AN205376

F²MC-16FX, ADA-16FX-ETHERNET, OpenTCP

This application note describes how to implement the OpenTCP stack from Viola Systems on the 16FX Microcontroller. In order to get the OpenTCP stack running the ADA-16FX-ETHERNET kit is necessary

Contents

1 Introduction....1
2 Hardware....2
2.1 Development Setup....2
3 Software....3
3.1 Overview about the OpenTCP Software package....4
3.2 Configuration of the OpenTCP stack....4
3.3 Timer functions....5
3.4 High level Ethernet functions....5
3.5 Low level Ethernet functions....5
3.6 Main routine and supporting functions....8
3.7 Basic software test....10
4 Web server....11
4.1 How web pages are stored in FLASH....11
4.2 How to address the web pages....11
4.3 How to deliver the web pages....12
4.4 What is the hash algorithm....12
4.5 How to build dynamic webpages....13
5 Telnet....14
5.1 The Telnet Server....14
5.2 The FShell....14
6 TFTP Server....17
6.1 Download data from 16FX-Ethernet....17
6.2 Upload data to 16FX Ethernet....18
7 E-Mail....19
7.1 Sending E-Mail via SMTP client....19
7.2 Receiving E-Mail via POP3....20
8 Low level FLASH driver....21
8.1 Low level functions....22
8.2 Filesystem functions....22
8.3 Filesystem Format....23
9 How to setup the OpenTCP Software....24
9.1 Set the correct MAC address....24
9.2 Set the correct IP address and Netmask....25
9.3 Starting the OpenTCP stack....25
10 How to keep track of your Ethernet packets....26
10.1 Connection through crossover cable....26
10.2 Connection through a switch or hub....27
11 Additional Information....28
Document History....29

1 Introduction

This application note describes how to implement the OpenTCP stack from Viola Systems on the 16FX Microcontroller. In order to get the OpenTCP stack running the ADA-16FX-ETHERNET kit is necessary.

The ADA-16FX-ETHERNET is an adapter board for the SK-16FX-EUROSCOPE evaluation board. It features a 10 Mbit/s Ethernet connection and is interfaced via the multiplexed external bus interface.

This product includes software developed by Viola Systems (http://www.violasystems.com/). A big advantage of the OpenTCP stack is that there is a variety of client and server software integrated. Some protocols of the stack are already configured in the demo application. The minimum FLASH amount of the stack is about 27 KByte and 7 KByte of RAM.

This application requires at least basic knowledge about Ethernet, the seven network layers of the Open Systems Interconnection Reference Model and network protocols. It is not the scope of this document to give an Ethernet introduction.

This application note will explain the CS8900 hardware driver, the high-level Ethernet functions, the Timer functions, the software modules shipped with the OpenTCP stack and the new modules provided by Cypress. In chapter 10 you will find instructions on how to set up the debug environment with Wireshark in order to analyze packets dedicated to the ADA-16FX-ETHERNET. For a detailed source code description of the other software modules shipped with the stack there is a good OpenTCP source code reference on the web at http://sourceforge.net/project/showfiles.php?group_id=66182. To convert file names into hash numbers you can use the tool hashgen from the cdrom. This tool is needed when new web pages are placed in the Flash Memory.

2 Hardware

The base for the OpenTCP stack is the 16FX evaluation board SK-16FX-EUROSCOPE. The board is assembled with the MB96F348HS CPU that features 576 KByte FLASH memory and 24 KByte RAM. The controller has an external bus interface that uses multiplexed address and data lines. The Ethernet connection is provided by the ADA-16FX-ETHERNET adapter board.

The starterkit is normally powered by an external power supply. The external power supply can also be used for the ADA-16FX-ETHERNET and SK-16FX-EUROSCOPE in combination. Power over USB is also an option for the SK-16FX-EUROSCOPE.

2.1 Development Setup

For an easy and uncomplicated debugging, a special debug setup is recommended for the hardware. Before working with the ADA-16FX-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-16FX-ETHERNET board.

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	

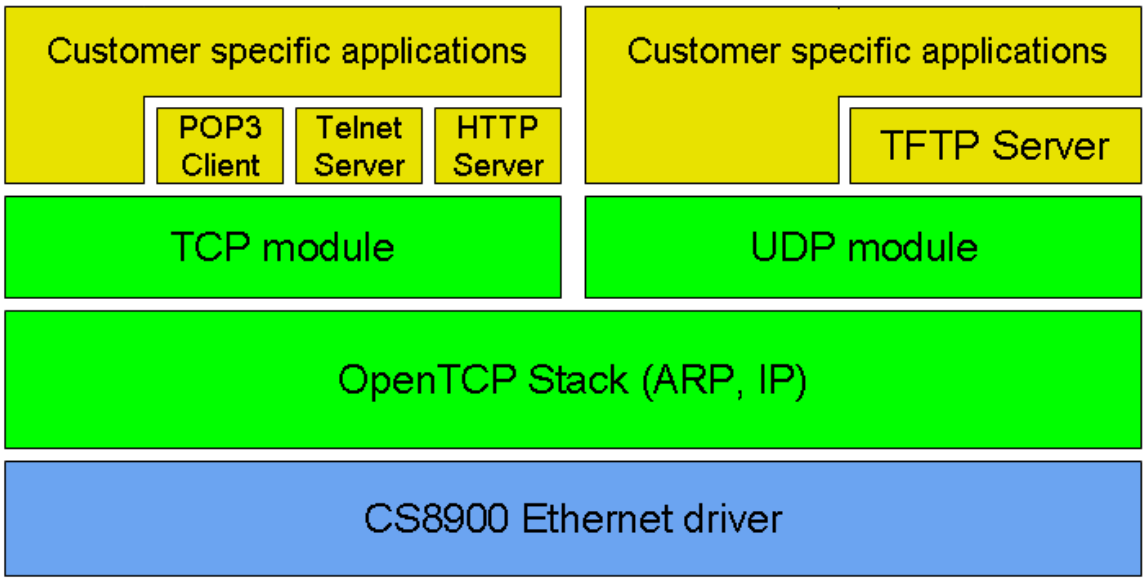
To monitor and analyse the traffic on the local microcontroller network tools like Wireshark can be used. To configure Wireshark please look at chapter 10. These tools capture all packets of a specified network card and show the flowing packets in real time. The software is able to interpret the packets and decodes the bytes corresponding to its protocol.

The advantage of an extra network setup for the Ethernet hardware is that only packets are viewable on the Ethernet that are dedicated to the ADA-16FX-ETHERNET or coming from a board dedicated to the computer.

3 Software

For Ethernet communication, a freely available IP Stack called OpenTCP is used. The OpenTCP Stack is developed from Viola Systems. It is mainly used for 16/32-bit microcontroller platforms. This stack has a little bit more overhead than the uIP stack from Adam Dunkels. Thus, it is not best fitted for 8-bit microcontrollers because of more FLASH and RAM amount. 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 by calling the run functions in the main thread.



The ADA-16FX-ETHERNET is shipped with a precompiled OpenTCP software package. In this package, the httpd server is enabled for demonstration purposes. To test the precompiled package, the 96340_opentcp.mhx file needs to be flashed into the controller. To do this the tool Cypress Flash MCU Programmer can be used. The demo file can be found in the ABS directory of the 96340_opentcp project on the cdrom. After flashing the file into the MCU, you can connect with a web browser to http://192.168.0.2.

3.1 Overview about the OpenTCP Software package

3.1.1 Original OpenTCP Stack

The OpenTCP Stack is downloadable as a compressed software package from http://sourceforge.net/project/showfiles.php?group_id=66182. The package includes only the source code. For the documentation, it is necessary to download the source documentation package. The source documentation package can be found on the same website as the source.

The current version of the OpenTCP stack is 1.0.4.

[..]		<DIR>	03.08.2007 14:23 ----
[arch]		<DIR>	03.08.2007 14:23 -a--
[bootp]		<DIR>	03.08.2007 14:23 -a--
[demo]		<DIR>	03.08.2007 14:23 -a--
[dhcp]		<DIR>	03.08.2007 14:23 -a--
[dns]		<DIR>	03.08.2007 14:23 -a--
[doxygen]		<DIR>	03.08.2007 14:23 -a--
[http]		<DIR>	03.08.2007 14:23 -a--
[include]		<DIR>	03.08.2007 14:23 -a--
[pop3]		<DIR>	03.08.2007 14:23 -a--
[smtp]		<DIR>	03.08.2007 14:23 -a--
[tftp]		<DIR>	03.08.2007 14:23 -a--
arp	c	26.024	03.08.2003 17:32 -a--
Changes	txt	2.254	03.08.2003 18:19 -a--
ethernet	c	22.345	03.08.2003 15:23 -a--
icmp	c	6.610	03.08.2003 15:42 -a--
ip	c	18.474	03.08.2003 15:57 -a--
license	txt	2.436	25.11.2002 16:22 -a--
OpenTCP-1.0.4.src	zip	191.878	03.08.2007 14:22 -a--
system	c	12.272	04.03.2003 17:37 -a--
tcp	c	67.572	03.08.2003 18:14 -a--
timers	c	7.486	25.11.2002 16:22 -a--
udp	c	19.253	03.08.2003 15:52 -a--

Uncompressed OpenTCP folder

There are several subfolders listed. To use the standard functions like ARP, UDP, and TCP only the files in the root directory need to be included. To use the applications shipped with the stack the other directories need to be included too.

3.1.2 OpenTCP Stack with Cypress modifications

In order to get higher usability with the OpenTCP stack, some changes are made and features added. These changes are listed below. The OpenTCP with Cypress modifications can be found in the directory OpenTCP-FME on the cdrom. The following modules are new:

- Telnet Server.
- FShell for the Telnet Server.
- Flash module.
- CS8900 hardware driver.

The following modules have been modified:

- Added new tftp handler to offer file downloads.
- Added new web server code to handle dynamic contents.

3.2 Configuration of the OpenTCP stack

In order to use the stack in the network some parameters have to be set. In the file *help.c* are options to configure the MAC address, IP address, subnet mask and default gateway. More information about stack configuration can be found in chapter 9. In order to get higher performance, the connection count for the web server can be changed. To do this the value NO_OF_HTTP_SESSIONS in the http_server.h must be changed. If you increase the maximum http session variable you have also to increase the NO_OF_TCPSOCKETS in the file tcp_ip.h. But beware of more memory consumption!

3.3 Timer functions

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

The following functions have to be provided by the developer:

```
 // OpenTCP needs
clock_time_t clock_time(void)   // return actual time in milliseconds
void clock_init(void)           // initializes clock

 // additional
__interrupt void RLT0IRQHandler(void) // reload time interrupt routine
void delay_ms(unsigned long delay)    // delays for given milliseconds
```

Actually a clock is implemented with the reload timer. The reload timer division register can be set accordingly, thus every ten milliseconds an interrupt is generated. With every interrupt the global time is increased for 10ms. To get the actual time it is necessary to calculate the milliseconds from the last interrupt till now. That can be done by reading the reload timer counter register.

3.4 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:

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()	Do a while(1) and wait for the watchdog bite.
OS_EnterCritical()	Disable interrupts globally.
OS_ExitCritical()	Enable interrupts globally.

3.5 Low level Ethernet functions

The overview about the low level functions shows the true story of the implementation. The initialization of the ADA-16FX-ETHERNET hardware is done in the `init()` function. This function needs to read and write to the CS8900A in IO-mode, thus a set of IO-mode functions is necessary. After initialization the chip can be accessed in memory mode. To handle the packets with the stack more easily a send and read function using the memory mode, a function for sending and receiving frames from the Ethernet has to be implemented. Last but not least an interrupt will be generated, whenever an event occurs in the Ethernet chip.

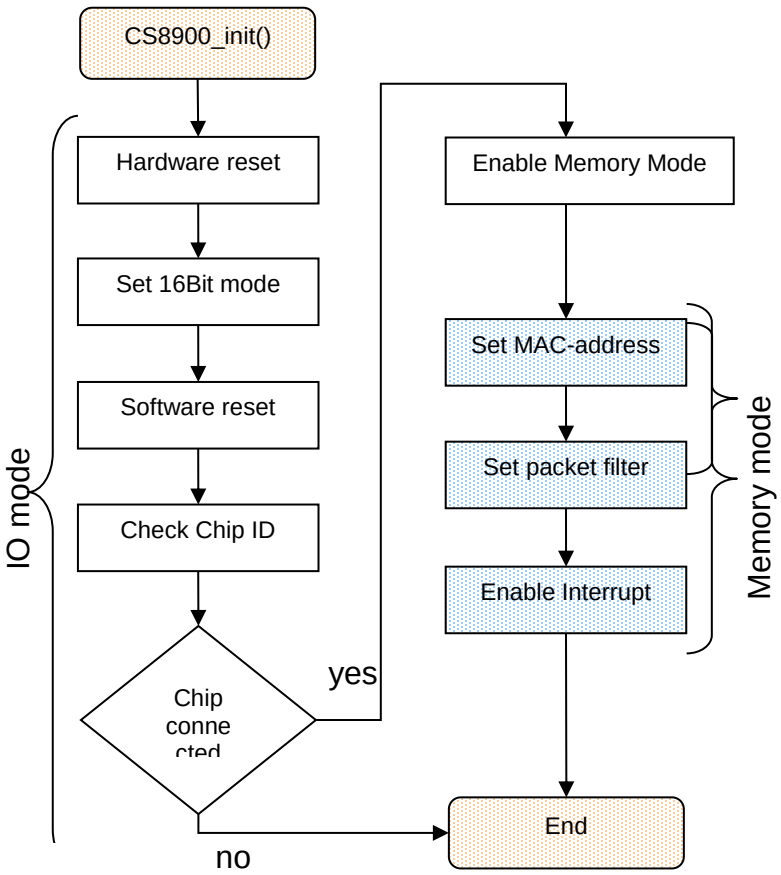
```
  // functions
INT16 CS8900_init(struct netif*);       // init external ethernet device
void CS8900_HWreset(void);              // toggle reset pin of cs8900
void CS8900_Set16BitIO(void);           // set 16Bit access
void CS8900_SetAddrLinesOutput(void);   // set Addr/Data lines to output
void CS8900_SetAddrLinesInput(void);    // set Addr/Data lines to input
void CS8900_SetAddrIO(UINT16);          // save addr in external latch
void CS8900_SetCsIO(void);              // Set Chipselect in IO mode
void CS8900_WriteIO(UINT16, UINT16);    // write data in IO mode
UINT16 CS8900_ReadIO(UINT16);           // read data in IO Mode

INT16 CS8900_read(unsigned char*);      // read frame from CS8900
INT16 CS8900_send(UINT16, const unsigned char*); // send frame over
Ethernet

__interrupt void CS8900IRQHandler(void);  // External Int from CS8900A

 // global variables
int ReceivePacket
```

The picture below shows the initial steps of the hardware initialisation. The set up of the Cirrus CS8900A is mainly made in IO-mode, as mentioned before.



In the initialization process, the MAC-address will be set. This is done in the initialization, because without a valid MAC-address the CS8900A will not work. The packet filter options controls which packets from the Ethernet will be saved inside the CS8900A. This helps reducing the load for the microcontroller.

The Cirrus Chip has 4 Kbytes on-chip SRAM, which is enough for almost three whole Ethernet packets, or many small ones. Whenever the Cirrus has not enough space available to receive a packet it will loose incoming data. Therefore it is important to fetch the Ethernet data quickly and copy it into the microcontroller RAM and process the frames. Processing the received frames could invoke the sending of new frames, and will take some time. Therefore it is impossible to do this inside the interrupt routine. The function CS8900IRQHandler() will only count the incoming RX-events. The function increases the global variable “ReceivePacket” each time, it gets triggered. Thus the variable always shows how many packets are pending for processing inside the Ethernet chip.

The reading of each packet is then done by polling the “ReceivePacket” variable. The function CS8900_read() will place the data inside the global receive buffer and set the receive length variable to the frame length. After a successful download of a packet the “ReceivePacket” variable will be decreased.

The processing of that frame is part of the OpenTCP stack. The stack also works with a polling mechanism. If the ReceivePacket variable is not zero, the stack begins its work and initiates further steps, e.g. sending an answer.

3.6 Main routine and supporting functions

In order to use various on chip resources these must be initialized first. This is done in the main function. The processor relevant start.asm file is not described here and not part of this application note.

First the hardware will be initialized, which is the UART, the interrupts, the clock and the Ethernet chip. Before the initialization of the Ethernet chip can take place, a valid MAC address must be defined. A structure is available that takes all necessary configuration options for the initialization. This structure is filled with the values that are defined in the `netif_init()` function. Both the CS8900 driver and the OpenTCP stack use these values.

Function	Description
void UART_init(int);	init UARTx.
void InitPorts(void);	init port for external interrupt.
void clock_init(void);	init system clock.
void netif_init(struct netif*);	init local ip configuration.
int CS8900_init(struct netif*);	init external Ethernet device.
void arp_init(void);	bring the arp layer up.
void udp_init(void);	bring the udp layer up.
void tcp_init(void);	bring the tcp layer up.
int https_init(void);	start the http server.

The netif structure holds information about the Ethernet device:

```
struct netif {
  LWORD localip;                    // local IP address
  BYTE  localHW;                    // MAC address (Hardware
address)
  LWORD defgw;                      // default gateway
  LWORD netmask;                    // subnet mask
};
```

After the addresses have been specified the hardware initialization function CS8900_init() and then the stack functions arp_init(), udp_init(), tcp_init() can be called. It is recommended that the init functions are called before any application is started. Finally a server application can be initialized like the webserver with https_init().

In the main loop, the ReceivePacket variable is constantly polled. If it is >0 the packets will be processed:

```
if( ReceivePacket > 0 )
{
  NETWORK_CHECK_IF_RECEIVED();

  switch(received_frame.protocol)
  {
    case PROTOCOL_ARP:
      process_arp(&received_frame);
      break;

    case PROTOCOL_IP:
      len = process_ip_in(&received_frame);

      if(len < 0)
        break;

      switch(received_ip_packet.protocol)
      {
          case IP_ICMP:
            process_icmp_in (&received_ip_packet,len);
            break;
          case IP_UDP:
            process_udp_in (&received_ip_packet,len);
            break;
           case IP_TCP:
            process_tcp_in (&received_ip_packet, len);
            break;
          default:
            break;
      }
      break;
    default:
      break;
  }
                /* discard received frame */
NETWORK_RECEIVE_END();
}
```

In this loop the stack looks for incoming packets and passes them to the correct protocol layer.
If an ARP request arrives, the stack calls the process_arp() function. If an IP packet arrives the stack disassembles the packet and runs the function depending on the protocol.

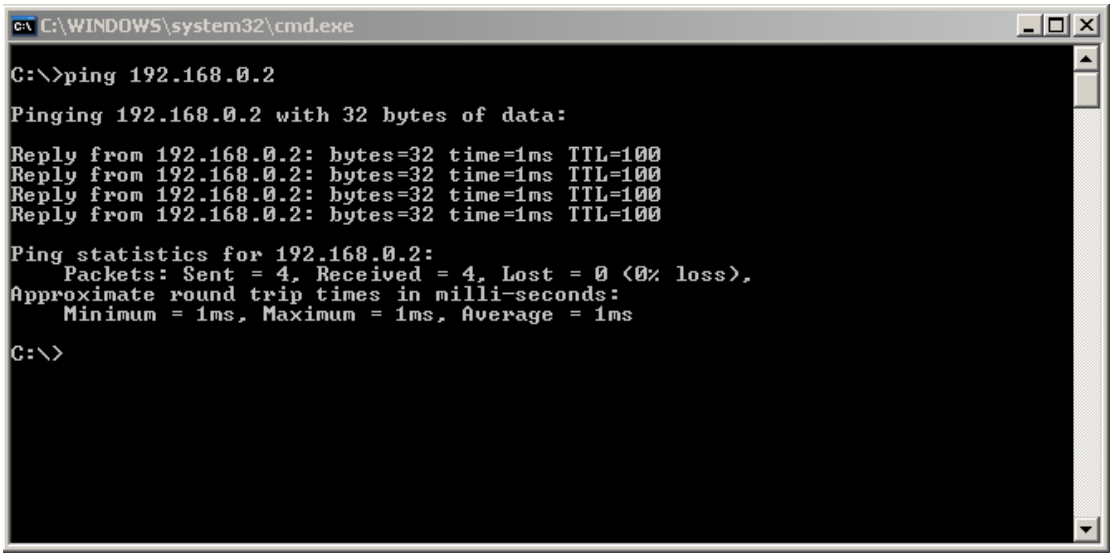
In order to realize timeouts for UDP and TCP it is recommended to implement system timers. The function clock_init() initializes the reload timer. After every timer overflow the stack increments the timers and restarts the hardware timer. The reload timer ticks every 10 ms. For more reference about timers the busytask example realizes a blink led and uses system timers.

To realize the timeouts properly the stack implements polling functions for the protocols. To write client or server applications on top of the OpenTCP stack the init and run functions must be implemented. The init function is called once at startup and the run function is called periodically from the while(1) loop.

```
arp_manage();          /* manage arp cache tables                  */
tcp_poll();            /* manage opened TCP connections
                          (retransmissions, timeouts,...)          */
```

3.7 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-16FX-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.



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:

```
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 hash number 43 it chooses the correct file array from the hash table. The next picture shows a part of the hash table:

```
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 are 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 next picture shows a small code snippet:

```
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);
}
```

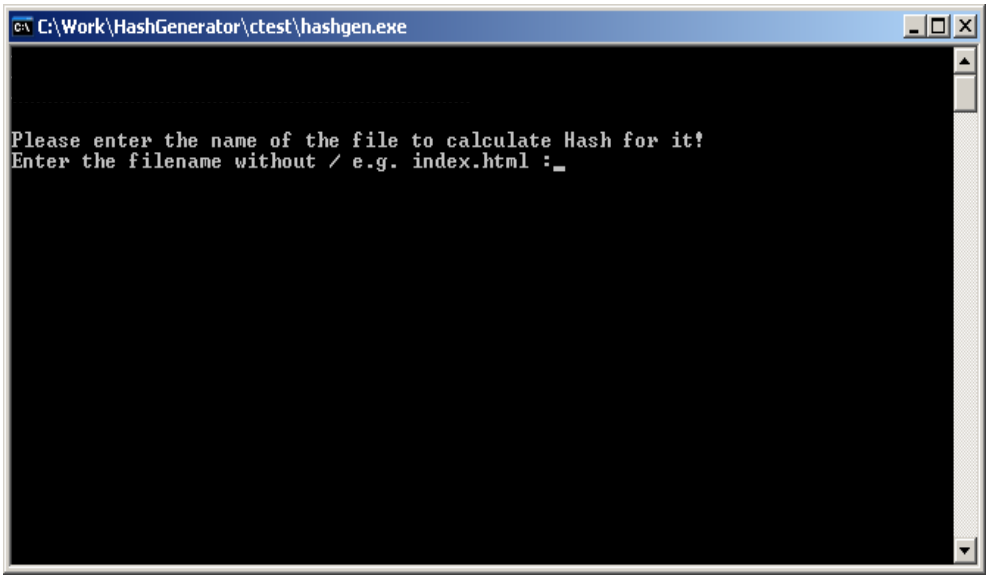
The "`if`" clause identifies the data area. The code in the loop cycle copies the data directly into the Ethernet buffer (the MCU RAM, not the CS8900 RAM!). If the buffer is full the "`if`" clause returns and the stack can send the packet to the receiver. After an ACK has been received, then the next 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 What is the hash algorithm

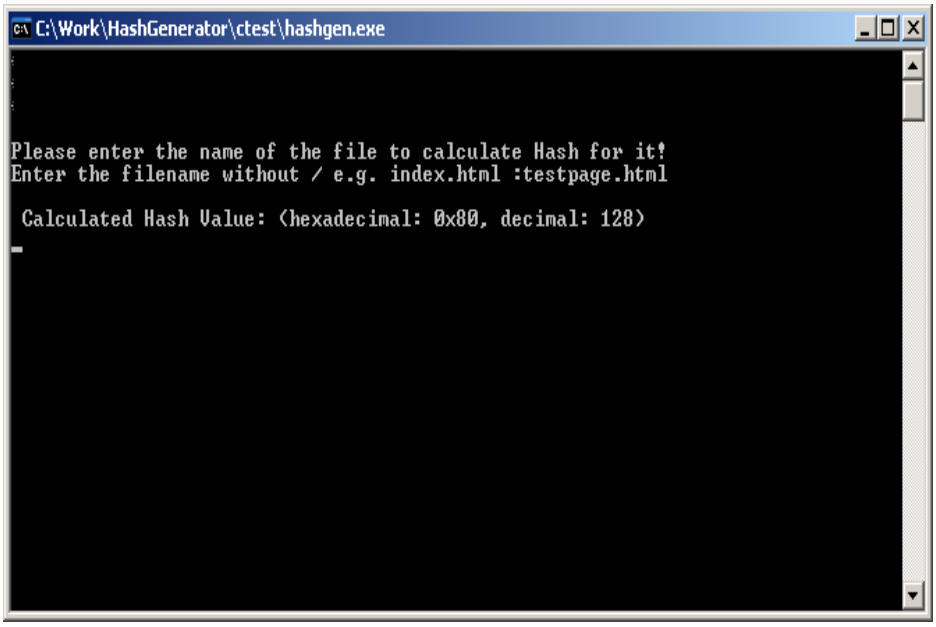
If the web server receives a file request the browser passes the file name via the GET command as a string. Because strings normally are very large and the microcontroller has not very much RAM the web server calculates the hash from "test.html". With this hash the web server can address the files in the FLASH memory. With the hash generator from CD calculation of new hashes are 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 cdrom can be used. When starting hashgen the following screen will appear



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



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:

```
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) two 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.

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 Telnet Server

The 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:

```
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);
```

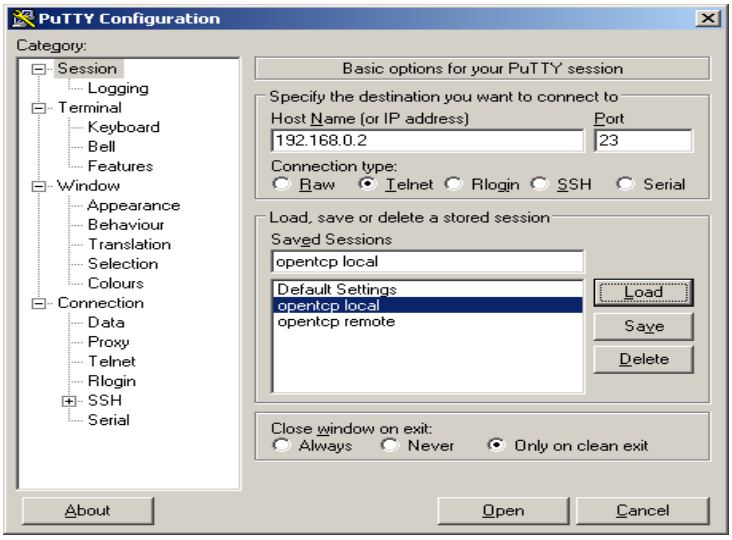
At first the callback functions have to be registered to the telnet server. This can be done with the `telnetd_registershell()` function. If a client wants to connect to the 16FX board the telnet server calls the `shconnect()` callback function. The parameter value is the IP address and port of the client. 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 FShell

With the FShell the 16FX 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

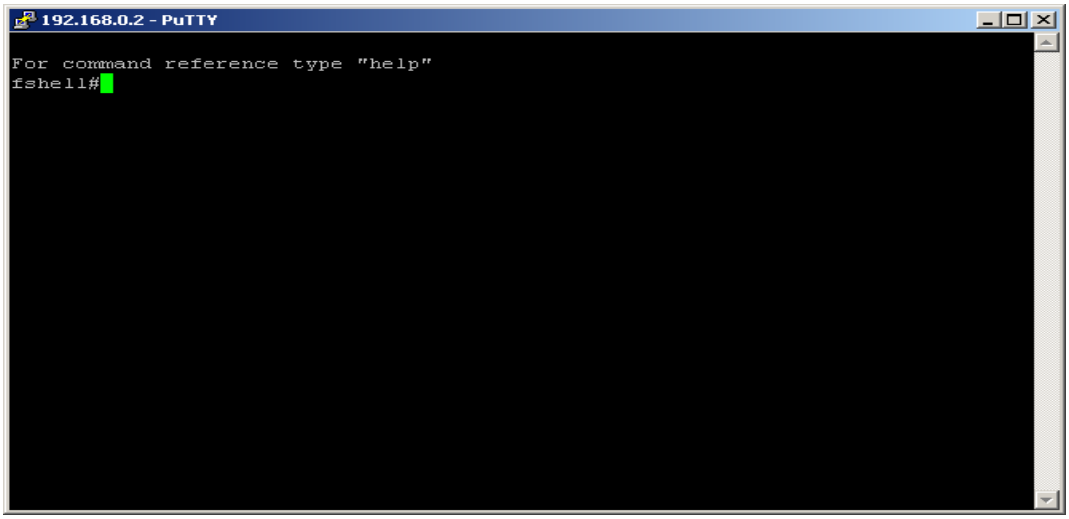
To connect to the ADA-16FX-Ethernet Putty has to be configured properly. Be sure to choose the connection type “Telnet”. The port number will change to 23. Normally putty starts in SSH mode; this will not work because the 16FX does not support encryption. In the next field type in the IP address of the ADA-16FX-Ethernet board. Normally the IP is 192.168.0.2 if it is not changed in help.c.



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 FShell:



5.2.3 The FShell command reference

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

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.
ipserver	Shows the IP address of the telnet server.
ipclient	Shows the IP address of the client from which you are connected.
httpses	Shows active and free http sessions from the http server.
flashinit	Initializes the satellite flash in order to copy files with tftp.
exit	Disconnects the client from the telnet server.

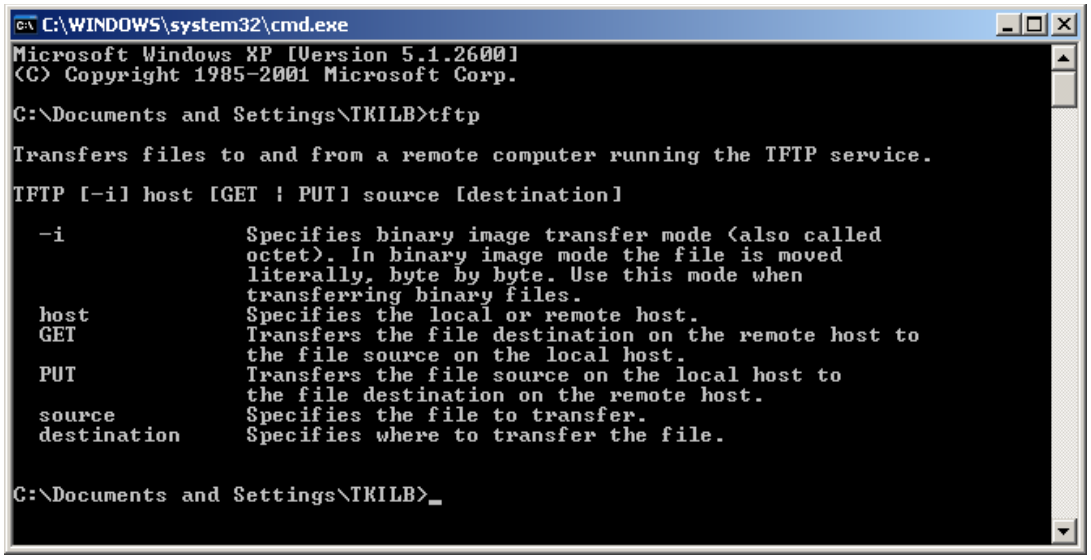
The FShell source code is available in the files fshell.c and fshell.h.

6 TFTP Server

To send and receive files to the 16FX Ethernet board normally an ftp server is needed. Because of lacking memory and nonexistent file system it is not possible to implement ftp in that way. For this task the TFTP protocol is the better way. TFTP stands for trivial file transfer protocol. With this protocol send and receive files with up to 32 MBytes of size are allowed. Because of the functional limitation there is no password check to block the transfer. Be careful working with it. Actual downloads and uploads of files to the 16FX Ethernet board are supported. The data is written into the Satellite Flash immediately.

6.1 Download data from 16FX-Ethernet

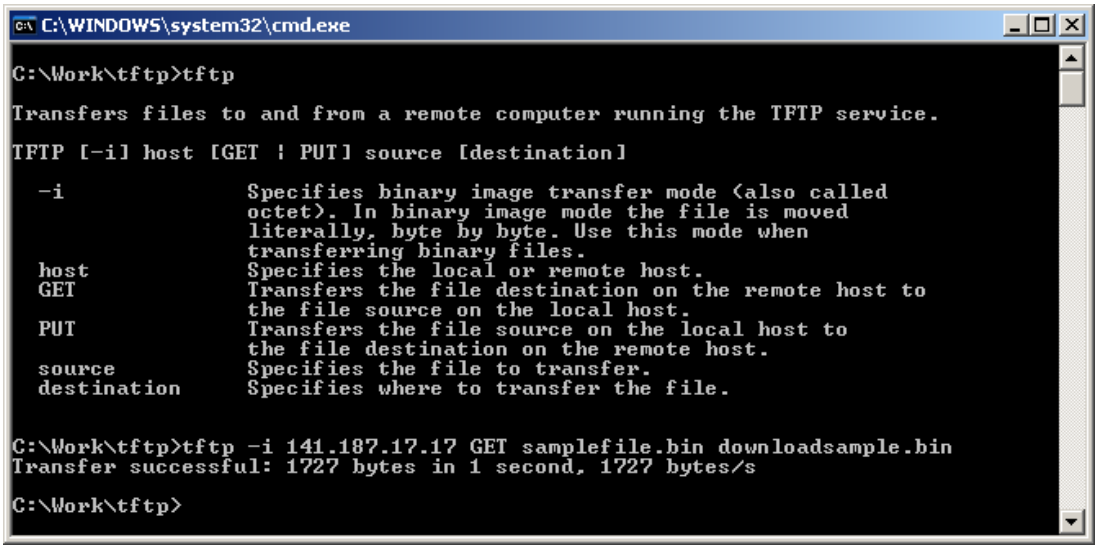
To download data any favorite TFTP client can be used. For example, the following picture shows the Microsoft Windows TFTP client. To start the client the command shell can be opened by clicking Start -> Run and typing “cmd” into the dialog. Type “tftp” in the command window and the tftp options will appear.



If it is desired to download files from the 16FX Ethernet to the PC the option GET has to be chosen. This keyword indicates that a file request is ongoing. To receive the file, type the following line into the command window:

```
tftp -i ipaddr GET samplefile.bin samplefile.bin
```

It is important to set the “-i” switch. The switch stands for binary transport so the packets are byte oriented. Other modes like ascii or mail are obsolete and not available. The first file is the file to download from the remote machine. The second file is the file name to be stored on the local harddisk.



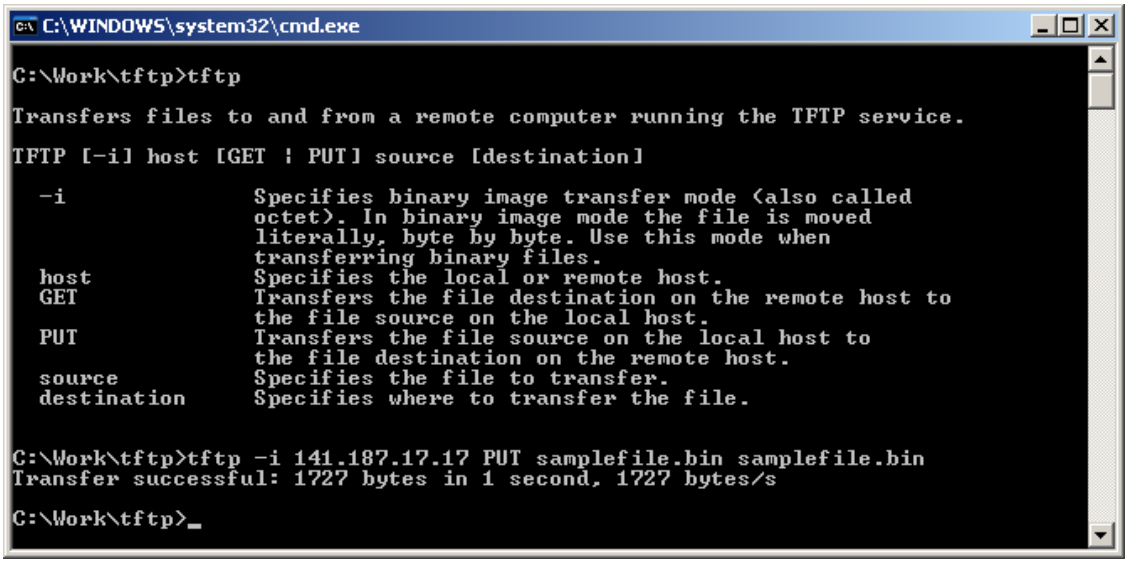
If the text “Transfer successful” appears on the screen, the samplefile.bin has been written properly to the harddisk.

6.2 Upload data to 16FX Ethernet

To upload data to the 16FX Satellite Flash it is necessary to check if there is enough space available. Otherwise the upload will cancel. To tell the TFTP tool an upload is ongoing, the keyword “PUT” should be used.

```
tftp –i ipaddr PUT samplefile.bin myupload.bin
```

The command syntax is nearly the same than downloading data from the MCU.



In the picture a complete file upload to the 16FX is shown. This file is saved as “samplefile.bin” on the FLASH memory. But beware, there is no memory buffer used. If the data transfer interrupts the Flash memory has to be deleted first, then a new download can be started.

7 E-Mail

Sometimes it is necessary to receive and send some status messages from a machine or to control a machine remotely. Therefore, the telnet protocol can be used to control it directly (see chapter 5). But it is easier to send an E-Mail to the target machine and receive status messages by E-Mail. For example, a coffee machine can be controlled at home by sending an E-Mail to it from the office. There are boundless opportunities for using the E-Mail system.

It is very important to mention the Open TCP SMTP module is not a server. It is a client. If E-Mail should be send by the application a dedicated E-Mail server to which the Open TCP stack can deliver the mail is needed. It is not possible to run the SMTP module as a server.

7.1 Sending E-Mail via SMTP Client

7.1.1 Filling the callback functions

To configure the SMTP client the smtpc_callbacks.c file needs to be edited. There are a bunch of callback functions for the SMTP client where E-Mail transmission can be handled properly. The functions are listed in the table below:

Function	Description
void smtpc_error(void);	This is called if there is an error (timeout, loosing of connection, etc.).
void smtpc_allok(void);	This is called if the mail has been successfully transferred to the mail server.
INT8 smtpc_getdomain(UINT8* dbuf);	With this function you can fill the buffer to specify the local domain.
INT8 smtpc_getsender(UINT8* dbuf);	With this function you must return the E-Mail address of the sender.
INT8 smtpc_getreceiver(UINT8* dbuf);	Here you must specify the receiver address.
INT8 smtpc_getsubject(UINT8* dbuf);	With this function you can return a subject.
INT8 smtpc_getdata(UINT8* dbuf, UINT16 buflen);	Here you can send the data of the email. If you are finished, return -1
void smtpc_dataacked(void);	This function is called if previous data has been acked.

7.1.2 The E-Mail transmission process

There is only one function needed to start the SMTP sending process. To activate the sending process the `smtpc_connect()` function needs to be called. The parameter values are the IP address and port of the mail server. The E-Mail sending process will start immediately.

```
smtpc_connect(0xC0A80001, 25);/*IP: 192.168.0.1 Port: 25 (SMTP)*/
```

7.2 Receiving E-Mail via POP3

7.2.1 Filling the callback functions

To receive E-Mails from the POP3 server (Post Office Protocol) a little bit of program code is needed to tell the system what to do with incoming messages. In order to write the program code some functions in the file pop3/pop3c_callback.c must be filled. The functions are listed in the table below.

Function	Description
void pop3c_error(void);	This function is used for error handling like timeout, losing of connection, etc.)
void pop3c_messages(UINT16 msgs);	This function informs about the number of new E-Mails on the server.
INT16 pop3c_msgoffer(UINT16 index, UINT32 msglen, UINT8* from, UINT8* subject);	With this function you can read the sender address and the subject line from the mails on the server.
INT8 pop3c_getusername(UINT8* dbuf);	This function returns the username for the pop3 server.
INT8 pop3c_getpassword(UINT8* dbuf);	This function returns the password for the pop3 server.
void pop3c_allok(void);	This function is called if mail is successfully received.
void pop3c_data(UINT8 data);	This function is called if the mail data arrives.

7.2.2 The E-Mail receive process

To receive E-Mail from the POP3 server the `pop3c_connect()` function needs to be called. The parameter values are the IP address and the port of the E-Mail server running the Post Office Protocol. The E-Mail receiving process will start immediately.

```
pop3c_connect(0xC0A80001, 110)/*IP: 192.168.0.1 Port: 110(POP3)*/
```

8 Low level FLASH driver

The SK-16FX-Euroscope starterkit is available with the Cypress MB96F348HS MCU. This MCU is equipped with two independent Flash memory controllers. The first controller is connected to the main Flash memory and handles accesses used for program code and the second controller is connected to the Satellite Flash memory. To program one of these Flash areas different steps have to be done which are described later. The below table shows the memory map of the complete Flash memory area.

Address	General Sector Name and Size	Flash Memory Type	544 + 32 Kbyte Flash Memory (MB96F348HS)
0xFFFFFF 0xFF0000	SA39 – 64K	Main Flash memory 64-Kbyte Sectors	SA39
0xFEFFFF 0xFE0000	SA38 – 64K		SA38
0xFDFFFF 0xFD0000	SA37 – 64K		SA37
0xFCFFFF 0xFC0000	SA36 – 64K		SA36
0xFBFFFF 0xFB0000	SA35 – 64K		SA35
0xFAFFFF 0xFA0000	SA34 – 64K		SA34
0xF9FFFF 0xF90000	SA33 – 64K		SA33
0xF8FFFF 0xF80000	SA32 – 64K		SA32
0xDF7FFF 0xDF6000	SA3 – 8K	Main Flash memory 8-Kbyte Sectors	SA3
0xDF5FFF 0xDF4000	SA2 – 8K		SA2
0xDF3FFF 0xDF2000	SA1 – 8K		SA1
0xDF1FFF 0xDF0000	SA0 – 8K		SA0
0xDE7FFF 0xDE6000	SB3 – 8K	Satellite Flash memory 8-Kbyte Sectors	SB3
0xDE5FFF 0xDE4000	SB2 – 8K		SB2
0xDE3FFF 0xDE2000	SB1 – 8K		SB1
0xDE1FFF 0xDE0000	SB0 – 8K		SB0

8.1 Low level functions

There is a bunch of low level Flash functions available. With them writing a sector or an area to the Flash is possible. There are also functions available to delete the complete Flash memory or just one sector. In the table below all necessary low level functions are listed.

Function	Description
unsigned char Sat_Flash_write(volatile __far unsigned int *wr_adr, volatile unsigned int wdata)	Writes the 16 bit value wdata to the flash address *wr_adr.
unsigned char Sat_Flash_write_bulk(volatile __far unsigned int *wr_adr, volatile unsigned int *wdata, long int len)	Writes an area of len bytes from the address *wdata to the flash address *wr_adr.
unsigned char Sat_Flash_sector_erase(volatile __far unsigned int *sec_adr)	Deletes the data on the address *sec_adr in the Flash memory.

8.2 Filesystem functions

To write and read files the Flash Filesystem functions are used. These are listed in the table below:

Function	Description
int ffs_getSector(void);	This function checks the Satellite Flash memory for a free sector and returns sector num. If no sector is free, -1 is returned.
int ffs_SetFile(unsigned char len, unsigned char *fname);	This function writes the filename and the hash into the current sector.
int ffs_Write(unsigned int data);	This function writes data into the Satellite Flash memory and increments the address pointer.
int ffs_CloseFile(unsigned int size);	This function closes the actual file and writes the file length into the filesystem.
int ffs_OpenFile(unsigned char *fname, unsigned char len);	This function searches for the appropriate file in the sectors and returns the length. If no file is found -1 is returned.
unsigned char ffs_ReadFile(void);	This function returns the actual data from the address pointer. If -1 is returned, there are no more data.

8.3 File System Format

In order to save more than one file on the 16FX Satellite Flash an information header for each file has to be defined. At the moment it is not possible to copy more than 3 files into the Satellite Flash and a file cannot be greater than 7500 bytes. This restriction comes from the Satellite Flash memory sector size. To implement a real file system, various information structures have to be held in ram. Because there is not enough RAM a static file system is implemented. To copy a file to the Satellite Flash memory with TFTP a check for a free sector has to be done first. This is done in the flash routine described in the Flash function table. The following format for each Satellite Flash sector is used:

File size (2 Bytes)	Reserved (1 Byte)	File Hash (1 Byte)	Reserved (1 Byte)	Filename Length (1 Byte)	Filename (n Bytes, n always even)	Data (n Bytes)

The File size is the length of the data section. The File Hash is a 1 byte value and used by the web server to find the files in the flash. The Filename Length field is an 8 Bit value which describes the length of the Filename string. The maximum filename length is set to 20 (TFTPS_FILENAME_MAXLEN). The Filename field is the original filename. After the Filename the data section begins. If the file name has an odd number of characters, a padding byte between the filename field and the data field is introduced. This padding byte is neither accounted for in the Filename Length nor in the Filesize field.

9 How to Setup the OpenTCP Software

There are only a few steps needed to configure the OpenTCP stack for the application. First the network parameters have to be configured:

- MAC address
- IP address
- Subnet mask
- Default Gateway address

To change the network parameters there is only one file which must be edited. For more information read the next chapter.

9.1 Set the correct MAC address

Because the Crystal LAN CS8900 has no built-in MAC address an own block has to be ordered by yourself. More information is available at http://standards.ieee.org/regauth/llc/index.html. Another chance to get some MAC addresses is to recycle some old network cards. But be careful, a MAC address is unique. It is not possible to use 2 equal MAC addresses at the same time in one network! To set a MAC address for the OpenTCP stack, edit src/help.c:

```
void netif_init(struct netif* localmachine)
{
…
/* MAC Address: 00-06-70-BA-BE-EE */
localmachine->localHW[5] = 0x00;
localmachine->localHW[4] = 0x06;
localmachine->localHW[3] = 0x70;
localmachine->localHW[2] = 0xBA;
localmachine->localHW[1] = 0xBE;
localmachine->localHW[0] = 0xEE;
…
}
```

9.2 Set the correct IP address and Netmask

To communicate with the ADA-16FX-ETHERNET the IP options have to be set properly. To do this, edit the src/help.c:

```
void netif_init(struct netif* localmachine)
{
…
/* IP Address: 192.168.0.2 */
localmachine->localip = 0xC0A80002;
/* Default Gateway: 192.168.0.1 */
localmachine->defgw =    0xC0A80001;
/* Subnet Mask: 255.255.255.0 */
localmachine->netmask = 0xFFFFFF00;
…
}
```

9.3 Starting the OpenTCP stack

To launch the stack with the new IP configuration the project needs to be recompiled first. After compiling, the Cypress MCU Flash Programmer can be used.

10 How to keep track of your Ethernet packets

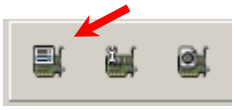
To debug the Ethernet communication a packet monitoring tool has to be used. The best tool to do this is the Wireshark packet sniffer. The client is free and can be downloaded at https://www.wireshark.org/#download.

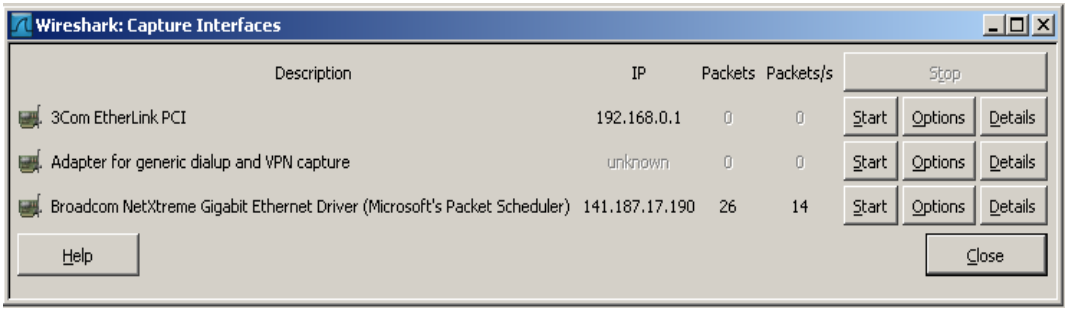
There are two methods to communicate with the ADA-16FX-Ethernet board. The first is to connect it directly to the network card. For this option a crossover Ethernet cable is needed. This method is the easiest to capture your packets.

The second method is to connect the ADA-16FX-Ethernet board to the existing network. Just grab a standard Ethernet patch cable and connect it to a hub or a switch. This method needs no second network card but a filter needs to be configured in the Wireshark packet sniffer.

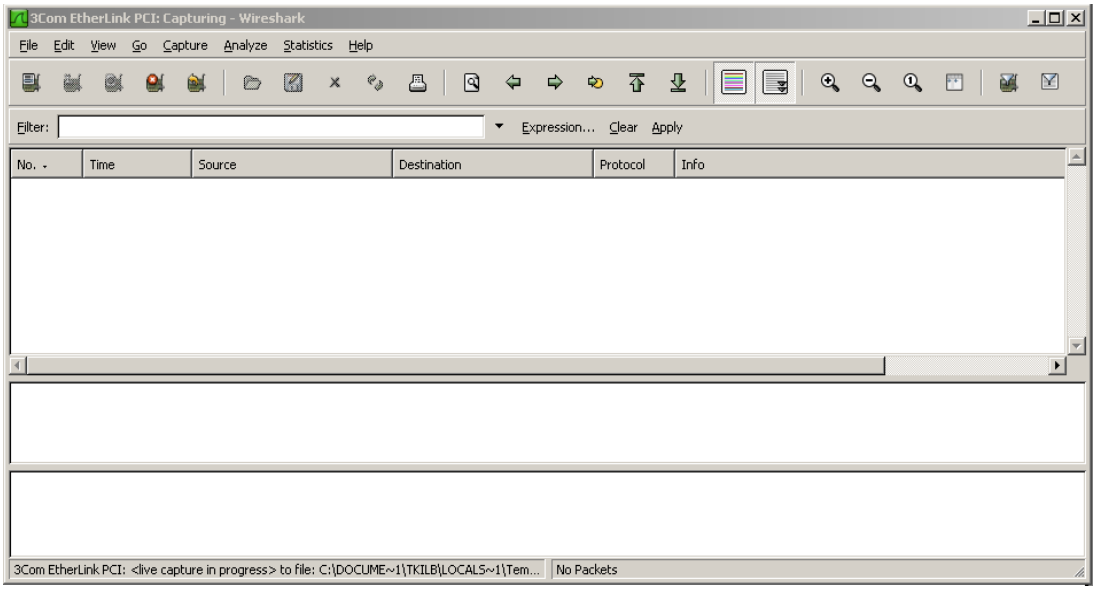
10.1 Connection through crossover cable

At first connect the ADA-16FX-Ethernet board as described in the “First Steps” section to the computer. After connecting the board Wireshark has to be started in order to capture the packets. Click on the left icon in the taskbar to configure the interfaces:

In the next dialog the valid  interface where the ADA-16FX-ETHERNET is connected can be chosen. In that case the 3com Etherlink is available.



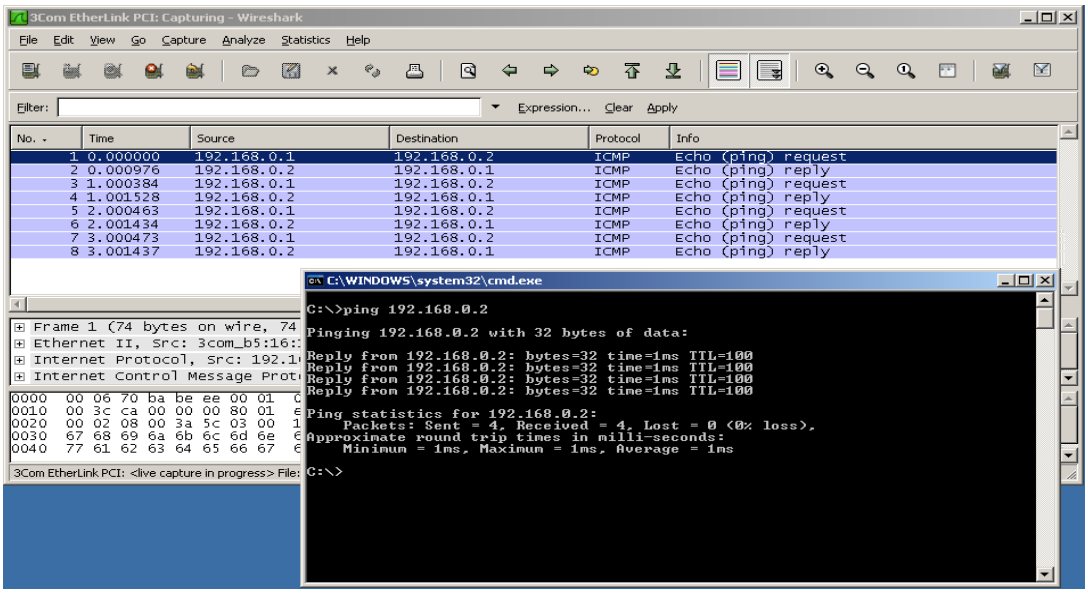
In this dialog click on the start button which is right of the 3com EtherLink label. An empty Wireshark screen will appear. In this screen the captured packets will be listed.



To test if the physical connection is alive check if the green LED on the ADA-16FX-Ethernet board is gleaming. If not, check the power connection of the 16FX starterkit and the Ethernet connection to your computer. If the green LED is gleaming the physical connection is available and working. At this point, a ping test should be done.

```
ping 192.168.0.2
```

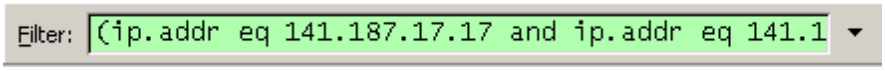
After pressing enter some packets should appear in the Wireshark window:



If the request and reply of the ICMP packets from the ADA-16FX-Ethernet board is visible, then the stack is working. Congratulations!

10.2 Connection through a switch or hub

To connect the ADA-16FX-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-16FX-Ethernet board. The solution is to configure a filter in the Wireshark options:



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

```
(ip.addr eq MYCOMPUTERIP and ip.addr eq ADA16FXETHERNETIP)
```

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-16FX-ETHERNET should be visible.

11 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:

96340_opentcp

It can be found on the following Internet page:

http://mcu.emea.fujitsu.com/mcu_product/mcu_all_software.htm

Information about the ADA-16FX-ETHERNET is available on the following Internet page:

http://www.cypress.com/16fx

Document History

Document Title: AN205376 - F²MC-16FX, ADA-16FX-ETHERNET, OpenTCP

Document Number: 002-05376

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	01/11/2008	V1.0, TKi, First Release
			02/11/2008	V1.1, PHu: removed “internal” marks; typos fixed; update file system header description
			02/21/2008	V1.2, PHu: update project name; add chapter “Additional Information”
*A	5067207	MKEA	05/18/2016	Converted Spansion Application Note “MCU-AN-300086-E-V12” to Cypress format
*B	5869121	AESATP12	08/31/2017	Updated logo and copyright.
*C	6070279	NOFL	01/17/2018	Sunset Review Migrated to new template Updated links

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

Forums | WICED IOT Forums | 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.