. Normally putty starts in SSH mode; this will not work because the FR does not support encryption. In the next field type in the IP address of the ADA-FR-Ethernet board. Normally the IP is 192.168.0.2.

Figure 5

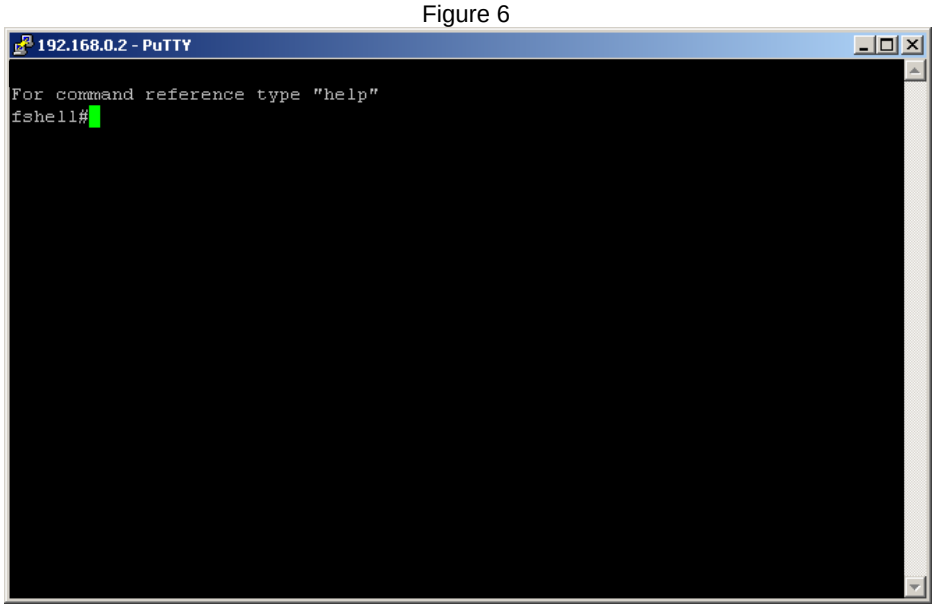


All other options can be set to standard and are not relevant for this test.

5.2.2 Connecting to the FShell

After configuration is done, putty can connect to the board. The following screen shows the Cypress FShell:

Figure 6



5.2.3 The FShell command reference

With the Cypress FShell information about the controller platform can be displayed. To get the help reference type “help” in the command window. Following commands are available:

Table 3

Command	Description
help	Shows the help screen with commands from this table.
sys	Shows the type of processor and some peripherals.
info	Shows the disclaimer text.
ethstats	Shows the Ethernet statistics of the Crystal LAN chip.
exit	Disconnects the client from the telnet server.

The Cypress FShell source code is available in the files fshell.c and fshell.h.

6 How to debug Ethernet Applications

To debug the Ethernet communication a packet monitoring tool can be helpful. One tool to do this is the Wireshark packet sniffer. The software is free and can be downloaded athttp://www.wireshark.org/download.html.

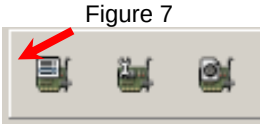
There are several methods to communicate with the ADA-FR-Ethernet board. The first is to connect it directly to your PCs network card. For this option a crossover Ethernet cable is needed.

The second method is to connect the ADA-FR-Ethernet board to the existing network. You can use a standard Ethernet patch cable and connect it to a hub or a switch. This method needs no second network card but it makes debugging a bit more difficult. Because there are a lot more Packets on the network than those you are interested in you should add a filter in the Wireshark packet sniffer. This will help you to concentrate on the packets from the ADA-FR-Ethernet board. But because the ADA-FR-Ethernet board may has to handle a lot of incoming messages which are floating around your network you may get random errors which cannot be replicated. Thus it is better to have a test setup where the ADA-FR-Ethernet board is connected to a separate private network.

6.1 Connection through crossover cable

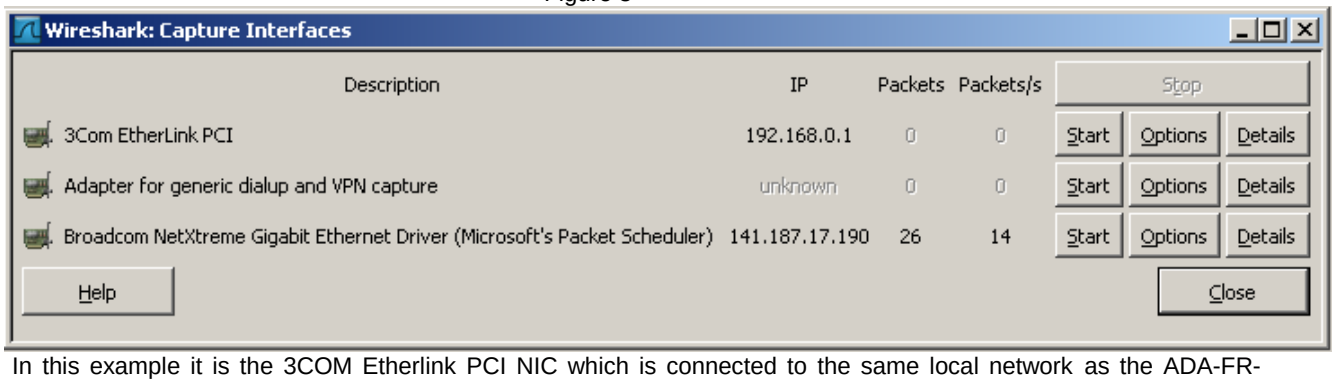
After the ADA-FR-Ethernet board is connected to your development PC Wireshark has to be started in order to capture the packets. Click on the left icon in the taskbar to configure the interfaces:

Figure 7



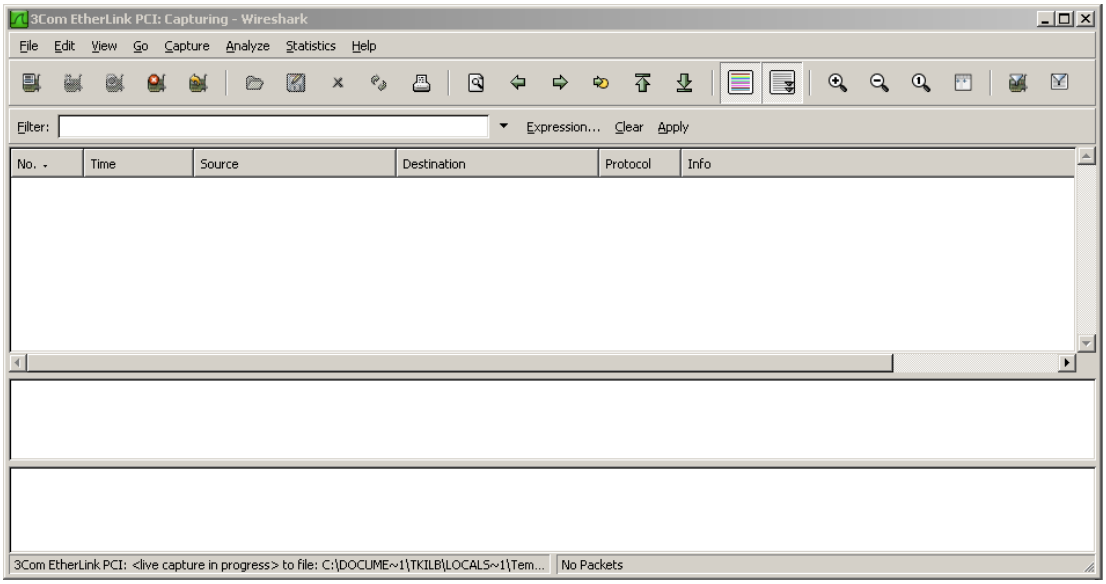
In the next dialog you have to click the start button of the Ethernet NIC which is connected to the ADA-FR-Ethernet board.

Figure 8



In this example it is the 3COM Etherlink PCI NIC which is connected to the same local network as the ADA-FR-Ethernet board.

Figure 9

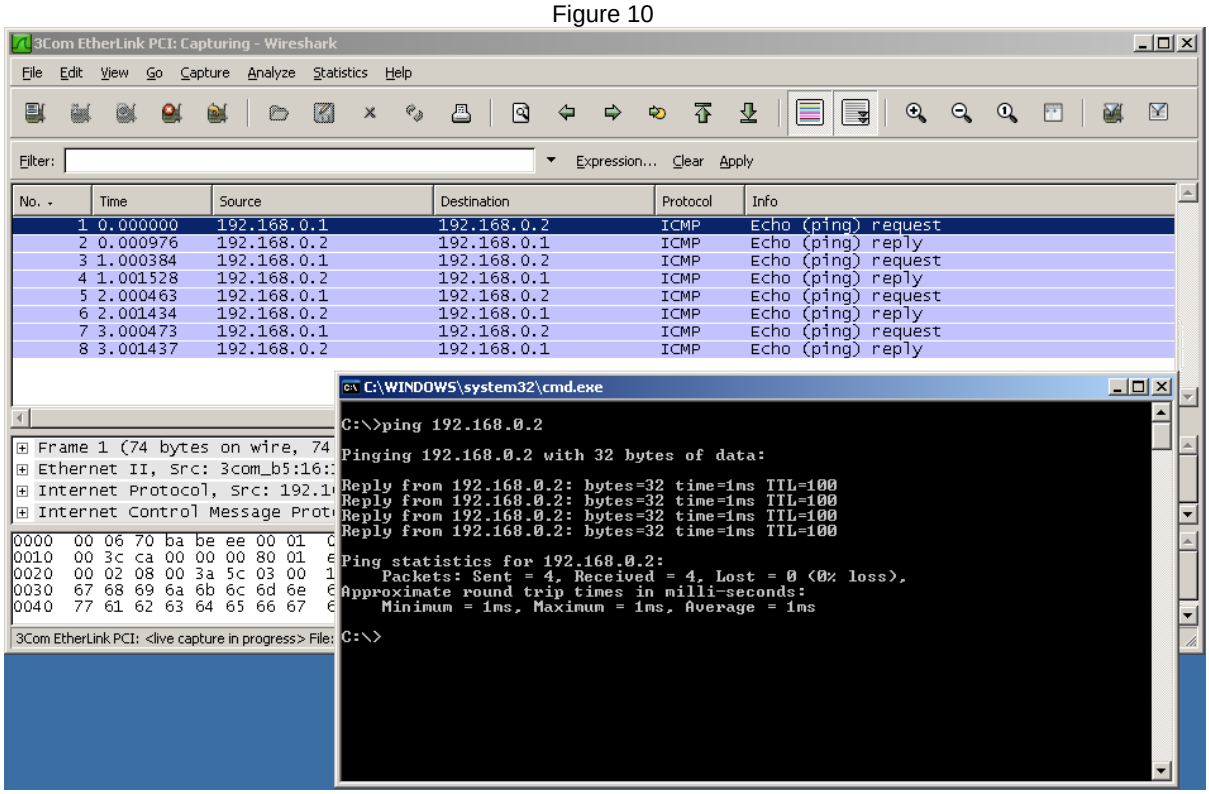


To test if the physical connection is alive check if the orange LED on the ADA-FR-Ethernet board is gleaming. If not, check the power connection of the FR starter kit and the Ethernet connection to your computer. If the orange LED is gleaming the physical connection is available and working. At this point, you can try to ping the board. Open a command window and type following command.

```
ping 192.168.0.2
```

After pressing enter, the ping packets and the answers should appear in the Wireshark window:

Figure 10

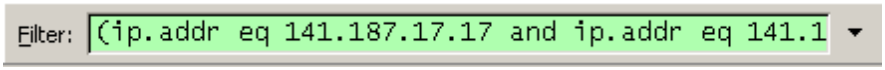


If the request and reply of the ICMP packets from the ADA-FR-Ethernet board is visible, the stack is working.

6.2 Connection through a switch or hub

To connect the ADA-FR-Ethernet directly to the network a standard Ethernet patch cable has to be used. The configuration for the Wireshark tool described above is also valid for the switch/hub connection. Because the network card is also connected to the local network, packets from any addresses of the subnet can be received. Due to this issue it is hard to see the packets coming from the ADA-FR-Ethernet board. The solution is to configure a filter in the Wireshark options:

Figure 11



The filter field is located in the menu bar in Wireshark. To define filtering to your needs type:

Listing 9

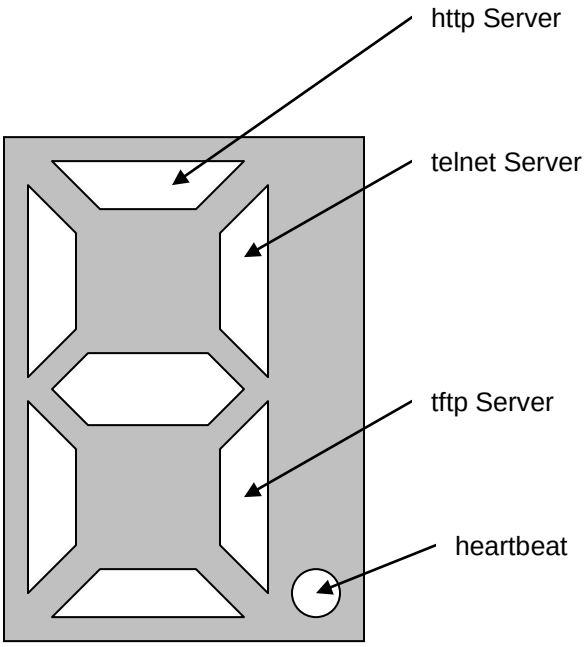
```
(ip.addr eq ADAFRFXETHERNETIP and ip.addr eq MYCOMPUTERIP)
```

Do not forget to press enter after type in the filter rules. Otherwise the filter will not apply on the packets. Now the communication between the computer and the ADA-FR-ETHERNET should be visible.

7 Status Display

To give some direct feedback on the system the seven segment display is used. Following figure shows how the tasks are mapped to the seven segment display. The segments will flicker each time the corresponding task will be executing. The dot point displays a heartbeat which indicates that the system is running.

Figure 12



8 Additional Information

Information about CYPRESS Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software example related to this application note is:

91460_opentcp

Document History

Document Title: AN205383 - FR, MB91460, ADA-FR-Ethernet,OpenTCP

Document Number:002-05383

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	06/27/2008	Markus Heigl First draft
*A	5135517	NOFL	02/12/2016	Converted Spansion Application Note “MCU-AN-300099-E-V10” to Cypress format
*B	5842129	AESATP12	08/02/2017	Updated logo and copyright
*C	6054547	NOFL	02/12/2018	Updated links Updated template

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2008-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.