Freescale Semiconductor
Application Note

Document Number: AN3779
Rev. 0, 09/2008

Ethernet Plus USB Applications Based on the MCF52259

HTTP/TFTP Server Plus USB Mass Storage Host

by: Li Kan
32-Bit Application Team

1 Introduction

This application note describes implementation of a HTTP/TFTP server based on the ColdFire MCF5225x processor with a USB file system where web pages or other files are stored. It details how to integrate a USB stack with a NicheTask and accessing the USB file system through a HTTP/TFTP server.

2 MCF52259

The MCF52259 is a highly-integrated 32-bit microcontroller based on the V2 ColdFire micro-architecture. Featuring 64 Kbytes of internal SRAM, 512 Kbytes of flash memory, a fast Ethernet controller, a USB on-the-go controller, an external bus interface, four 32-bit timers, a 4-channel DMA controller, two IIC modules, three UARTs, and a queued SPI. The MCF52259 family has been designed for general-purpose industrial control applications.

Contents

1 Introduction 1
2 MCF52259 1
3 Stack 2
3.1 Interniche 2
3.2 CMX USB-Lite 2
4 HTTP Server Implementation 2
4.1 Project Archtecture 2
4.2 Implementation 3
4.3 Running HTTP Web Server on the MCF52259EVB 8
4.4 Connecting to the Web Page on the USB Flash . . . 8
5 TFTP Server Implementation 10
5.1 Software Architecture 10
5.2 Implementation 10
5.3 Project Compile 12
5.4 Running the TFTP Server on the MCF52259EVB. 12
5.5 Uploading a File to TFTP Server 13
5.6 Downloading a File from TFTP Server 14
6 Conclusion 15

© Freescale Semiconductor, Inc., 2008. All rights reserved.

freescale™ semiconductor

3 Stack

The applications detailed in this document use the fast ethernet controller and USB on-the-go controller receiving full support from the InterNiche and CMX.

3.1 InterNiche

This ColdFire® TCP/IP stack is a public source stack provided for use with the ColdFire line of processors. It can be divided into two parts; a mini-TCP layer library and a mini-IP layer library. It also includes a virtual file system (VFS) that supports the american standard code for information interchange (ASCII) and binary data, integrated with a round-robin tasking system named NicheTask. For more details on the ColdFire TCP/IP stack refer to application notes *AN3470—ColdFire TCP/UDP/IP Stack and RTOS* and *AN3455—ColdFire Lite HTTP Server.*

3.2 CMX USB-Lite

The CMX USB-Lite is provided by Freescale and CMX for ColdFire and S08 USB microcontrollers. It supports a USB device, host, and on-the-go (OTG) functionality to meet various design requirements. Included are high-level class drivers for keyboards, mouse devices, and generic human interface devices (HID). Communication device classes (CDC) to universal asynchronous receivers and transmitters (UART) provide communication between UART and USB (mass storage demos for the host mode). The stacks can be accessed at www.freescale.com/USB.

4 HTTP Server Implementation

4.1 Project Architecture

This project is created with CodeWarrior 7.0. The architecture is listed in Table 1.

Table 1. HTTP Web Server Project Architecture

Module	Description
CodeWarrior specific	Linker file for the M52259EVB.
ColdFirelite	InterNiche stack containing multiple folders for the Ethernet TCP/IP stack, supported protocols, and RTOS processor independent files.
Common	Standard C functions and UART input and output support.
CPU	Contains respective processor dependent files.
Drivers	Drivers for MII interface.
Freescale_Web_Server	Applications for HTTP web server.
Project files	Main and interrupts source files.
CMXUSB_LITE	CMX USB stack containing multiple folders for USB mass host support.

4.2 Implementation

This project is implemented based on the *MCF52259EVB*.

4.2.1 Integration of InterNiche and CMX USB Stack

In this project InterNiche is used to manage network protocols and the CMX stack is used to manage file operations based on the USB file system. The key point of this project is to combine the two stacks to work concurrently.

4.2.1.1 Creating usb-mass-host-task.c for NicheTask

The InterNiche also provides a RTOS named NicheTask which is a cooperative multi-tasking scheduler. In this architecture each task has its own call-stack. Voluntarily manages control back to the master scheduler. The integrating method is to invoke APIs from CMX stack as tasks through the NicheTask. This is achieved by a source file named usb-mass-host-task.c where a state machine is created for a USB mass host. This file provides several APIs for a NicheTask listed in Table 2.

Table 2. CMX APIs for InterNiche

Function	Description
void demo_process(void)	State machine for the usb mass host
int cmd_dir(void * pio)	Commands for struct menu_op* usbmassmenu
int cmd_dump(void * pio)	
int cmd_type(void * pio)	
TK_ENTRY(tk_cmxmasshosttask)	Task object defined by nicheTask, using APIs from CMX-USB-LITE and InterNiche
void create_cmxusb_task(void)	Add CMX task the NicheTask's TCB table and gets executed once per loop

4.2.1.2 Invoking APIs from the usb-mass-host-task.c in NicheTask

In usb-mass-host-task.c a function named create_cmxusb_task() is created to support NicheTask. It is invoked in the task object tk_keyboard that supports the standard I/O in NicheTask and added to the task control block (TCB) table after reset.

```
TK_ENTRY(tk_keyboard)
{
   for (;;)
   {
      TK_SLEEP(1);                  /* make keyboard yield some time */
      kbdio();                      /* let Iniche menu routines poll for char */
      keyboard_wakes++;             /* count wakeups */
      #ifdef MCF52259
      if((usbmst_attach == 1)&&(usbtsk_created == 0))
      {
                create_cmxusb_task();
                usbtsk_created = 1;
      }
      else if((usbmst_attach == 0)&&(usbtsk_created == 1))
```

```
      {
                  tk_kill(to_cmxmasshosttask);
                  usbtsk_created = 0;
      }
      #endif
      if (net_system_exit)
         break;
   }
   TK_RETURN_OK();
}
```

4.2.1.3 Adding a Timer for Integration from CMX and OS

NicheTask tk_sleep(tick) is used to delay tasks and in the CMX host_ms_delay(ms) is used for almost the same purpose. The time delayed is different between them. One tick equals 5 ms. For integration, host_ms_delay(ms) is replaced with tk_sleep(tick) and modified in the usb_host.c.

```
void host_ms_delay(hcc_u32 delay)
{
 #ifdef MCF52259
  delay/=5;       // 5 ms is used for 1 tick
  delay+=1;       //there might be a decimal fraction cut as 0 after the division, add 1
  tk_sleep(delay);
  #else
  start_mS_timer((hcc_u16)delay);
  do {
  } while(!check_mS_timer());
  #endif
}
```

4.2.2 Adding File Operation for the USB File Type

The Freescale_Web_Server module file operation supports only html and text files. In this project web pages can be stored in the USB device. Therefore, a new file type definition is added in freescale_http_server.h that supports a USB file type named FILE_TYPE_DYANMIC3.

```
// File types scanned for dynamic HTML content
#define FILE_TYPE_DYANMIC1                  'html'
#define FILE_TYPE_DYANMIC2                  'text'
#ifdef MCF52259
#define FILE_TYPE_DYANMIC3                  'usbf'
#endif
```

The function emg_web_open() is used to open web pages. Codes are now inserted in it to support the new file type defined as FILE_TYPE_DYANMIC3. To avoid destroying the original code structure, the inserted codes are enclosed by a macro defined as MCF52259_MST.

```
void emg_web_open( unsigned char session, unsigned char *filename )
{
…
#ifdef MCF52259_MST
if(usbmst_attach == 1)
{
  F_FIND find;
  int ret;
  volatile unsigned char* fname;
```

```
  F_FILE *file;

  if(filename[0] == 0)
    fname = (volatile unsigned char*)"index.htm";
  else
    fname = filename;
#if HTTP_VERBOSE>4
  printf("open session %d, %s\n\r",session,fname);
#endif
  file=f_open((const char*)fname, "r");
  if(file!=0)
  {
  freescale_http_sessions[session].file_type = FILE_TYPE_DYANMIC3;
  freescale_http_sessions[session].file_size = f_filelength((const char*)fname);
  freescale_http_sessions[session].file_pointer =(void *)file; //used here is a file pointer
to pass the file handle
  freescale_http_sessions[session].file_index = 0;
  freescale_http_sessions[session].state = EMG_HTTP_STATE_SEND_FILE;
  found = 1;
  return;
}
}//if(usbmst_attach == 1)
#endif
…
}
```

The html/text file read operation is supported by a function named emg_web_read(). A function named usb_web_read() is created to support reading data from a USB file. These functions are located in freescale_file_api.c.

```
unsigned long usb_web_read( unsigned char session, unsigned char *buffer_out, unsigned long
max_bytestoread )
{
        unsigned long     i, k, j;
        unsigned long     read_index, bytes_to_read, last_read, read_size;
        unsigned char     *data;
        unsigned long r;

if(freescale_http_sessions[session].file_index>=freescale_http_sessions[session].file_size )
        {
                return(0);
        }

        if( (freescale_http_sessions[session].file_index + max_bytestoread) >
                freescale_http_sessions[session].file_size )
        {
                max_bytestoread = freescale_http_sessions[session].file_size -

freescale_http_sessions[session].file_index;
        }


r=f_read(buffer_out,1,max_bytestoread,(F_FILE*)(freescale_http_sessions[session].file_pointer
));
        //#if HTTP_VERBOSE>4
        printf("session %d read %d\n\r",session,r);
        //#endif
        freescale_http_sessions[session].file_index+=r;
```

```
        return r;
}
#endif
```

4.2.3 Adding a USB Detecting Device in CMX

This HTTP web server also contains a web page in the internal flash. The application can also provide a web page in case there is no USB device attached. An application knows if a USB device is attached by adding a variable named usbmst_attach in the USB interrupt service routine, this tells the application that a USB device is attached or not.

```
extern unsigned char usbmst_attach;
__declspec(interrupt)
void usb_it_handler(void)//
{
  unsigned char istr;
  /* Save irq USB status. */
  //istr=MCF_USB_INT_STAT;
  if(MCF_USB_INT_STAT & MCF_USB_INT_STAT_ATTACH)
  {
        MCF_USB_INT_ENB &= ~MCF_USB_INT_ENB_ATTACH;//clear interrupt flag
        usbmst_attach = 1;         //tell applications that a usb flash is attached
  }
}
```

The device attach interrupt is disabled after reset and needs to be enabled in main(). To achieve this, two registers need be configured. One is MCF_USB_INT_STAT, the other is MCF_USB_INT_ENB.

```
int main(void)
{
…
MCF_USB_INT_STAT = MCF_USB_INT_STAT_ATTACH; //write to clear it
MCF_USB_INT_ENB |= MCF_USB_INT_ENB_ATTACH;
…
}
```

4.2.4 Displaying Sensor Data and Network Status on a Web Page

This project is implemented on the MCF52259EVB. On the board there is a rheostat connected to the ADC port that can be used to collect data and provide the web server with it. The server also places network statistic data on the web page. The codes to collect data are located in a function named evb_specific_collect_sensor_data() that can be referenced in MCF52259_evb.c. The ADC collecting data is achieved by a function named read_AD(). Information about the network status comes from a data structure ip_mib used by the InterNiche stack to store IP status for simple network management protocol (SNMP).

```
void  evb_specific_collect_sensor_data(void) //FSL new function for evb specific implementation
{
        …
        html_vars[3] = read_AD(0);
        html_vars_flags[3] = 1;

        html_vars[4] = read_AD(1);
        html_vars_flags[4] = 1;
```

```
        html_vars[5] = read_AD(2);
        html_vars_flags[5] = 1;

        html_vars[6] = read_AD(3);
        html_vars_flags[6] = 1;

        html_vars[7] = read_AD(4);
        html_vars_flags[7] = 1;

        html_vars[8] = read_AD(5);
        html_vars_flags[8] = 1;

        html_vars[9] = read_AD(6);
        html_vars_flags[9] = 1;

        html_vars[10] = read_AD(7);
        html_vars_flags[10] = 1;

        …

        html_vars[14] = ip_mib.ipInReceives;        /* total received datagrams (bad too) */
        html_vars_flags[14] = 1;

        html_vars[15] = ip_mib.ipInHdrErrors;       /* Header Err (xsum, ver, ttl, etc) */
        html_vars_flags[15] = 1;

        html_vars[16] = ip_mib.ipInAddrErrors;      /* nonsense IP addresses */
        html_vars_flags[16] = 1;

        html_vars[17] = ip_mib.ipUnknownProtos;     /* unknown protocol types */
        html_vars_flags[17] = 1;

        html_vars[18] = ip_mib.ipInDelivers;        /* delivered receive packets */
        html_vars_flags[18] = 1;

        html_vars[19] = ip_mib.ipOutRequests;       /* sends (not including routed) */
        html_vars_flags[19] = 1;

        html_vars[20] = ip_mib.ipOutDiscards;       /* sends dropped (no buffer) */
        html_vars_flags[20] = 1;

        html_vars[21] = ip_mib.ipOutNoRoutes;       /* dropped, does not route */
        html_vars_flags[21] = 1;
}
```

4.2.5 Project Compiling

Modifications listed in "Section 4.2.1, 4.2.2 and 4.2.3" are inserted in the original codes. To avoid destroying the original code structure they are all enclosed by a macro named MCF52259. Please predefine this macro before compiling the project.

```
#define MCF52259
```

4.3 Running HTTP Web Server on the MCF52259EVB

Before running the HTTP web server, the evaluation board (EVB) must be connected to a PC through a UART0 port. The baudrate of the HyperTerminal must be set to 115200-8N1. After power on reset, information appears as in Figure 1.

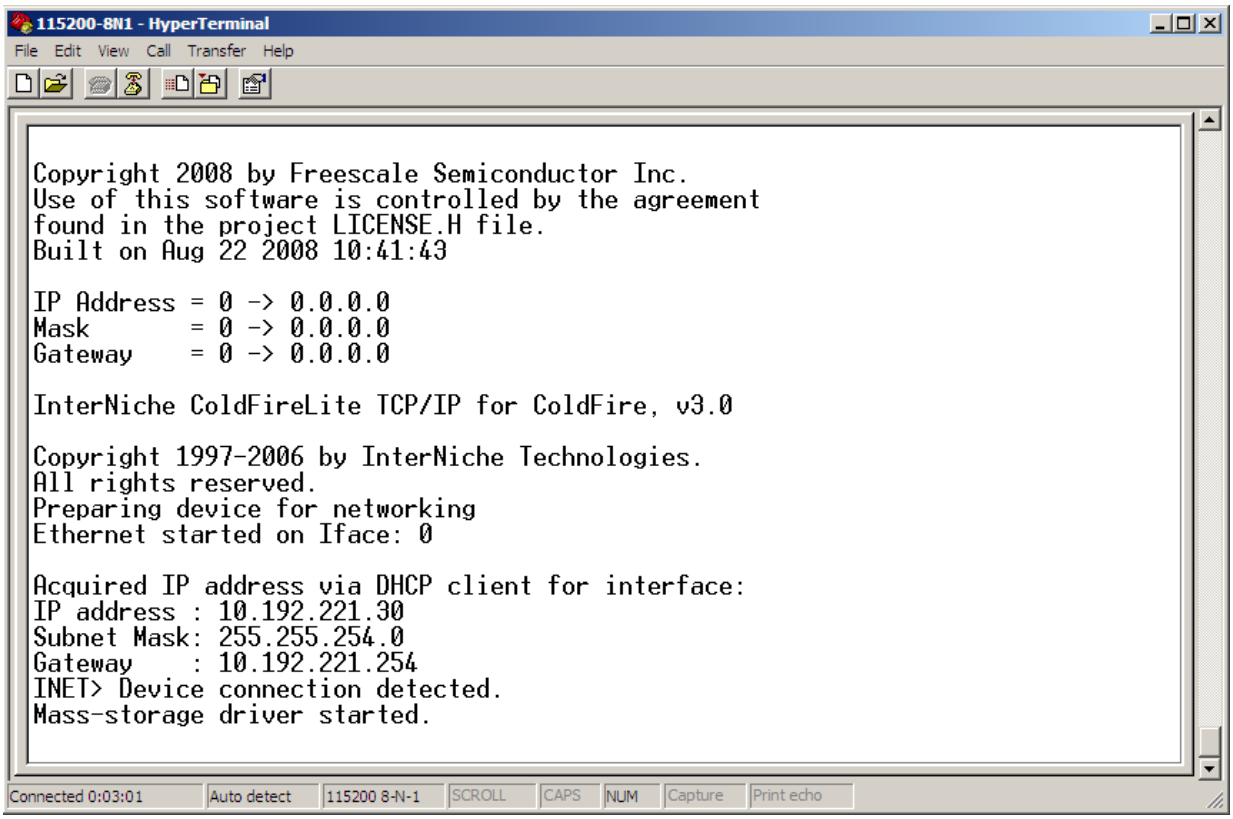

Figure 1. HyperTerminal

The HTTP web server also has a DHCP client, therefore it can automatically obtain an IP address. In Figure 1 the IP address for the server is 10.192.221.30. If a USB device is plugged in, information about this is output.

4.4 Connecting to the Web Page on the USB Flash

Because the IP address is obtained, the HTTP web server can be visited through Internet Explorer by typing http://10.192.221.30 in the address bar.

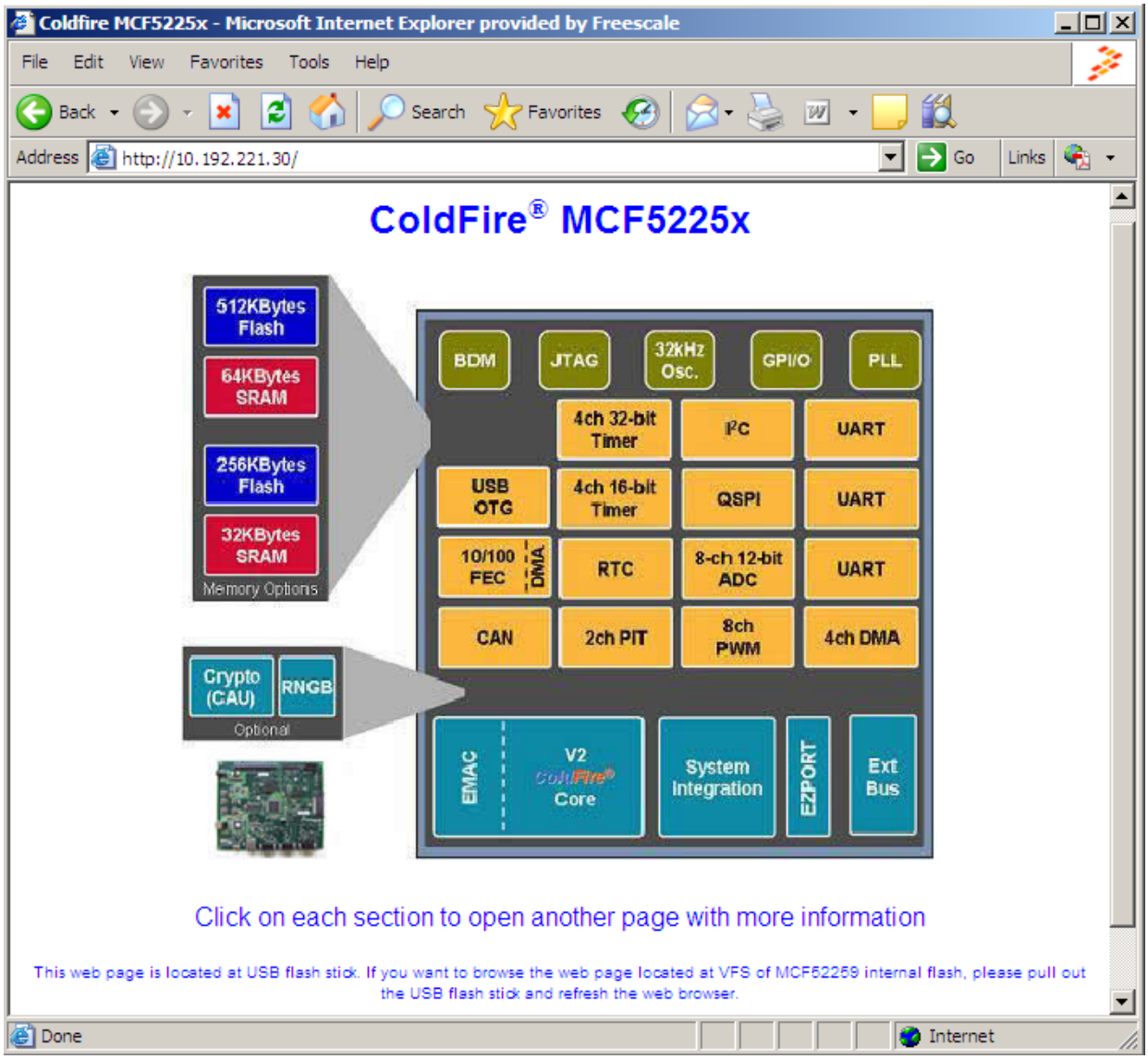

Figure 2. Internet Web Page

5 TFTP Server Implementation

5.1 Software Architecture

This architecture is similar to the HTTP web server. The architecture includes source files supporting TFTP protocol instead of HTTP server support, as in Table 3.

Table 3. TFTP Server Project Architecture

Module	Description
CodeWarrior specific	Linker file for the M52259EVB
ColdFirelite	InterNiche stack containing multiple folders for the Ethernet TCP/IP stack, supported protocols, and RTOS processor independent files
Common	Standard C functions and UART input/output support
CPU	Contains respective processor dependent files
Drivers	Drivers for MII interface
Project files	Main and interrupts source files
CMXUSB_LITE	CMX USB stack containing multiple folders for USB mass host support

5.2 Implementation

This project is also implemented based on the MCF52259EVB. The TFTP server has been already provided by InterNiche. That particular example accesses files stored in the internal flash through VFS and therefore there is a critical file size limitation. In this project, files are stored in a USB device that allows for a larger file size. The key point is to make the TFTP server access the USB files through the CMX file system instead of VFS.

5.2.1 Adding File Operation Support for a USB File

In this project, the APIs regarding file operation are all replaced by the CMX file system.

5.2.1.1 Replacing File Descriptor in VFS with the CMX File

The file descriptor is an important parameter used by the file system. The VFILE is defined for VFS and now replaced by the CMX file system.

```
#ifdef MCF52259_MST
   F_FILE*tf_fd;
#else
   VFILE *     tf_fd;                      /* file descriptor for xfer */
#endif
```

5.2.1.2 Replacing File Open Function in VFS with CMX File

The function vfopen() is used as a file open function in VFS and is replaced by f_open() from the CMX file system.

```
#ifdef MCF52259_MST
#include "thin_usr.h"
#endif
…
#ifdef MCF52259_MST //
           cn->tf_fd = f_open(fname, "w");
           #else
          cn->tf_fd = vfopen(fname, "w");
           #endif
      else if(mode == OCTET)
           #ifdef MCF52259_MST
           cn->tf_fd = f_open(fname, "w");
           #else
          cn->tf_fd = vfopen(fname, "wb");
           #endif
…
if(mode == ASCII)
           #ifdef MCF52259_MST
           cn->tf_fd = f_open(fname,"r");
           #else
          cn->tf_fd = vfopen(fname, "r");
           #endif
      else if(mode == OCTET)
           #ifdef MCF52259_MST
           cn->tf_fd = f_open(fname,"r");
           #else
          cn->tf_fd = vfopen(fname, "rb");
           #endif
      else
          return("Invalid mode");
…
```

5.2.1.3 Replacing File Close Function in VFS with CMX File

The function vfclose() is used as the file close function in VFS and is replaced by f_close() from the CMX file system.

```
if(cn->tf_fd != NULL)
#ifdef MCF52259_MST
      f_close(cn->tf_fd);
         #else
      vfclose(cn->tf_fd);
#endif
```

5.2.1.4 Replacing File Write Function in VFS with CMX File

The function vfwrite() is used as a file write function in VFS, and is replaced by f_write() from the CMX file system.

```
#ifdef MCF52259_MST
   if((int)f_write(data,1,len,cn->tf_fd)!=(int)len)
   #else
   if((int)vfwrite(data, 1, len, cn->tf_fd) != (int)len)
#endif
```

5.2.1.5 Replacing File Read Function in VFS with CMX File

The function vfread() is used as a file read function in VFS and is replaced by f_read() from the CMX file system.

```
/* load file data into tftp buffer */
   if(!cn->tf_NR)                  /* if this is NOT a retry, read in new data */
   {
         #ifdef MCF52259_MST
         bytes = f_read(tfdata->tf_data, 1, NORMLEN, cn->tf_fd);
         #else
      bytes = vfread(tfdata->tf_data, 1, NORMLEN, cn->tf_fd);   /* read next block from file */
         #endif
       if(bytes < NORMLEN)           /* end of file? */
       {
          #ifndef MCF52259_MST
           if(vferror(cn->tf_fd))    /* check if it is an error */
           {
              if(cn->callback)
                  cn->callback(TFC_FILEREAD, cn, "file read error");
              return FALSE;
           }
          #endif
                                     /* else at End Of File; fall through to do the last send */
       }
       cn->tf_flen = bytes;          /* bytes in last packet sent */
       cn->tf_size += bytes;         /* total bytes sent so far */
   }
…
```

5.2.2 Adding the TFTP Server Task in the NicheTask

Predefining the TFTP_SERVER to activate the TFTP server codes. Do not forget to comment out the macro defined for including VFS.

```
#ifdef TFTP_PROJECT
#define TFTP_SERVER       1  /* include TFTP server code */
//#define VFS_FILES       1  /* include Virtual File System */
#endif
```

5.3 Project Compile

Similar to the HTTP project, predefines must be executed before compiling this porject.

```
#define TFTP_PROJECT
#define MCF52259_MST
```

5.4 Running the TFTP Server on the MCF52259EVB

Before running the TFTP server, the EVB must be connected to the PC through the UART0 port. The baudrate of the HyperTerminal must be set to 115200-8N1. After power on reset, information is displayed as in Figure 3.

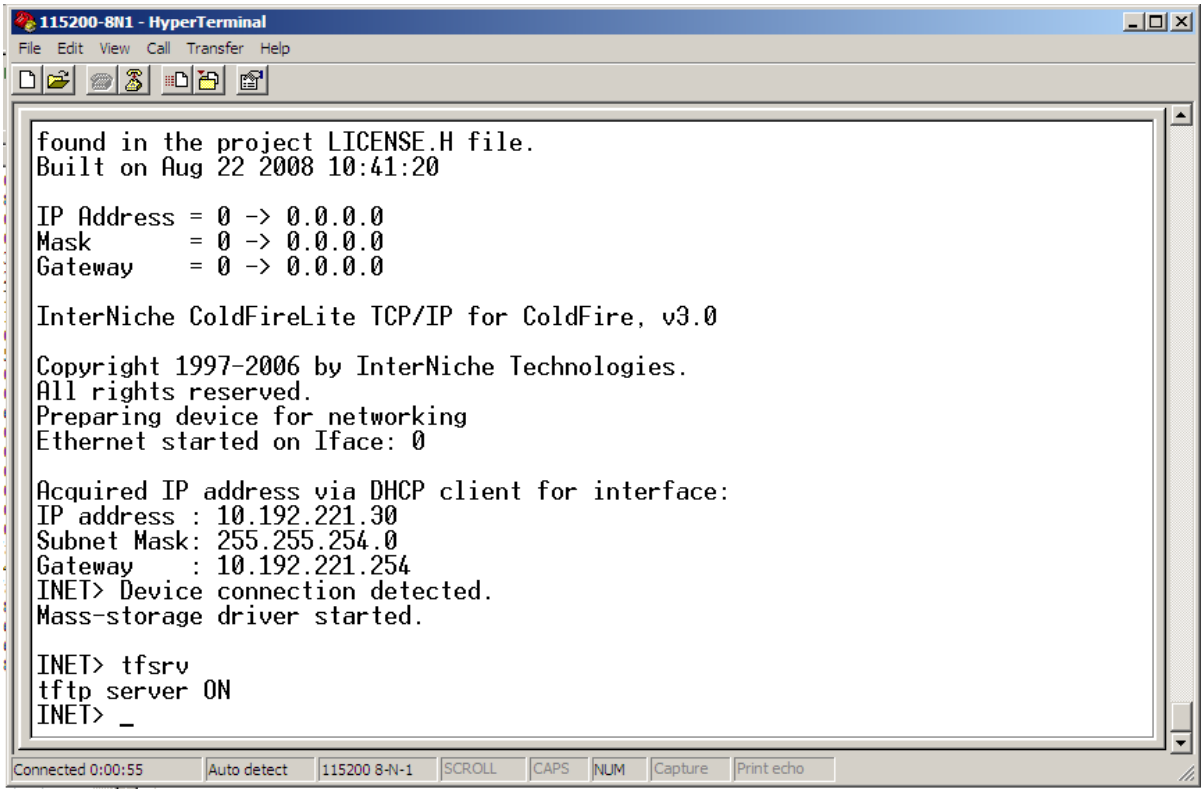

Figure 3. HyperTerminal

The TFTP server also has a DHCP client. It can obtain the IP address automatically. In Figure 3 the IP address for the server is 10.192.221.30. If a USB device is plugged in information is output. After, type the command tfsrv to activate the TFTP server.

5.5 Uploading a File to the TFTP Server

After the IP address for the TFTP server is obtained files can be uploaded to this server by DOS command tftp –i <dst_addr> put filename.xx. The dst_addr is 10.192.221.30 and the filename can be any file in the opened folder.

NOTE

Never forget the option –i.

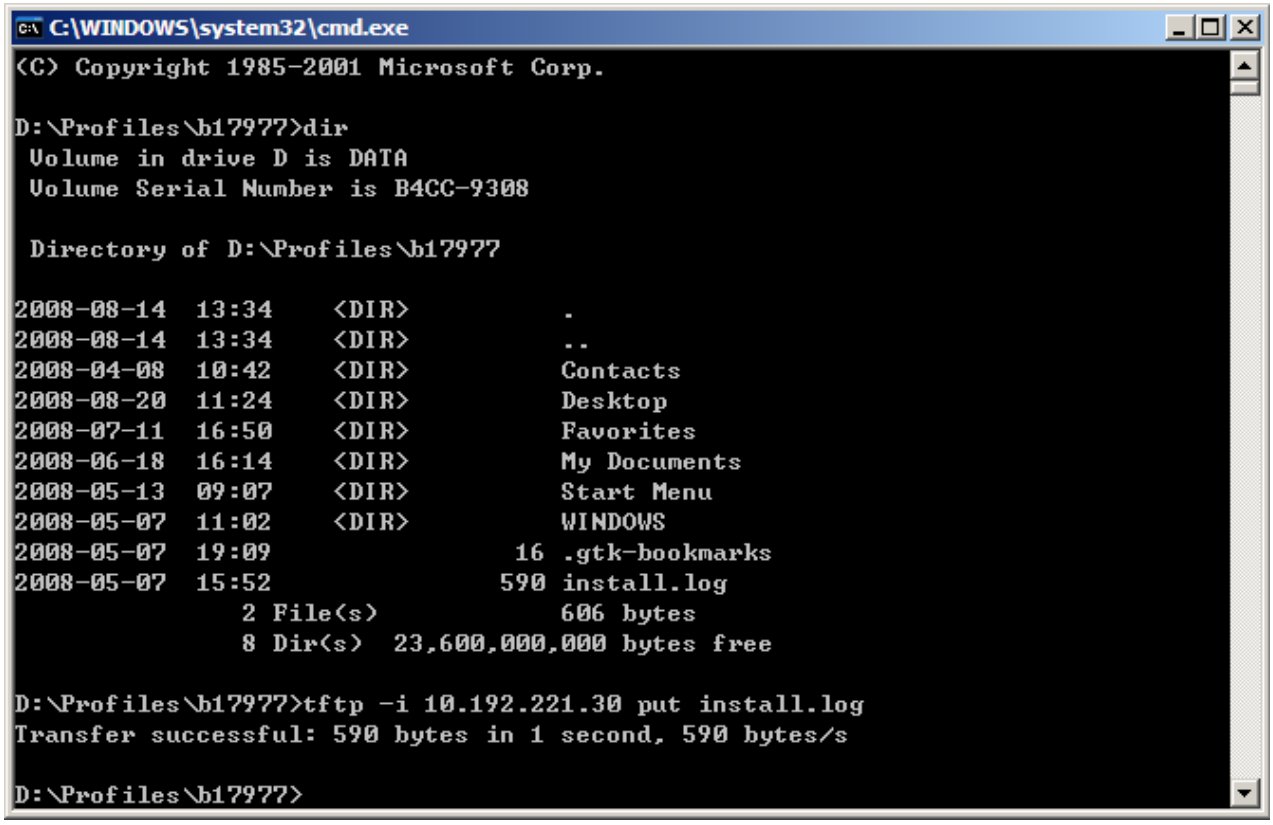

Figure 4. Uploading Files to the TFTP Server

5.6 Downloading a File from the TFTP Server

The same as uploading the file. Files can be downloaded from this server with a DOS command tftp –i <dst_addr> get filename.xx.

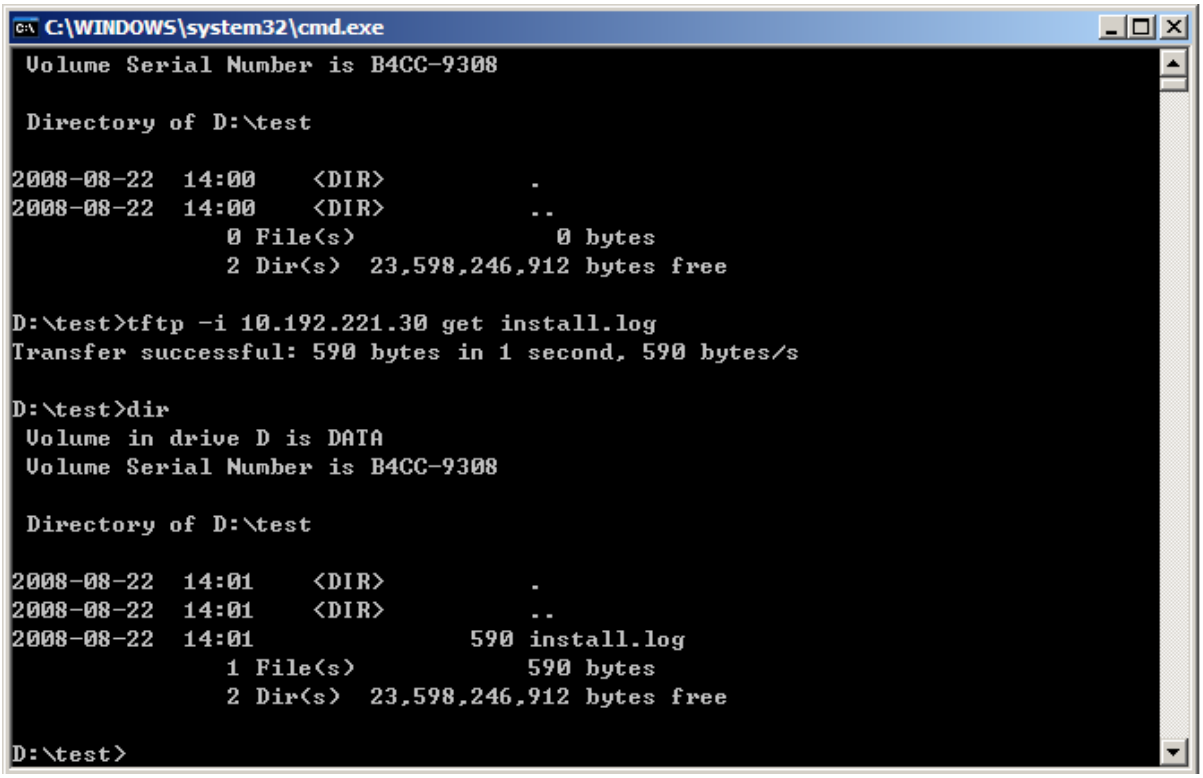

Figure 5. Downloading Files from the TFTP Server

6 Conclusion

The HTTP/TFTP server projects are implemented based on the FEC and USB module MCF52259 that provide not only high performance but also flexible connectivity. Applications based on the MCF52259 are not limited in the HTTP/TFTP server. If a USB camera is connected to the MCU instead of a flash a web monitor is achieved with minimal changes to the HTTP/TFTP server. It can monitor what is happening through the web page. The MCF52259 is ideal for applications in the industrial controlling field. Further developement regarding Network applications can be achieved functionality based on this application note.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2008. All rights reserved.

AN3779

Rev. 0
09/2008